

INSTITUTO TECNOLÓGICO DE CIUDAD MADERO

DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN



INFORME FINAL DE AÑO SABÁTICO

**B.1 ELABORACIÓN DE APUNTES/B.1.2 ELABORACIÓN DE APUNTES DE ASIGNATURA(S):
MATEMÁTICAS DISCRETAS E INGENIERÍA DE SOFTWARE**

Que presenta

Eurí Salgado Escobar

1127

Realizado en

Instituto Tecnológico de Ciudad Madero

Ciudad Madero, Tamaulipas

Durante el periodo

01/Febrero/2024 al 31/Enero/2025

Dictamen No. AS-1-044/2024



Ciudad de México, **23 de enero de 2024**

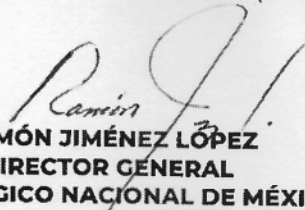
AUTORIZACIÓN No AS-1-044/2024

LUCIANO AGUILERA VÁZQUEZ
DIRECTOR DEL INSTITUTO TECNOLÓGICO
DE CIUDAD MADERO
PRESENTE

Con fundamento en el Reglamento Interior de Trabajo del Personal Docente de los Institutos Tecnológicos, en las Políticas Académicas Generales del Periodo Sabático y las bases de la Convocatoria correspondiente, el Tecnológico Nacional de México a través de la Comisión Central, ha revisado la solicitud del **Año Sabático** de la **C. EURI SALGADO ESCOBAR** en el Programa B. Elaboración de Apuntes, Libros, Objetivos Educativos y Reactivos de Evaluación: bajo la modalidad: **B.1 Elaboración de Apuntes/B.1.2 Elaboración de apuntes de asignatura(s): Matemáticas Discretas e Ingeniería de Software**, se tiene a bien emitir el siguiente dictamen de:

AUTORIZACIÓN

Del programa presentado a partir del **01 de febrero de 2024** hasta el **31 de enero de 2025**, quedando su seguimiento bajo la responsabilidad de la Dirección a su cargo, debiendo recibir y evaluar por parte de la Academia correspondiente y de la Comisión Dictaminadora, el reporte intermedio y final de las actividades desarrolladas durante el ejercicio del periodo sabático autorizado al docente. Así mismo, se deberá remitir a la Secretaría Académica, de Investigación e Innovación a través de la Dirección de Docencia e Innovación Educativa del Tecnológico Nacional de México, la evaluación de dichos reportes que comprenderán la totalidad de lo establecido en la calendarización, que se encuentra en el expediente de la solicitud del docente; además de un ejemplar en físico y en electrónico (en formato PDF) del producto final comprometido.


RAMÓN JIMÉNEZ LÓPEZ
DIRECTOR GENERAL
TECNOLÓGICO NACIONAL DE MÉXICO


SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL
DE MÉXICO
DIRECCIÓN GENERAL

M00-PR-04-R08

Versión 4



Av. Universidad 1200, Col. Xoco, Alcaldía Benito Juárez, C.P. 03330, Ciudad de México.
Tel. (55) 3601-7500, Ext. 65050, e-mail: direccion@tecnm.mx tecnm.mx



Dada la naturaleza de las operaciones y que su registro obra resguardado en los sistemas de mi representada, esta prueba se debe de practicar directamente sobre los registros obtenidos directamente del sistema de mi representada, mediante la inspección judicial asistida de peritos, dicha inspección deberá de llevarse a cabo en el **Módulo de Atención a Autoridades** de mi representada, ubicado en las oficinas principales de mi poderdante en Avenida Paseo de la Reforma No. 510, de la Colonia Juárez, Delegación Cuauhtémoc en la Ciudad de México, destacando la necesidad de llevar a cabo esta diligencia probatoria en aquella Ciudad en virtud de que es en el domicilio indicado en donde se concentran los registros y huellas de las operaciones realizadas a través de los canales digitales que mi representada pone a disposición de sus clientes.

Sin perjuicio de lo anterior y si fuera el caso de que su Señoría estimase gravosa la práctica de esta probanza en aquella Ciudad, se propone como opción alterna para el desahogo de este medio de convicción que la prueba pericial verse sobre los registros impresos y certificados en términos del Artículo 100 de la Ley de Instituciones de Crédito que se acompañan al presente escrito, de los cuales

Asimismo, mi representada manifiesta a su Señoría que el perito que en su caso llegare a designar el Actor, en el análisis de los sistemas electrónicos no podrá hacer uso de alguna cámara fotográfica o bien, a través de la utilización de cualquier otro medio mecánico, digital, electrónico, fotográfico o similar, ya que pone en riesgo información confidencial respecto de los demás clientes que tiene mi representada registrados en sus sistemas electrónicos, es por ello que desde este momento se solicita decrete que el perito queda impedido a usar y tomar fotografías al sistema y/o a los archivos electrónicos del sistema, amén de que la prueba pericial ofrecida por las partes en el presente juicio, no consiste en tomar fotografías, sino exclusivamente que el perito revise los sistemas de información que contienen los registros de las transacciones objeto de la litis del presente juicio realizadas por medio de la banca en línea haciendo uso del Internet.

Introducción

Matemáticas discretas es una asignatura que aporta al perfil del egresado los conocimientos lógico-matemáticos para entender, inferir, aplicar y desarrollar modelos matemáticos tendientes a resolver problemas en el área de las ciencias computacionales. Es el soporte para un conjunto de asignaturas que se encuentran vinculadas directamente con las competencias profesionales que se desarrollarán, por lo que se incluye en los primeros semestres de la trayectoria escolar. Aporta conocimientos a las asignaturas de Estructura de Datos y Redes de Computadoras.

En el primer tema se revisan procedimientos para realizar la conversión entre diferentes sistemas numéricos y realización de operaciones básicas: suma, resta, multiplicación y división. Los conceptos básicos de conjuntos y relaciones son revisados en el tema dos de la asignatura, donde se analizan propiedades y operaciones entre conjuntos, mismos que serán validados en los temas de álgebra booleana y lógica matemática.

El concepto de inducción matemática es abordado en forma particular dada su aplicación en el proceso de análisis y demostración de modelos matemáticos. La asignatura proporciona los conocimientos relacionados con la Teoría de grafos, árboles y redes.

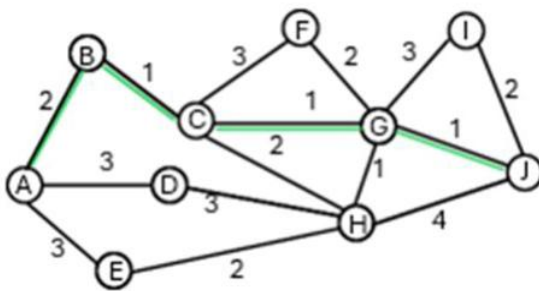
La asignatura de Ingeniería de software aplica una metodología para el desarrollo de software en un contexto multidisciplinario; se relaciona con materias precedentes como: Fundamentos de Programación, Programación Orientada a Objetos, Estructura de Datos, Fundamentos de Ingeniería de Software, Tópicos Avanzados de Programación, Sistemas Operativos, Arquitectura de Computadoras y Taller de Bases de Datos. También se relaciona con las materias posteriores Gestión de Proyectos de Software y Programación Web.

La asignatura debe ser teórico – práctico, capaz de desarrollar en el estudiante la habilidad para la aplicación de las diferentes técnicas en el desarrollo de software, considerando siempre los principios de la ingeniería de software, para lo cual se organiza el temario en cuatro temas.

En el tema uno, el Modelo de Análisis, se obtienen los requisitos del sistema, se realiza el estudio de factibilidad y un análisis de costos. A partir del modelo del análisis, se elabora un diseño de clases, de datos, diagramas de secuencia, así como el diseño de una interfaz de usuario. Posteriormente se construye el sistema y por último se realizan las Pruebas e Implantación. Las diferentes pruebas al sistema se realizan para verificar la eficiencia del sistema y garantizar la calidad del sistema. El papel del docente es ejemplificar cada etapa de la Ingeniería de software y propiciar el trabajo en equipo para atender proyectos del contexto que induzcan al estudiante a la aplicación de la metodología en un ambiente real.

**Apuntes de la
asignatura:**

Matemáticas discretas



1. Datos Generales de la asignatura

Nombre de la asignatura:	Matemáticas Discretas
Clave de la asignatura:	AEF-1041
SATCA¹:	3 - 2 - 5
Carrera:	Ingeniería Informática e Ingeniería en Sistemas Computacionales

2. Presentación

Caracterización de la asignatura

Esta asignatura aporta al perfil del egresado los conocimientos lógico-matemáticos para entender, inferir, aplicar y desarrollar modelos matemáticos tendientes a resolver problemas en el área de las ciencias computacionales.

Es el soporte para un conjunto de asignaturas que se encuentran vinculadas directamente con las competencias profesionales que se desarrollarán, por lo que se incluye en los primeros semestres de la trayectoria escolar. Aporta conocimientos a las asignaturas de Estructura de Datos y Redes de Computadoras con los conceptos básicos de Grafos y Árboles.

Intención didáctica

La asignatura se organiza en seis temas de aprendizaje. Los dos primeros temas abordan conceptos básicos que serán utilizados a lo largo de curso, mientras que los cuatro restantes analizan contenidos propios del área de las ciencias computacionales.

El primer tema se revisan los procedimientos para realizar la conversión entre diferentes bases numéricas, examinando las operaciones básicas: suma, resta, multiplicación y división, buscando que el estudiante analice y genere un procedimiento general de conversión entre bases numéricas.

Los conceptos básicos de conjuntos y relaciones son revisados en el tema dos de la asignatura, en ella se analiza las características, propiedades y operaciones entre conjuntos, mismos que serán validados en los temas de álgebra booleana y lógica matemática. Por otro lado, los tópicos concernientes a las relaciones se abordan por medio del conocimiento del producto cartesiano, enfatizando en las relaciones de tipo binario y su representación. Adicionalmente se revisan las propiedades, relaciones de equivalencia, órdenes parciales y funciones como casos particulares de relaciones.

Por su parte el tema tres, hace un análisis de la lógica proposicional con la finalidad de llegar a procesos de demostración formal, así mismo se examinan los conceptos de lógica de predicados y algebra declarativa. El concepto de inducción matemática es abordado en forma particular dada su aplicación en proceso de análisis y demostración de modelos matemáticos.

En el tema cuatro, se estudian los teoremas y postulados con operaciones básicas en la simplificación de expresiones booleanas.

¹ Sistema de Asignación y Transferencia de Créditos Académicos

El penúltimo tema de la asignatura, proporciona los conocimientos relacionados con la Teoría de grafos: conceptos básicos, representación, clasificación; así como los algoritmos de recorrido y búsqueda son analizados para su posterior aplicación.

Finalmente, el tema seis Árboles y redes son revisados como un caso especial de grafos.

Los contenidos se abordarán de manera secuencial como los marca el programa, buscando la aplicación del conocimiento, basado en actividades que promuevan en el estudiante el desarrollo de sus habilidades para trabajar en equipo y llevar el conocimiento a la práctica, buscando con ello que integre ese saber a su experiencia personal mediante un aprendizaje significativo.

La extensión y profundidad de los temas será la suficiente para garantizar que el estudiante logre las competencias señaladas oportunamente.

El docente además de ser un motivador permanente en el proceso educativo deberá ser promotor y director de la enseñanza a través de la transmisión del conocimiento, así como la aplicación de sus habilidades y destrezas utilizando las herramientas tradicionales y digitales a su alcance para cautivar a sus estudiantes e interesarlos en el tema.

3. Participantes en el diseño y seguimiento curricular del programa

Lugar y fecha de elaboración o revisión	Participantes	Evento
Instituto Tecnológico de Aguascalientes del 15 al 18 de junio de 2010.	Representantes de los Institutos Tecnológicos de: Centro Interdisciplinario de Investigación y Docencia en Educación Técnica, Acapulco, Aguascalientes, Apizaco, Boca Río, Celaya, Chetumal, Chihuahua, Chilpancingo, Chiná, Cd. Cuauhtémoc, Cd. Juárez, Cd. Madero, Cd. Victoria, Colima, Comitán, Cautla, Durango, El Llano de Aguascalientes, Huixquilucan, Valle Bravo, Guaymas, Huatabampo, Huejutla, Iguala, La Laguna, La Paz, La Zona Maya, León, Lerma, Linares, Los Mochis, Matamoros, Mazatlán, Mérida, Mexicali, Minatitlán, Nuevo Laredo, Orizaba, Pachuca, Puebla, Querétaro, Reynosa, Roque, Salina Cruz, Saltillo, San Luis Potosí, Tehuacán, Tepic,	Elaboración del programa de estudio equivalente en la Reunión Nacional de Implementación Curricular y Fortalecimiento Curricular de las asignaturas comunes por área de conocimiento para los planes de estudio actualizados del SNEST.

	Tijuana, Tlaxiaco, Toluca, Torreón, Tuxtepec, Valle de Oaxaca, Veracruz, Villahermosa, Zacatecas, Zacatepec, Altiplano de Tlaxcala, Coatzacoalcos, Cuautitlán Izcalli, Fresnillo, Irapuato, La Sierra Norte Puebla, Macuspana, Naranjos, Pátzcuaro, Poza Rica, Progreso, Puerto Vallarta, Tacámbaro, Tamazula Gordiano, Tlaxco, Venustiano Carranza, Zacapoaxtla, Zongólica y Oriente del Estado Hidalgo.	
Instituto Tecnológico de Morelia del 10 al 13 de septiembre de 2013.	Representantes de los Institutos Tecnológicos de: Aguascalientes, Apizaco, Boca del Río, Celaya, CRODE Celaya, Cerro Azul, Chihuahua, Cd. Cuauhtémoc, Cd. Hidalgo, Cd. Juárez, Cd. Madero, Cd. Valles, Coacalco, Colima, Iguala, La Laguna, Lerdo, Los Cabos, Matamoros, Mérida, Morelia, Motúl, Múzquiz, Nuevo Laredo, Nuevo León, Oriente del Estado de México, Orizaba, Pachuca, Progreso, Purhepecha, Salvatierra, San Juan del Río, Santiago Papasquiaro, Tantoyuca, Tepic, Tlatlauquitpec, Valle de Morelia, Venustiano Carranza, Veracruz, Villahermosa, Zacatecas y Zacatepec.	Reunión Nacional de Seguimiento Curricular de las Asignaturas Equivalentes del SNIT.

4. Competencia(s) a desarrollar

Competencia(s) específica(s) de la asignatura
Comprende y aplica los conceptos básicos de lógica matemática, relaciones, grafos y árboles para aplicarlos a modelos que resuelvan problemas computacionales.

5. Competencias previas

Ninguna

6. Temario

No.	Temas	Subtemas
1.	Sistemas numéricos	1.1 Sistemas numéricos (Binario, Octal, Decimal, Hexadecimal) 1.2 Conversiones entre sistemas numéricos 1.3 Operaciones básicas (Suma, Resta, Multiplicación y División) 1.4 Aplicación de los sistemas numéricos en la computación
2.	Conjuntos y relaciones	2.1 Características de los conjuntos y subconjuntos 2.2 Operaciones con conjuntos 2.3 Propiedades y aplicaciones de los conjuntos 2.4 Conceptos básicos: producto cartesiano y relación binaria 2.5 Representación de las relaciones 2.6 Propiedades de las relaciones 2.7 Relaciones de equivalencia 2.8 Funciones 2.9 Aplicaciones de las relaciones y las funciones en la computación
3.	Lógica matemática	3.1 Lógica proposicional 3.1.1 Propositiones simples y compuestas 3.1.2 Tablas de verdad 3.1.3 Tautologías, contradicción y contingencia 3.1.4 Equivalencias lógicas 3.1.5 Reglas de inferencia 3.1.6 Argumentos válidos y no válidos 3.1.7 Demostración formal 3.2 Lógica de predicados 3.2.1 Cuantificadores 3.2.2 Representación y evaluación de predicados 3.3 Algebra declarativa 3.4 Inducción matemática 3.5 Aplicaciones de la lógica matemática en la computación
4.	Algebra booleana	4.1 Teoremas y postulados 4.2 Optimización de expresiones booleanas 4.3 Aplicación del algebra booleana 4.3.1 Mini y maxi términos 4.3.2 Representación de expresiones booleanas con circuitos lógicos
5.	Teoría de grafos	5.1 Elementos, características y componentes de los grafos.

		5.1.1 Tipos de grafos 5.2 Representación de los grafos. 5.2.1 Matemática 5.2.2 Computacional 5.3 Algoritmos de recorrido y búsqueda. 5.3.1 El camino más corto 5.3.2 A lo ancho 5.3.3 En profundidad
6.	Árboles y redes	6.1 Árboles. 6.1.1 Componentes y propiedades 6.1.2 Clasificación por altura y número de nodos 6.2. Árboles con peso 6.2.1 Recorrido de un árbol 6.3 Redes. 6.3.1 Teorema de flujo máximo 6.3.2 Teorema de flujo mínimo 6.3.3 Pareos y redes de Petri

7. Actividades de aprendizaje de los temas

Sistemas numéricos	
Competencias	Actividades de aprendizaje
Específica(s): Comprende y aplica las conversiones entre los diferentes sistemas de numeración para su aplicación en problemas computacionales. Genéricas: • Capacidad de abstracción, análisis y síntesis • Capacidad de aplicar los conocimientos en la práctica • Conocimientos sobre el área de estudio y la profesión • Habilidades para buscar, procesar y analizar información procedente de fuentes diversas • Capacidad de trabajo en equipo	• Investigar en diferentes fuentes el concepto de sistema numérico, historia de los sistemas numéricos, utilidad y tipos de sistemas numéricos. • Discutir en equipos la información investigada para llegar a conclusiones por equipos y finalmente grupal. • Elaborar en equipos de trabajo un procedimiento general para convertir un número decimal a su equivalente en otro sistema numérico posicional. • Investigar los procedimientos para convertir del sistema binario a octal y hexadecimal, de octal a binario y hexadecimal, y de hexadecimal a binario y octal. • Representar y convertir cantidades en los sistemas numéricos: decimal, binario, octal y hexadecimal. • Por medio de una hoja electrónica de cálculo desarrollar un método para llevar a cabo conversiones entre sistemas posicionales. • Realizar operaciones básicas de suma, resta, multiplicación y división en los sistemas decimal, binario, octal y hexadecimal



	considerando como base los algoritmos investigados. • Por medio de una hoja electrónica de cálculo desarrollar un método que permita llevar a cabo operaciones aritméticas entre diferentes sistemas numéricos. • Realizar sumas de cantidades en binario usando para ello complemento a dos. • Realizar multiplicaciones y divisiones en binario usando el algoritmo de Booth. • • Elaborar un mapa conceptual del tema de sistemas numéricos.
Conjuntos y relaciones	
Competencias	Actividades de aprendizaje
Específica(s): Conoce y aplica las operaciones y propiedades de los conjuntos y relaciones para la resolución de problemas reales. Genéricas: • Capacidad de abstracción, análisis y síntesis • Capacidad de aplicar los conocimientos en la práctica • Conocimientos sobre el área de estudio y la profesión • Capacidad de comunicación oral y escrita • Capacidad de investigación • Habilidades para buscar, procesar y analizar información procedente de fuentes diversas • Capacidad de trabajo en equipo	• Investigar la definición, desarrollo histórico, características y propiedades de los conjuntos, operaciones entre conjuntos y aplicación de los conjuntos. • Elaborar un mapa conceptual donde se represente el producto de la investigación realizada. • Representar información del ambiente cotidiano utilizando conjuntos, resolver problemas con las operaciones de conjuntos (unión, conjunción, complemento, diferencias, conjunto potencia). • Investigar la representación de conjuntos y sus operaciones mediante Diagramas de Venn, en grupos de trabajo resolver problemas que muestren esta técnica, como una manera de ilustrar y comprender mejor la operación entre conjuntos. • Elaborar una lista de aplicaciones de los conjuntos en el área de la computación. • Investigar los conceptos de: producto cartesiano, relación y relación binaria. • Utilizando conjuntos, matrices y diagramas de flechas presentar ejemplos de relaciones. • Resolver ejercicios de las operaciones que pueden realizarse entre relaciones (unión, intersección, complemento, inversa y composición). • Construir ejemplos de manera individual que presente las propiedades de una relación usando su definición formal.

	• Resolver ejercicios donde una relación que no tenga la propiedad de equivalencia, adquiera está aplicando las cerraduras reflexiva, simétrica y transitiva. • Determinar cuándo una relación sea de orden parcial y determinar el diagrama de Hasse de dicha relación. • Realizar un cuadro comparativo entre una relación de equivalencia y una de orden parcial, identificando sus coincidencias y diferencias. • Elaborar un resumen con las aplicaciones de las relaciones de equivalencia y orden parcial en las ciencias computacionales. • Analizar los diferentes tipos de funciones (inyectiva, suprayectiva, biyectiva). • Presentar ejemplos del ambiente cotidiano donde se muestre el comportamiento de estas funciones, representar gráficamente los resultados.
Lógica matemática	
Competencias	Actividades de aprendizaje
Específica(s): Analiza y resuelve problemas computacionales utilizando las técnicas básicas de lógica e inducción matemática. Genéricas: • Capacidad de abstracción, análisis y síntesis • Capacidad de aplicar los conocimientos en la práctica • Conocimientos sobre el área de estudio y la profesión • Capacidad de comunicación oral y escrita • Capacidad de investigación • Habilidades para buscar, procesar y analizar información procedente de fuentes diversas • Capacidad de trabajo en equipo	• Investigar el concepto de argumento, proposición y proposición lógica. • Presentar ejemplos de proposiciones lógicas. • Elaborar un esquema con los tipos de conexiones lógicas, su representación y tabla de verdad. • Representar enunciados usando para ello notación lógica. • Analizar ejemplos de evaluación de proposiciones lógicas compuestas mediante tablas de verdad. • Construir la tabla de verdad de proposiciones lógicas compuestas propuestas como ejercicios. • Usar una herramienta computacional para desarrollar un método que permita elaborar tablas de verdad de proposiciones compuestas. • Identificar cuando una proposición es una tautología, contradicción y contingencia. • Obtener por medio de tablas de verdad proposiciones lógicamente equivalentes, tautologías, reglas de inferencia lógica, discutir los resultados en grupos de trabajo.

	• Investigar que es la inferencia lógica, sus silogismos y equivalencias lógicas y discutir en plenaria la información localizada para obtener conclusiones. • Resolver un problema de argumentos válidos y no válidos para determinar cuándo un argumento es válido o no, usando para ello tablas de verdad y reglas de inferencia. • Desarrollar ejercicios para la construcción de demostraciones formales utilizando silogismos. • Demostrar que dos proposiciones son lógicamente equivalentes apoyándose en las equivalencias lógicas. • Demostrar la validez de un teorema usando para ello la demostración formal por el método directo y el método por contradicción, apoyándose en tautologías, reglas de inferencia y equivalencias lógicas conocidas. • Representar enunciados usando para ello la lógica de predicados, operadores lógicos y cuantificadores. • Investigar el concepto de inducción matemática y el método de demostración por inducción. • Analizar en grupos de trabajo el proceso de resolución de problemas por el método inductivo. • Elaborar una lista de aplicaciones de la lógica matemática en la computación, justificando con argumentos válidos cada una de esas aplicaciones. • Elaborar un resumen individual donde se explique la relación que existe entre los elementos y conceptos de la lógica proposicional, de predicados, el álgebra declarativa y la inducción matemática.
Álgebra booleana	
Competencias	Actividades de aprendizaje
Específica(s): Aplica los conceptos y propiedades del álgebra booleana, para optimizar expresiones booleanas y diseñar circuitos básicos con compuertas lógicas.	• Investigar en grupos de trabajo el concepto, historia, postulados y propiedades del álgebra booleana. • En reunión plenaria, discutir el material investigado y llegar a una conclusión grupal.
Genéricas:	• Elaborar un mapa conceptual de los conceptos

• Capacidad de abstracción, análisis y síntesis. • Capacidad de aplicar los conocimientos en la práctica. • Conocimientos sobre el área de estudio y la profesión. • Capacidad de comunicación oral y escrita. • Habilidades para buscar, procesar y analizar información procedente de fuentes diversas. • Capacidad de trabajo en equipo. • Capacidad de investigación. • Habilidades en el uso de las tecnologías de la información y de la comunicación.	de álgebra booleana, las operaciones que se utilizan y las propiedades. • Resolver problemas de representación de expresiones booleanas usando para ello compuertas básicas (and, or, not y x-or). Obtener expresiones booleanas a partir de una tabla de verdad que muestre todos los posibles valores de un sistema lógico. • Usar software para representar expresiones booleanas por medio de compuertas lógicas. • Simplificar expresiones booleanas usando para ello teoremas del álgebra booleana. • Desarrollar ejercicios de optimización de expresiones booleanas, aplicando las propiedades del álgebra booleana. • Resolver problemas para obtener la expresión equivalente simplificada a partir de un circuito lógico (mapas de Karnaugh). • Construir circuitos lógicos utilizando compuertas lógicas.
Teoría de grafos	
Competencias	Actividades de aprendizaje
Específica(s): Aplica los conceptos básicos de grafos para resolver problemas afines al área computacional, relacionados con el recorrido, búsqueda y ordenamiento en grafos. Genéricas: • Capacidad de abstracción, análisis y síntesis. • Capacidad de aplicar los conocimientos en la práctica. • Conocimientos sobre el área de estudio y la profesión. • Capacidad de comunicación oral y escrita. • Habilidades para buscar, procesar y analizar información procedente de fuentes diversas. • Capacidad de trabajo en equipo. • Capacidad para organizar y planificar el tiempo. • Capacidad de investigación. • Habilidades en el uso de las tecnologías de la información y de la comunicación.	• Investigar los elementos y características de los grafos (vértice, arista, lazos, valencias, caminos) • Elaborar una presentación electrónica donde se identifiquen los conceptos básicos investigados. • Construir un esquema donde se muestren los diferentes tipos de grafos, sus características y ejemplos de cada uno de ellos. • Investigar y realizar un reporte cómo se representan los grafos utilizando matrices, identificar las razones por las cuales se utilizan cada una de las representaciones y cuál es la más adecuada para su manejo en la computadora. • Investigar los diferentes algoritmos para el cálculo del número de caminos en un grafo, así como el camino más corto, analizar sus características y determinar cuál es el más óptimo. • Investigar cuáles son las estrategias y algoritmos de búsqueda existentes, analizar los resultados en grupos de trabajo y presentar por



	escrito un resumen • Realizar ejercicios de grafos en la que se aplique búsqueda de información a lo ancho y en profundidad. • Usar software para determinar características, propiedades y recorridos en grafos.
Árboles y redes	
Competencias	Actividades de aprendizaje
Específica(s): Aplica la organización y relación entre los datos mediante procesos de ordenamiento, para resolver problemas de programación matemática donde se hace uso de las redes. Genéricas: • Capacidad de abstracción, análisis y síntesis • Capacidad de aplicar los conocimientos en la práctica • Capacidad de comunicación oral y escrita • Habilidades para buscar, procesar y analizar información procedente de fuentes diversas • Habilidades en el uso de las tecnologías de la información y de la comunicación • Capacidad de trabajo en equipo	• Elaborar una presentación electrónica con los conceptos básicos de árboles y sus propiedades. • Diferenciar los conceptos entre un grafo y un árbol. • Analizar en grupos de trabajo la clasificación de los árboles, presentar un resumen de resultados. • Investigar los procedimientos para realizar el recorrido de un árbol, así como el ordenamiento y la búsqueda de los elementos. • Resolver ejercicios para el recorrido de árboles en preorden, inorden y postorden. • Investigar las aplicaciones de los recorridos de árboles en el área de las ciencias computacionales. • Estructurar la información en un árbol para llevar a cabo evaluación de ecuaciones matemáticas y ordenamiento de información por medio de sus diferentes recorridos. • Resolver ejercicios de búsqueda a lo ancho y en profundidad, así como el ordenamiento de información utilizando árboles. • Realizar la decodificación de información por medio del método de Huffman. • Realizar un mapa mental para realizar la aplicación de flujo máximo, flujo mínimo y pareos. • Resolver ejercicios planteado por el profesor donde se apliquen los conceptos de flujo máximo, flujo mínimo, Pareos y Redes de Petri.

8. Práctica(s)

- Elaborar a través de una hoja electrónica de cálculo un proceso para la conversión y realización de operaciones aritméticas básicas de cantidades en diferentes bases numéricas.
- Buscar en Internet software que permita llevar a cabo ejercicios de conversión, operaciones matemáticas básicas (suma, resta, multiplicación y división) en diferentes sistemas numéricos, utilizarlo para resolver problemas planteados en clase.
- Utilizando herramientas computacionales disponibles para el estudiante, representar el comportamiento de las operaciones con conjuntos mediante diagramas de *Venn*.
- Elaborar, con ayuda de una hoja electrónica de cálculo, un proceso para llevar a cabo la evaluación de una proposición compuesta mediante tablas de verdad.
- Utilizando un simulador, verificar el comportamiento de una expresión proposicional.
- Utilización de diagramas de *Venn* para la determinación de razonamiento.
- Construir un circuito usando compuertas lógicas, implementarlas utilizando software para la construcción de circuitos electrónicos.
- Utilizando software disponible para el estudiante, determinar las propiedades de una relación, aplicar cerraduras para lograr que una relación sea de equivalencia y determinar el diagrama de *Hasse* de relaciones de orden parcial.
- Representar un grafo utilizando una hoja electrónica de cálculo, y obtener el número de caminos de longitud n mediante el cálculo correspondiente.
- Mediante software disponible para el estudiante, determinar características, propiedades y recorridos importantes en un grafo.
- Realizar el recorrido de un árbol que represente una expresión matemática y obtener su valor usando para ello el concepto de pila para almacenar resultados.
- Crear un árbol binario a partir de una lista de números aleatorios y llevar a cabo búsquedas y ordenamiento de dichos datos.
- Usar software disponible para el estudiante, con el cual se simule el recorrido, búsqueda de información, representación y evaluación de un árbol.

9. Proyecto de asignatura

El objetivo del proyecto que plantee el docente que imparta esta asignatura, es demostrar el desarrollo y alcance de la(s) competencia(s) de la asignatura, considerando las siguientes fases:

- **Fundamentación:** marco referencial (teórico, conceptual, contextual, legal) en el cual se fundamenta el proyecto de acuerdo con un diagnóstico realizado, mismo que permite a los estudiantes lograr la comprensión de la realidad o situación objeto de estudio para definir un proceso de intervención o hacer el diseño de un modelo.
- **Planeación:** con base en el diagnóstico en esta fase se realiza el diseño del proyecto por parte de los estudiantes con asesoría del docente; implica planificar un proceso: de intervención empresarial, social o comunitaria, el diseño de un modelo, entre otros, según el tipo de proyecto, las actividades a realizar los recursos requeridos y el cronograma de trabajo.
- **Ejecución:** consiste en el desarrollo de la planeación del proyecto realizada por parte de los estudiantes con asesoría del docente, es decir en la intervención (social, empresarial), o construcción del modelo propuesto según el tipo de proyecto, es la fase de mayor duración que implica el desempeño de las competencias genéricas y específicas a desarrollar.
- **Evaluación:** es la fase final que aplica un juicio de valor en el contexto laboral-profesión, social e investigativo, ésta se debe realizar a través del reconocimiento de logros y aspectos a mejorar se

estará promoviendo el concepto de “evaluación para la mejora continua”, la metacognición, el desarrollo del pensamiento crítico y reflexivo en los estudiantes.

10. Evaluación por competencias

Para evaluar las actividades de aprendizaje se recomienda solicitar: mapas conceptuales, reportes de investigación, exposiciones en clase, problemario, portafolio de evidencias, tablas comparativas, glosarios, entre otros.

Para verificar el nivel del logro de las competencias del estudiante se recomienda utilizar: listas de cotejo, listas de verificación, matrices de valoración, rúbricas, entre otros, mediante la heteroevaluación, coevaluación y autoevaluación.

11. Fuentes de información

- 1 Aho, Alfred; Ullman, Jeffrey. (1996) Foundations of Computer Sciences. Editorial W. H. Freeman.
- 2 Anderson, James. (2001). Discrete Mathematics with Combinatorics. Prentice Hall.
- 3 Anderson, Ian. First Course in Discrete Mathematics. Springer.
- 4 Baase, Sara. (1998). Computer Algorithms. Addison Wesley. Graham, Knuth,
- 5 Brookshear, J. Glenn. (1993). “Teoría de la computación”. USA: Addison-Wesley/Iberoamericana.
- 6 García Valle, J. Luis. (1993). “Matemáticas especiales para computación”. México: McGraw-Hill.
- 7 Grassmann, Winfried Karl. Tremblay, Jean-Paul. (1997). “Matemática Discreta y Lógica, una perspectiva desde la ciencia de la computación”. España: Prentice Hall. España.
- 8 Grimaldi y Ralph P. (1998). “Matemáticas discreta y combinatoria” (3ra. Ed.). México: Pearson Educación.
- 9 Jiménez, J. (2008). “Matemáticas para la computación”. (1ra. Ed.). México: Alfaomega.
- 10 Johnsonbaugh, R. (2005). “Matemáticas Discretas”. (6ta. Ed.). México: Pearson Educación.
- 11 Rosen, H. (2004). “Matemática Discreta y sus aplicaciones”. (5ta. Ed.). Edición. España: McGraw-Hill.
- 12 Ross, Kenneth, A. y Wright, R. (2003). “Discrete mathematics”. (5ta. Ed.). U.S.A.: Pearson Education.
- 13 Skreumar, D. (2005). “Fundamental Approach to Discrete Mathematics”. USA : New Age International Publisher.
- 14 Suppes, P. y Hill, S. (2008). “Primer Curso de Lógica Matemática”. (3ra. Ed.). España: Reverté.
- 15 Tremblay, J. (1996). “Matemáticas discretas. Con aplicación a las ciencias de la computación”. México: CECSA.
- 16 Barceló, A. (2007). ¿Qué tan matemática es la lógica matemática? Disponible desde Internet en: <http://dianoia.filosoficas.unam.mx/info/2003/d51-Barcelo.pdf> [con acceso el 1 de Febrero de 2010]
- 17 Universidad Autónoma de México. (2006) Matemáticas IV (Matemáticas Discretas). México. Disponible desde Internet en: http://fcaenlinea.unam.mx/apuntes/interiores/docs/98/6/mate_4.pdf [Con acceso el 4 de enero de 2010].
- 18 Instituto Tecnológico de Buenos Aires. (2001) Matemática Discreta. Argentina. [Web en línea]. [con acceso el 8 de enero de 2010] <http://www.allaboutcircuits.com/> Fecha desconocida. All About Circuits. USA [Web en línea]. [con acceso el 8 de enero de 2010].

Matemáticas discretas

Tema 1	17
1.1 Sistemas numéricos	18
1.2 Conversiones entre sistemas numéricos	28
1.3 Operaciones básicas (suma, resta, multiplicación y división)	33
1.4 Aplicación de los sistemas numéricos en la computación	41
Tema 2	42
2.1 Características de los conjuntos y subconjuntos	43
2.2 Operaciones con conjuntos	46
2.3 Propiedades y aplicaciones de los conjuntos	49
2.4 Conceptos básicos: producto cartesiano y relación binaria	53
2.5 Representación de las relaciones	56
2.6 Propiedades de las relaciones	58
2.7 Relaciones de equivalencia	61
2.8 Funciones	63
2.9 Aplicaciones de las relaciones y las funciones en la computación	66
Tema 3	67
3.1 Lógica proposicional	68
3.1.1 Proposiciones simples y compuestas	68
3.1.2 Tablas de verdad	73
3.1.3 Tautologías, contradicción y contingencia	75
3.1.4 Equivalencias lógicas	75
3.1.5 Reglas de inferencia	76
3.1.6 Argumentos válidos y no válidos	82
3.1.7 Demostración formal	82
3.2 Lógica de predicados	83
3.2.1 Cuantificadores	83
3.2.2 Representación y evaluación de predicados	84
3.3 Álgebra declarativa	84
3.4 Inducción matemática	84
3.5 Aplicaciones de la lógica matemática en la computación	85
Tema 4	87

4.1 Teoremas y postulados	90
4.3 Optimización de expresiones booleanas.....	92
4.4 Aplicación del álgebra booleana	93
4.4.1 Minitérminos y maxitérminos	94
4.3.2 Representación de expresiones booleanas con circuitos lógicos.....	95
Tema 5	99
5.1 Elementos, características y componentes de los grafos	101
5.1.1 Tipos de grafos.....	103
5.2 Representación de los grafos.....	104
5.2.1 Representación matemática de un grafo.....	104
5.2.2 Representación computacional de un grafo.....	106
5.3 Algoritmos de recorrido y búsqueda	107
5.3.1 El camino más corto	107
5.3.2 Búsqueda a lo ancho.....	111
5.3.3 Búsqueda en profundidad	111
Tema 6	113
6.1 Árboles.....	114
6.1.1 Componentes y propiedades.....	114
6.1.2 Clasificación por altura y número de nodos	116
6.2 Árboles con peso	118
6.2.1 Recorrido de un árbol	123
6.3 Redes	127
6.3.1 Teorema de flujo máximo	128
6.3.2 Teorema de flujo mínimo	128
6.3.3 Pareos y redes de Petri	128

.

Tema 1

Sistemas numéricos

Desde los inicios de la humanidad, el ser humano necesitó una forma de contar y expresar cantidades de los objetos que se encontraban a su alrededor; para contar, el hombre utilizó objetos como palos, piedras, conchas, frutos, nudos en una cuerda y hasta contar con los dedos de sus manos. Considerando las limitantes de estos métodos, se vieron en la necesidad de encontrar nuevas alternativas para el conteo; así surgieron los sistemas de numeración.

Las culturas ancestrales buscaron nuevas formas de resolver sus necesidades de cálculo. Los pueblos antiguos como los egipcios, babilonios, chinos, griegos, romanos hicieron sus aportaciones en la historia de los sistemas numéricos. El sistema de numeración que se utiliza actualmente fue iniciado por los hindúes. Los árabes en sus viajes comerciales por la India encontraron un libro sobre aritmética escrito en hindú, este libro fue llevado a Europa. Debido a que el libro que introdujo los números a Europa provenía de Arabia, a los números se les llama números arábigos. Los sistemas de numeración surgieron por la necesidad de otorgar un símbolo a los objetos que se quieren contar y también para conocer un valor al realizar un trueque o intercambio comercial. Los sistemas de numeración se clasifican en: sistemas de numeración posicionales y sistemas de numeración no posicionales. En este tema se profundiza el conocimiento sobre los sistemas numéricos posicionales y su importancia para el funcionamiento de las computadoras.

Competencia específica: Comprende y aplica las conversiones entre los diferentes sistemas de numeración para su aplicación en problemas computacionales.

1.1 Sistemas numéricos

Introducción

En las actividades rutinarias de trabajo o en cualquier ámbito de nuestra vida, como la ciencia, la tecnología, los negocios, la medicina, la educación, los deportes, las actividades sociales y comerciales etc., se utilizan constantemente cantidades. Cada cantidad representa un valor que se mide, monitorea, se manipula en forma aritmética, se observa, se registra, se clasifica, se organiza, etc.

Existen dos formas de representar un valor numérico de las cantidades:

- a) **Representación analógica.** En la representación analógica, una cantidad oscila en un intervalo de valores, por ejemplo: los velocímetros de los automóviles, los termómetros con columna de mercurio, los relojes de manecillas. En un velocímetro que muestra la velocidad a la cual se desplaza un vehículo, la velocidad se encuentra representada por una aguja que cambia de posición al incrementar o decrementar la velocidad en el automóvil. La deflexión de la aguja es proporcional a la velocidad del automóvil y sigue cualquier cambio que se produzca a medida que el vehículo aumente o reduzca su velocidad; la posición de la aguja que indica la velocidad del automóvil no es precisa, es una aproximación a la velocidad real del automóvil.
- b) **Representación digital.** En una representación digital, las cantidades son fijas y bien definidas. Cada cantidad se representa por un símbolo llamado dígito, y el valor para pasar de un dígito a otro cambia en un incremento fijo llamado incremento discreto. Cada incremento en el sistema digital es representado por un dígito distinto, los dígitos son numerables y se distinguen entre ellos porque cada valor tiene una representación única. A estos dígitos que son numerables y que tienen una representación única se les llama valores discretos.

Muchas de las cosas que percibimos en la naturaleza y que podemos medir cuantitativamente como el clima, el tiempo, la distancia, el sonido, existen en forma analógica, una cantidad analógica puede ser interpretada por infinitos valores, pues los valores son continuos, al medir queda un espacio abierto para la interpretación. Cuando se mide una cantidad analógica, el valor “se redondea” o se aproxima a un nivel conveniente de precisión.

En cambio, una cantidad digital se representa por un conjunto de símbolos específicos, no puede tomar cualquier valor, únicamente aquellos valores que pertenecen al conjunto numerable que representan los símbolos del sistema de numeración. Esta cualidad de una cantidad digital es la razón por la cual es propicia para ser utilizada como base en el funcionamiento de las computadoras.

Sistema digital

Es la combinación de dispositivos diseñados para manipular información lógica o cantidades físicas que se representan en forma digital; es decir, las cantidades que sólo pueden tener

valores discretos. Algunos sistemas digitales son la computadora, equipos electrónicos de audio, video digital y sistema telefónico digital. Cabe mencionar que muchos de los dispositivos que existen hoy en día iniciaron como sistemas analógicos, pero con el mejoramiento de la tecnología ahora existen también en forma digital, como los relojes que existen tanto relojes analógicos como relojes digitales, radios, televisiones, equipos de audio, teléfonos celulares, etc. Cada vez más las aplicaciones en electrónica y otras tecnologías utilizan técnicas digitales para hacer operaciones que antes se realizaban mediante métodos analógicos. Los sistemas digitales requieren que las cantidades se representen como valores discretos. En la representación digital, las cantidades se representan mediante símbolos llamados dígitos, por ejemplo, el reloj digital indica la hora en forma de dígitos que representan la hora, los minutos y en ocasiones los segundos. Las horas avanzan en forma continua, el tiempo no se detiene, pero el despliegue de la hora en un reloj digital cambia en incrementos discretos de tiempo, hora por hora, minuto por minuto, segundo por segundo.

Los sistemas numéricos digitales

En la tecnología digital actual, se utilizan diferentes sistemas numéricos; dependiendo de las características de los sistemas de numeración es su utilidad, los más comunes son: sistema decimal, sistema binario, sistema octal y sistema hexadecimal.

Sistema de numeración decimal

El sistema de numeración decimal está formado por 10 números, dígitos o símbolos. Estos símbolos son: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Con estos símbolos como dígitos de un número, se puede expresar cualquier cantidad, el sistema de numeración también se conoce como sistema base 10. El sistema ha evolucionado en forma natural, pues el ser humano tiene 10 dedos, lo que facilita la actividad de contar hasta 10 (Tocci, 2007).

Una característica importante del sistema decimal es que es un sistema de numeración posicional, en el cual el valor de un dígito depende de su posición además de su propio valor. Ejemplo:



El número 249 tiene 2 centenas, 4 decenas y 9 unidades. El dígito que está más hacia la izquierda en el número 249 y que es diferente de cero, en este caso es el 2, es el dígito con el mayor valor en el número porque tiene un valor de 200, esto es debido a que ocupa el lugar de las centenas. Una cantidad digital se puede expresar de diferentes formas.

Ejemplo

El número 27.38 equivale a: 2 decenas, 7 unidades, 3 décimas y 8 centésimas. El punto decimal separa la parte entera de la parte fraccionaria.

Otra forma de expresar el mismo número es: $2 \times 10 + 7 \times 1 + 3 \times 0.1 + 8 \times 0.01$

¿De qué otra forma se puede expresar este número?

27.38

El punto decimal separa la parte entera de la parte fraccionaria

Las diversas posiciones que tiene un número decimal se consideran a partir del punto decimal, éstas tienen un peso el cual se puede expresar como potencias de 10.

Ejemplo:

03827.915₁₀

Expresado en potencias de 10:

$$(3 \times 10^3) + (8 \times 10^2) + (2 \times 10^1) + (7 \times 10^0) + (9 \times 10^{-1}) + (1 \times 10^{-2}) + (5 \times 10^{-3})$$

$$03827.915_{10} = 3 \times 1000 + 8 \times 100 + 2 \times 10 + 7 \times 1 + 9 \times 0.1 + 1 \times 0.01 + 5 \times 0.001$$

En general, cualquier número es la suma de los productos del valor de cada dígito y su valor posicional. Los números que son 0 y se encuentran a la izquierda de un número no tienen valor pues cualquier valor del peso al multiplicarse por cero se obtiene como resultado cero.

El número decimal 03827.915, en potencias de 10 se representa:

10 ⁴	10 ³	10 ²	10 ¹	10 ⁰		10 ⁻¹	10 ⁻²	10 ⁻³
0	3	8	2	7	.	9	1	5

MSD

El punto decimal

LSD

El valor de cada posición

El conteo decimal

Una de las primeras actividades aprendidas por el ser humano es contar, con el objetivo de conocer cuántos objetos se tienen (Fernández, 2022). Existen algunos principios importantes al momento de contar:

- Al contar se empieza en 0
- Está definido el orden de cada dígito
- No hay ambigüedad en la secuencia
- No existe ningún número intermedio entre dos dígitos consecutivos
- Los dígitos consecutivos se pueden expresar en un rango de valores como 0..9, pues el orden y los símbolos que pertenecen al conjunto no cambian



La base del sistema decimal consta de 10 dígitos o símbolos diferentes, con los cuales se puede representar cualquier cantidad. ¿Cómo se establece la secuencia de los dígitos? ¿Cómo es posible que con 10 dígitos se puedan representar un número infinito de cantidades?. El primer valor posicional que se tiene en un número con una sola cifra se inicia en la posición de las unidades, que empieza en 0 y se incrementa dígito por dígito en el orden establecido hasta llegar a 9. Debido a que el número máximo de símbolos utilizados es 10 y al llegar a 9 se han utilizado todos los dígitos, entonces es necesario agregar una nueva cifra

hacia la izquierda del número inicial, que tiene un valor posicional de 10, es decir, se agregan decenas al número, cada número que aparezca en dicha posición tiene un valor de 10, nuevamente el conteo comienza en 0 y hasta llegar al nueve, y en el momento en el que tanto las unidades como las decenas llegan a su máximo símbolo, es necesario agregar una nueva cifra (las centenas). Y así sucesivamente (Figura 1.1)

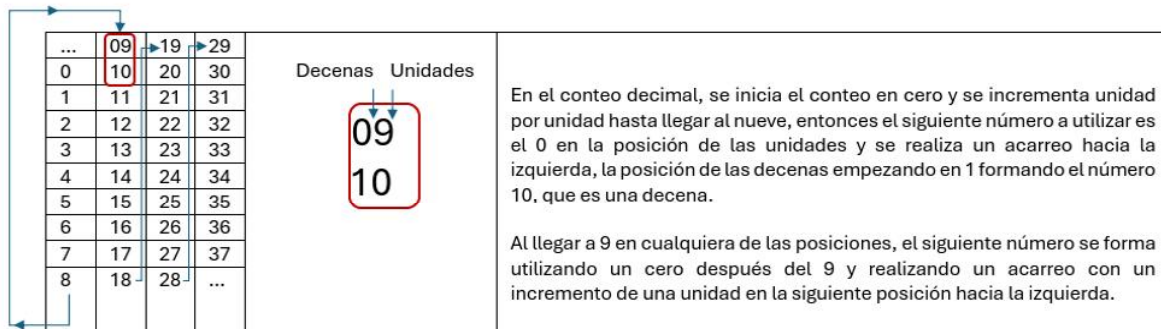


Figura 1.1 El conteo decimal

En el conteo decimal, la posición de las unidades, que es el dígito menos significativo (LSD) en la parte entera del número (primer dígito que se encuentra a la izquierda del punto decimal), este número se incrementa en una unidad en cada paso del conteo. El dígito en la posición de las decenas se incrementa en una unidad cada 10 incrementos del dígito que se encuentra en la posición de las unidades. El dígito en la posición de las centenas se incrementa en una unidad cada 10 incrementos del dígito que se encuentra en la posición de las decenas.

La cantidad de números que se pueden representar en un número decimal depende de la cantidad de dígitos que contenga el número, es decir, la cantidad de posiciones en el número.

9				Con 1 lugar se representan 10 números diferentes (10^1 , del 0 al 9)
9	9			Con 2 lugares se representan 100 números (10^2 , del 0 al 99)
9	9	9		Con 3 lugares se representan 1000 números (10^3 , del 0 al 999)
9	9	9	9	Con 4 lugares se representan 10000 números (10^4 , del 0 al 9999)

Cada una de las cantidades que se pueden expresar, debe estar representada por un número diferente, se pueden representar infinita cantidad de números agregando dígitos a la izquierda del número. Para representar un número, se necesitan:

10^N números distintos ($0..10^N - 1$)

Donde N es la cantidad de posiciones del número que se quiere representar

El sistema decimal es un sistema de numeración adecuado para el ser humano, pues su base 10 coincide con la cantidad de dedos en sus manos. Sin embargo, el sistema numérico decimal no es el apropiado para una implementación conveniente en los sistemas digitales. Es muy difícil diseñar equipo electrónico de forma que pueda trabajar con 10 niveles de voltaje distinto, por otro lado, es muy apropiado diseñar circuitos electrónicos simples y precisos que operen sólo con dos niveles de voltaje, los cuales representan la ausencia o presencia de

energía eléctrica. Por esta razón, casi cualquier sistema digital utiliza el sistema binario como el sistema numérico básico para realizar sus operaciones.

Sistema de numeración binario

El sistema binario está formado por 2 números, dígitos o símbolos llamados comúnmente como dígito binario, es muy común que el término dígito binario (**binary digit**) se abrevie como bit. Estos 2 valores o símbolos son 0 y 1. También se le denomina sistema base 2 y puede representar cualquier cantidad que pueda representarse en el sistema decimal o en cualquier otro sistema de numeración (aunque requiere un mayor número de dígitos binarios para expresar una cantidad determinada). Como en el sistema binario, los únicos dos dígitos que pueden formar parte del número son 0 y 1 (ningún otro dígito), se facilitan las operaciones y cálculos que se pueden llevar a cabo en la computadora, es el sistema de numeración apropiado para implementarse en un dispositivo electrónico.

Al igual que el sistema decimal, el sistema de numeración binario es un sistema de numeración posicional, esto es, el valor de cada dígito que forma parte del número binario depende de la posición que ocupa en el número y de la cantidad misma del dígito binario. Mediante un punto “.”, se separa la parte entera de la parte fraccionaria.

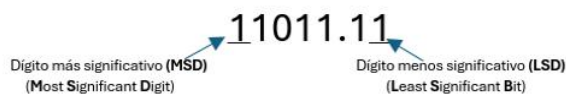
Por ejemplo:

El número binario 11011.11_2 consta de 7 dígitos, cada uno de los dígitos tiene un valor diferente de acuerdo con la posición que ocupa en el número. La base del sistema binario es 2 y cada una de las posiciones en el dígito es una potencia de 2. Expresado el número como potencias de 2, se puede conocer el número equivalente en decimal.

$$11011.11_2 = (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})$$

$$11011.11_2 = 16 + 8 + 0 + 2 + 1 + 0.50 + 0.25$$

$$11011.11_2 = 27.75_{10}$$



Los números que son 0 a la izquierda de un número no tienen valor pues cualquier valor del peso al multiplicarse por cero se obtiene como resultado cero.

El valor posicional de un número binario se expresa como potencia de 2:

2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
16	8	4	2	1		0.50	0.25	0.125
1	1	0	1	1	.	1	1	0

Labels and arrows: MSD points to the first '1' in the bottom row; El punto binario points to the decimal point; LSD points to the last '0' in the bottom row. Arrows from the top right point to the header row with the label "El valor de cada posición".

El conteo binario

Así como existe un orden en la secuencia de conteo decimal, de la misma forma existe un orden para contar en binario. En el sistema binario, sólo hay dos valores posibles 0 y 1. También se llama sistema de numeración base 2 y con estos dos dígitos es posible representar cualquier cantidad. Para diferenciar los números expresados en los diferentes sistemas de numeración es necesario distinguirlos a través de un subíndice que indica la base del sistema de numeración que utiliza, esta convención evita la confusión al emplear más de un sistema numérico. Las aseveraciones del sistema decimal aplican al sistema binario así también a los sistemas octal y hexadecimal, en general a todos los sistemas de numeración posicionales. El sistema binario, es un sistema de numeración posicional, en donde cada dígito binario (bit) tiene su propio peso expresado como potencia de 2. Los lugares a la izquierda del punto “.” son potencias positivas de 2, mientras que los lugares a la derecha del punto son potencias negativas de 2.

Secuencia de conteo de números en binario

Cantidad en decimal		Número binario equivalente			
		Pesos de cada posición			
		2^3	2^2	2^1	2^0
		8	4	2	1
0		0	0	0	0
1		0	0	0	1
2		0	0	1	0
3		0	0	1	1
4		0	1	0	0
5		0	1	0	1
6		0	1	1	0
7		0	1	1	1
8		1	0	0	0
9		1	0	0	1
10		1	0	1	0
11		1	0	1	1
12		1	1	0	0
13		1	1	0	1
14		1	1	1	0
15		1	1	1	1

La cantidad de números que se pueden representar en un número binario depende de la cantidad de dígitos que contenga el número, es decir, la cantidad de posiciones en el número. Observe cuidadosamente la tabla con los números binarios del 0 al 15.

1			
1	1		
1	1	1	
1	1	1	1

Con 1 lugar se representan 2 números distintos (2^1 lugares, el 0 y 1)
 Con 2 lugares se representan 4 números distintos (2^2 lugares, del 0 al 3)
 Con 3 lugares se representan 8 números distintos (2^3 lugares, del 0 al 7)
 Con 4 lugares se representan 16 números distintos (2^4 lugares, del 0 al 15)

Cada cantidad que se puede expresar debe estar representada por un número diferente y es posible representar una cantidad infinita de números en binario agregando a la izquierda los dígitos que sean necesarios para su representación.

La cantidad de bits que se requieren para representar un número decimal es 2^N números distintos ($0..2^N - 1$). Donde N representa la cantidad de lugares o posiciones en el número binario.

Sistema de numeración octal

El sistema octal está formado por 8 dígitos que son: 0, 1, 2, 3, 4, 5, 6, 7, que tienen el mismo valor que en el sistema decimal, pero a partir del número 8, que es la base de este sistema de numeración, se comienzan a repetir los dígitos agregando cifras a la izquierda del número, la base de un sistema de numeración (en este caso la base es 8), no forma parte de los dígitos que conforman un número octal. Este sistema es muy utilizado en computación por tener una base que es divisible entre 2, esta característica hace que la conversión de binario a octal y de octal a binario sea muy simple. El sistema octal es un sistema de numeración posicional, donde el valor cada posición es una potencia de 8.

Por ejemplo, el número 13612.44_8 expresa cada uno de sus dígitos de acuerdo con el valor de cada una de sus posiciones:

8^4	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}
4096	512	64	8	1		0.125	0.0156
1	3	6	1	2	.	4	4

MSD

El punto octal

LSD

El valor de cada posición

Por ejemplo, en el número 13612.44_8 cada uno de los dígitos tiene un valor diferente de acuerdo con la posición que ocupa en el número. La base del sistema octal es 8 y cada una de las posiciones en el número es una potencia de 8. Expresado el número como potencias de 8, se puede conocer el número equivalente en decimal.

$$13612.44_8 = (1 \times 8^4) + (3 \times 8^3) + (6 \times 8^2) + (1 \times 8^1) + (2 \times 8^0) + (4 \times 8^{-1}) + (4 \times 8^{-2})$$

$$13612.44_8 = 4096 + 1536 + 384 + 8 + 2 + 0.50 + 0.0624$$

$$13612.44_8 = 6026.5624_{10}$$

13612.44

Dígito más significativo (MSD)
(Most Significant Digit)

Dígito menos significativo (LSD)
(Least Significant Bit)

El conteo octal

El conteo en octal empieza en 0 y está definido el orden de cada dígito, por lo que no hay ambigüedad en la secuencia. No existe ningún número intermedio entre dos dígitos consecutivos. Los dígitos que forman parte del sistema octal se pueden expresar como un rango de valores de 0..7. En el sistema octal, existe un orden o secuencia de conteo, como en el sistema decimal y binario, también se tiene que, al utilizar N lugares o dígitos en el número codificado en octal, se pueden realizar 8^N conteos, empezando desde el 0 y hasta llegar a 8^{N-1} .

...	07	17	27
0	10	20	30
1	11	21	31
2	12	22	32
3	13	23	33
4	14	24	34
5	15	25	35
6	16	26	36

Anexar un dígito

07
10

¿Qué sigue del número 7 en octal? Al llegar a 7 en el conteo octal, se requiere agregar una segunda cifra del lado izquierdo del 7, y la primera cifra regresa a 0 para continuar con la secuencia de conteo. Observe como el incremento es en el dígito a la izquierda del 7.

Secuencia de números en octal

Cantidad en decimal		Número octal equivalente			
		Pesos de cada posición			
		8^3	8^2	8^1	8^0
		512	64	8	1
0		0	0	0	0
1		0	0	0	1
2		0	0	0	2
3		0	0	0	3
4		0	0	0	4
5		0	0	0	5
6		0	0	0	6
7		0	0	0	7
8		0	0	1	0
9		0	0	1	1
10		0	0	1	2
11		0	0	1	3
12		0	0	1	4
13		0	0	1	5
14		0	0	1	6
15		0	0	1	7

La cantidad de números que se pueden representar en un número octal depende de la cantidad de dígitos que contenga el número, es decir, la cantidad de posiciones en el número. Observe cuidadosamente la tabla con los números en octal del 0 al 15.

7					Con 1 lugar se representan 8 números distintos (8^1 lugares, del 0 al 7)
7	7				Con 2 lugares se representan 64 números distintos (8^2 lugares, del 0 al 63)
7	7	7			Con 3 lugares se representan 512 num. distintos (8^3 lugares, del 0 al 511)
7	7	7	7		Con 4 lugares se representan 4096 num. distintos (8^4 lugares, del 0 al 4095)

Con el sistema de numeración octal se puede representar cualquier cantidad. La razón por la cual es posible representar una cantidad infinita de números en octal es porque se pueden agregar dígitos a la izquierda, para representar cantidades mayores, y el dígito agregado siempre será el dígito más significativo (MSD), pues se anexa a la izquierda. La fórmula general para conocer la cantidad de dígitos que se requieren para la representación de un número en particular es:

$$8^N = 8^N \text{ números distintos } (0..8^N - 1)$$

Donde N representa la cantidad de lugares o posiciones para almacenar el número en octal.

Sistema de numeración hexadecimal

El sistema hexadecimal está formado por 16 dígitos o símbolos. Se comienza con los 10 dígitos decimales, y se continua con las 6 primeras letras del alfabeto (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F). Se conoce como sistema de numeración base 16. Este sistema es muy utilizado en computación por tener una base que es potencia exacta de 2, característica que hace la conversión simple entre los sistemas de numeración binario y hexadecimal.

16^4	16^3	16^2	16^1	16^0		16^{-1}
65536	4096	256	16	1		0.0625
1	F	4	5	3	.	A

MSD

El punto hexadecimal

LSD

El valor de cada posición

Por ejemplo, el número hexadecimal $1F453.A_{16}$ al ser expresado como potencias de 16 se puede conocer su valor en el sistema decimal.

$$1F453.A_{16} = (1 \times 16^4) + (F \times 16^3) + (4 \times 16^2) + (5 \times 16^1) + (3 \times 16^0) + (10 \times 16^{-1})$$

$$1F453.A_{16} = (1 \times 65536) + (15 \times 4096) + (4 \times 256) + (5 \times 16) + (3 \times 1) + (10 \times 0.0625)$$

$$1F453.A_{16} = 65536 + 61440 + 1024 + 80 + 3 + 0.625$$

$$1F453.A_{16} = 128,083.625_{10}$$

El sistema de numeración hexadecimal requiere de menor cantidad de dígitos para representar un número que los sistemas de numeración decimal, octal y binario. Con una sola posición, puede representar hasta 16 números diferentes. Las ventajas de poder representar un número muy grande con una menor cantidad de dígitos y ser un múltiplo de 2, convierten al sistema hexadecimal en un sistema ideal para trabajar con los sistemas digitales.

El conteo hexadecimal

El conteo en hexadecimal con un solo dígito es 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. El orden en estos dígitos no cambia, por lo que no hay ambigüedad en la secuencia. No existe ningún número intermedio entre dos dígitos consecutivos. Cuando se necesita expresar un número mayor que los 16 primeros dígitos del sistema hexadecimal, se agregan cifras a la izquierda del dígito de la misma manera que en los demás sistemas de numeración posicional.

En la secuencia de conteo en el sistema hexadecimal, como en el sistema decimal y binario, también se tiene que, al utilizar N lugares o dígitos en el número codificado en hexadecimal, se pueden realizar 16^N conteos, empezando desde el 0 y hasta llegar a $16^N - 1$.

Cantidad en decimal		Número hexadecimal equivalente			
		Pesos de cada posición			
		16^3	16^2	16^1	16^0
		4096	256	16	1
0		0	0	0	0
1		0	0	0	1
2		0	0	0	2
3		0	0	0	3
4		0	0	0	4
5		0	0	0	5
6		0	0	0	6
7		0	0	0	7
8		0	0	0	8
9		0	0	0	9
10		0	0	0	A
11		0	0	0	B
12		0	0	0	C
13		0	0	0	D
14		0	0	0	E
15		0	0	0	F

La cantidad de números que se pueden representar en un número hexadecimal depende de la cantidad de posiciones que se dispongan. En la tabla con los números equivalentes entre el sistema decimal y hexadecimal se puede observar que con un solo dígito hexadecimal se pueden representar hasta 16 números decimales.

F				Con 1 lugar se representan 16 números distintos (16^1 lugares, del 0 al 15)
F	F			Con 2 lugares se representan 256 números distintos (16^2 lugares, del 0 al 255)
F	F	F		Con 3 lugares se representan 4096 números distintos (16^3 lugares, del 0 al 4095)
F	F	F	F	Con 4 lugares se representan 65536 números distintos (16^4 lugares, del 0 al 65535)

El símbolo F, es el símbolo de mayor valor en el sistema hexadecimal. El siguiente número en la secuencia de conteo en hexadecimal después de F es 10, es decir, se agrega un dígito a la izquierda de F, y se cambia F por el siguiente símbolo después de F, que es 0.

Cada una de las cantidades que se pueden expresar en el sistema hexadecimal, se representa por un número diferente, es posible representar una infinita cantidad de números porque se pueden agregar los dígitos que sean necesarios para su representación siempre a la izquierda de dígito más significativo.

La fórmula general para conocer la cantidad de dígitos hexadecimales que se requieren para representar un número decimal en particular es

16^N números distintos ($0..16^N - 1$)

Donde N representa la cantidad de posiciones en el número hexadecimal

Tabla de equivalencia de números del 0 al 15												
Decimal	Binario				Octal				Hexadecimal			
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	0	2	0	0	0	2
3	0	0	1	1	0	0	0	3	0	0	0	3
4	0	1	0	0	0	0	0	4	0	0	0	4
5	0	1	0	1	0	0	0	5	0	0	0	5
6	0	1	1	0	0	0	0	6	0	0	0	6
7	0	1	1	1	0	0	0	7	0	0	0	7
8	1	0	0	0	0	0	1	0	0	0	0	8
9	1	0	0	1	0	0	1	1	0	0	0	9
10	1	0	1	0	0	0	1	2	0	0	0	A
11	1	0	1	1	0	0	1	3	0	0	0	B
12	1	1	0	0	0	0	1	4	0	0	0	C
13	1	1	0	1	0	0	1	5	0	0	0	D
14	1	1	1	0	0	0	1	6	0	0	0	E
15	1	1	1	1	0	0	1	7	0	0	0	F

Número equivalente
binario - decimal

Figura 1.2 Equivalencia de números decimal - binario - octal - hexadecimal

En la tabla de equivalencia de números, se puede notar que para representar el número 15_{10} en binario se requieren 4 posiciones, mientras que para representar el número 15_{10} en hexadecimal sólo se requiere una sola posición. Cada dígito hexadecimal se representa por un conjunto de 4 dígitos binarios.

1.2 Conversiones entre sistemas numéricos

Los sistemas de numeración decimal, binario, octal y hexadecimal son sistemas de numeración posicionales y se pueden llevar a cabo conversiones entre ellos, tomando en cuenta el peso de cada una de sus posiciones.

Conversión de binario a decimal

Para convertir un número de binario a decimal, se suman los pesos de las posiciones en el número binario que contenga un 1 y el resultado es el equivalente en el sistema decimal.

Ejemplo:

$$1001.101_2 = (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$$

$$1001.101_2 = 8 + 0 + 0 + 1 + 0.50 + 0 + 0.125$$

$$1001.101_2 = 9.125_{10}$$

Conversión decimal a binario

Para convertir un número decimal al sistema binario se consideran los pesos tanto de las posiciones en el número decimal como binario. Existen varios métodos para convertir un número decimal en su equivalente número en el sistema binario.

El primer método es el proceso inverso de convertir un número de binario a decimal. Se expresa el número como una suma de potencias de 2 y posteriormente se escriben 1's y 0's en las posiciones de bits apropiadas.

Ejemplo: Convertir el número 87.20_{10} a binario

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}
128	64	32	16	8	4	2	1		0.50	0.25	0.125	0.0625	0.03125
	1	0	1	0	1	1	1	.	0	0	1	1	0

1.3 Tabla de pesos de las posiciones en el sistema binario

Primer método: suma de potencias

1. Se busca en la tabla con los pesos en binario (Tabla 1.3) de izquierda a derecha el primer número que sea menor o igual al número decimal, en este caso el 64, porque el número 128 es mayor que 87.20_{10} .
2. Si el número en el paso 1 es menor o igual, se coloca un 1 en la posición del 64, y se resta el 64 del número decimal, $87.20 - 64 = 23.20_{10}$.
3. Si el número en el paso 1 no es menor o igual, se coloca un 0 y se continua con la siguiente posición a la derecha en los valores posicionales del sistema binario.
4. Se toma el resultado de la resta para repetir nuevamente los pasos del 1 al 3.
5. El procedimiento funciona tanto para la parte entera como la parte fraccional.

Ejemplo:

Parte entera $\text{¿}128 \leq 87\text{? No}$ $\text{¿}64 \leq 87\text{? Sí, restar}$ $\text{¿}32 \leq 23\text{? No}$ $\text{¿}16 \leq 23\text{? Sí, restar}$ $\text{¿}8 \leq 7\text{? No}$ $\text{¿}4 \leq 7\text{? Sí, restar}$ $\text{¿}2 \leq 3\text{? Sí, restar}$ $\text{¿}1 \leq 1\text{?}$ ----- Parte fraccional $\text{¿}0.50 \leq 0.20\text{? No}$ $\text{¿}0.25 \leq 0.20\text{? No}$ $\text{¿}0.125 \leq 0.20\text{? Sí, restar}$ $\text{¿}0.0625 \leq 0.075\text{? Sí, resta}$ $\text{¿}0.03125 \leq 0.0125\text{? No}$	--- $87 - 64 = 23$ --- $23 - 16 = 7$ --- $7 - 4 = 3$ $3 - 2 = 1$ $1 - 1 = 0$ ----- --- --- $0.20 - 0.125 = 0.075$ $.075 - .0625 = 0.0125$ ---	Poner 0 en la posición del 128 Poner 1 en la posición del 64 y restar Poner 0 en la posición del 32 Poner 1 en la posición del 16 y restar Poner 0 en la posición del 8 Poner 1 en la posición del 4 y restar Poner 1 en la posición del 2 y restar Poner 2 en la posición del 1 y restar ----- Poner 0 en la posición del 0.50 Poner 0 en la posición del 0.25 Poner 1 en la posición del 0.125 y restar Poner 1 en la posición del 0.0625, resta Poner 0 en la posición del 0.03125
--	--	--

Segundo método: Divisiones sucesivas

El segundo método para convertir un número decimal a binario consiste en utilizar la división entre 2. Se divide en forma repetida el número decimal entre 2 y se anota el residuo después de cada división hasta que el cociente sea 0. El resultado binario se obtiene al escribir el primer residuo como el LSB y el último como el MSB. Para la parte fraccional se realizan multiplicaciones sucesivas (Figura 1.4). Este método permite convertir de decimal a cualquier otro sistema de numeración posicional, dividiendo entre la base del sistema correspondiente.

Ejemplo: Convertir el número 25.55_{10} a binario

Procedimiento para convertir la parte entera del número:

1. Tomar la parte entera del número que se quiere convertir y dividir entre 2.
2. Si la división es exacta, es decir, el residuo es 0, entonces se coloca un 0 a la izquierda del número.
3. Si la división no es exacta, es decir el residuo es 1, se coloca un 1 a la izquierda del número que se dividió.
4. Realizar los pasos del 1 al 3 tomando la parte entera de la división realizada.
5. Al finalizar se toman en orden los residuos, siendo el último residuo obtenido el bit más significativo (MSB) y el bit menos significativo (LSB) el primero obtenido.

Procedimiento para convertir la parte fraccional del número:

1. La parte fraccional se multiplica por 2 y se anota el resultado de la multiplicación.
2. Separar la parte entera, que es parte del resultado de la conversión y la parte fraccional del producto de la multiplicación se utiliza nuevamente para multiplicar nuevamente por 2.
3. Se repiten los pasos 1 y 2 hasta que el resultado sea 0 o se obtengan la cantidad de dígitos en la fracción requeridos (en la conversión de un número se puede obtener un número infinito de dígitos fraccionales).

	Residuo	Parte entera			Parte fraccionaria			Entero	
		25_{10}	/2	Se colocan los residuos, en el orden que se obtuvieron desde la última división hasta el primero: $25_{10} = 11001.100011_2$	0.55_{10}	x 2	1.1	1	
LSB →	1	12	/2		MSB →	0.1	x2	0.2	0
	0	6	/2			0.2	x2	0.4	0
	0	3	/2			0.4	x2	0.8	0
	1	1	/2			0.8	x2	1.6	1
MSB →	1	0	*		LSB →	0.6	x2	1.2	1

Figura 1.4. Procedimiento de conversión de un número decimal a su equivalente en binario

Cuando se utiliza la calculadora para dividir entre 2, el residuo es 0 si no hay fracción en la división, pero si hay una parte fraccional (en el caso de dividir un número entre 2, la parte fraccionaria siempre es 0.5), entonces el residuo es 1.

Conversión octal a decimal

Ejemplo: Convertir el número 621.53_8 a decimal

8^4	8^3	8^2	8^1	8^0		8^{-1}	8^{-2}
4096	512	64	8	1		0.125	0.0156
0	0	6	2	1	.	5	3

Para convertir un número octal a decimal, se suman los productos del peso de la posición por su valor. Ejemplo:

$$621.53_8 = (6 \times 8^2) + (2 \times 8^1) + (1 \times 8^0) + (5 \times 8^{-1}) + (3 \times 8^{-2})$$

$$621.53_8 = (6 \times 64) + (2 \times 8) + (1 \times 1) + (5 \times 0.125) + (3 \times 0.0156)$$

$$621.53_8 = 384 + 16 + 1 + 0.625 + 0.0468$$

$$621.53_8 = 401.6718_{10}$$

Conversión decimal a octal

Para convertir un número decimal a octal se realizan divisiones sucesivas para la parte entera del número y multiplicaciones sucesivas por 8 para la parte fraccional. El resultado en octal se obtiene al escribir el primer residuo como el LSB y el último como el MSB. Ejemplo

Convertir el número 401.67_{10} a octal

Procedimiento:

1. Tomar la parte entera del número y dividir entre 8.
2. El residuo de la división se multiplica por 8
3. Se repiten los pasos 1 y 2 hasta que la parte entera del número sea 0.

	Residuo x 8	Parte entera			Parte fraccionaria			Entero	
		401_{10}	/8	$401.67_{10} = 621.527_8$	MSB →	0.67	x 8	5.36	5
LSB →	$0.125 \times 8 = 1$	50	/8			0.36	x 8	2.88	2
	$0.25 \times 8 = 2$	6	/8			0.88	x 8	7.04	7
MSB →	$0.75 \times 8 = 6$	0	*		LSB →	0.04	x 8	0.32	0

Conversión de binario a octal

Como se muestra en la figura 1.2, para representar un grupo de 3 bits, se requiere un solo dígito en octal, por lo que la conversión binario – octal se realiza como sigue:

1. A partir del punto binario, hacer grupos de 3 bits de derecha a izquierda, completar con ceros si es necesario cada grupo, recuerde que los 0 a la izquierda no alteran el valor.
2. Cada grupo de 3 bits equivale a un número octal, sustituir por el correspondiente número en octal.
3. Para la parte fraccional, a partir del punto binario, de izquierda a derecha hacer grupos de 3 bits, completar con ceros si es necesario.
4. Cada grupo de 3 bits corresponde a un número octal, sustituir por el correspondiente número en octal.

Convertir el número 1111011000001.1101_2 a octal

$001-111-011-000-001.110-100_2$

1 7 3 0 1 . 6 4₈

$1111011000001.1101_2 = 17301.64_8$

Conversión octal a binario

Para convertir un número octal en binario, cada dígito octal se convierte en su equivalente binario de 3 bits.

Ejemplo: convertir el número 4563_8 en su equivalente en binario

Sustituir cada dígito octal por su grupo de 3 bits equivalente:

$4563_8 = 100101110011_2$

Conversión de número hexadecimal a decimal

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Para convertir un número hexadecimal a decimal, se suman los productos del peso de la posición por su valor. Como los dígitos en el sistema hexadecimal son 16, al momento de tomar el valor de la posición, si el dígito es mayor que 9, entonces se toma el valor numérico que corresponde a la letra. Ejemplo: Convertir $7B9F.C_{16}$ a decimal

16^4	16^3	16^2	16^1	16^0		16^{-1}
65536	4096	256	16	1		0.0625
0	7	B	1	F	.	C

$$7B9F.C_{16} = (7 \times 16^3) + (11 \times 16^2) + (9 \times 16^1) + (15 \times 16^0) + (12 \times 16^{-1})$$

$$7B9F.C_{16} = (7 \times 4096) + (11 \times 256) + (9 \times 16) + (15 \times 1) + (12 \times 0.0625)$$

$$7B9F.C_{16} = 28672 + 2816 + 144 + 15 + 0.75$$

$$7B9F.C_{16} = 31647.75_{10}$$

Conversión decimal a hexadecimal

Para convertir un número decimal a hexadecimal se realizan divisiones sucesivas entre 16 en la parte entera del número y multiplicaciones sucesivas por 16 en la parte fraccionaria (Método de divisiones sucesivas). Ejemplo: Convertir el número 31647.75_{10} a hexadecimal

Procedimiento:

1. Tomar la parte entera del número y dividir entre 16.
2. El residuo de la división se multiplica por 16
3. Se repiten los pasos 1 y 2 hasta que la parte entera del número sea 0.
4. Para convertir la parte fraccional del número se realizan multiplicaciones sucesivas de la parte fraccional, se toma la parte entera para formar parte del número y la parte fraccional se multiplica nuevamente por 16.

	Residuo x 16	Parte entera			Parte fraccional			Entero	
		31647	/16		MSB → 0.75	x 16	12.0	C	
LSB →	0.9375x16=15	1977	/16			x 16			
	0.5625x16=9	123	/16			x 16			
MSB →	0.6875x16=11	7	/16		LSB →	x 16			
	0.4375x16=7	0	*						

$$31647.75_{10} = 7B9F.C_{16}$$

Conversión de binario a hexadecimal

Como se muestra en la figura 1.2, para representar un dígito en hexadecimal se requiere un grupo de 4 bits. La conversión binario – hexadecimal es simple:

1. A partir del punto binario, hacer grupos de 4 bits de derecha a izquierda, completar con ceros si es necesario, recuerde que los 0 a la izquierda no alteran el valor.
2. Cada grupo de 4 bits equivale a un número hexadecimal, sustituir por el correspondiente número hexadecimal.
3. Para la parte fraccionaria, a partir del punto binario, de izquierda a derecha hacer grupos de 4 bits, completar con ceros si es necesario.
4. Cada grupo de 4 bits corresponde a un número hexadecimal, sustituir por el correspondiente número hexadecimal.

Convertir el número 1111011000001.1101_2 a hexadecimal

0001-1110-1100-0001.1101₁₆

1 E C 1 . D₁₆

$$1111011000001.1101_2 = 1EC1.D_{16}$$

Conversión hexadecimal a binario

Para convertir un número hexadecimal a binario, cada dígito hexadecimal se convierte en su equivalente binario de 4 bits. Ejemplo: convertir el número $4A18_{16}$ en su equivalente en binario. Sustituir cada dígito hexadecimal por su grupo de 4 bits equivalente:

$$4A18_{16} = 0100101000011000_2$$

1.3 Operaciones básicas (suma, resta, multiplicación y división)

Los sistemas de numeración posicionales tienen la característica de mantener los pesos de cada una de sus posiciones con respecto a potencias consecutivas de la base del sistema, esta característica permite que las operaciones aritméticas básicas como son la suma, resta, multiplicación y división, se realizan de la misma forma en cualquier sistema de numeración, para lo cual se toma en cuenta la base del sistema.

Suma en decimal

$$\begin{array}{r} 271 \\ + 383 \\ \hline 654 \end{array}$$

MSD ← → LSD

La suma se inicia en el dígito menos significativo, si la suma es menor que 10, la suma se lleva a cabo utilizando el conteo numérico de la forma previamente vista. Si la suma es mayor o igual 10, entonces al llegar a 10 se comienza el conteo numérico en cero y se realiza un acarreo. Este procedimiento es válido para cualquier sistema de numeración posicional.

Suma en binario

El sistema de numeración binario es el sistema de numeración apropiado para utilizarse en todos los sistemas digitales. Los circuitos que se encargan de realizar la “suma” lo hacen utilizando dos voltajes de corrientes que se traducen en 1’s y 0’s. Debido a que el sistema de numeración binario sólo utiliza dos dígitos, en la suma binaria ocurren 4 casos:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ y se presenta un acarreo hacia la siguiente posición}$$

$$\begin{array}{r} 010_2 \\ + 001_2 \\ \hline 011_2 \end{array}$$

$$\begin{array}{r} 011_2 \\ + 011_2 \\ \hline 110_2 \end{array}$$

Cuando la suma de dos números binarios sean en forma alternada 0’s y 1’s, no hay acarreo. Pero si la suma es $1 + 1$, entonces, el resultado es 10_2 , por lo que se anota 0 en el dígito menos significativo y se realiza el acarreo. Si en la siguiente posición a la izquierda, la suma es nuevamente $1 + 1$, entonces se suma el bit del acarreo y el resultado es 11_2 .

Resta en binario

De la misma forma que se realiza la resta en decimal, se lleva a cabo la resta en binario. Para que el resultado de la resta sea válido, el minuendo debe ser mayor que el sustraendo. Las reglas básicas para realizar la resta son:

$$0 - 0 = 0$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

$$0 - 1 = 1 \text{ y se presenta un acarreo negativo hacia la siguiente posición}$$

$$\begin{array}{r} 011_2 \\ - 001_2 \\ \hline 010_2 \end{array}$$

$$\begin{array}{r} 010_2 \\ - 001_2 \\ \hline 001_2 \end{array}$$

Multiplicación en binario

La multiplicación de números en binario presenta resultados que se pueden reproducir en forma muy sencilla mediante un dispositivo digital, en este caso no se presenta acarreo. Los diferentes resultados de la multiplicación de números binarios son:

$0 \times 0 = 0$		
$0 \times 1 = 0$		
$1 \times 0 = 0$		
$1 \times 1 = 1$	$\begin{array}{r} 011_2 \\ \times 1_2 \\ \hline 011_2 \end{array}$	$\begin{array}{r} 010_2 \\ \times 0_2 \\ \hline 000_2 \end{array}$

División en binario

Para dividir dos números binarios, se realiza de la misma forma que la división larga, aquí se aplican las operaciones de suma, resta y multiplicación. Para poder realizar una división de una forma más sencilla, es conveniente conocer los números en binario para poder mentalmente hacer aproximaciones de la división. Ejemplo: Dividir 1010_2 entre 11_2

$$11_2 \overline{)1010_2}$$

Como resultado de una división en binario, se obtiene 0 o 1, en este caso el resultado es 1. Posteriormente se realiza de multiplicación de $1_2 \times 11_2$, y se anota debajo del dividendo.

$11_2 \overline{)1010_2}$ 1. Para poder dividir un número, el dividendo debe ser mayor que el divisor, en otro caso, el resultado de la división será una fracción.	$\begin{array}{r} 1 \\ 11_2 \overline{)1010_2} \end{array}$ 2. Como en el sistema decimal, analizamos cuántas veces cabe el 10_2 en el 11_2, y vemos que "no cabe" por lo que es necesario tomar una cifra más, 101_2.	$\begin{array}{r} 1 \\ 11_2 \overline{)1010_2} \\ 11 \end{array}$ 3. Se multiplica el resultado parcial por divisor.	$\begin{array}{r} 1 \\ 11_2 \overline{)1010_2} \\ - 11 \\ \hline 10 \end{array}$ 5. Se realiza la resta, empezando con el LSD, $1_2 - 1_2 = 0_2$, y posteriormente se toman 2 cifras por lo que se tiene un acarreo negativo: $10_2 - 1_2 = 1_2$
$\begin{array}{r} 1 \\ 11_2 \overline{)1010_2} \\ 11 \\ \hline 100 \end{array}$ 6. Se baja el 0, siguiente dígito del dividendo	$\begin{array}{r} 11 \\ 11_2 \overline{)1010_2} \\ 11 \\ \hline 100 \end{array}$ 7. Se analiza cuántas veces cabe el 100_2 entre el 11_2, una vez.	$\begin{array}{r} 11 \\ 11_2 \overline{)1010_2} \\ 11 \\ \hline 100 \\ 11 \end{array}$ 8. El multiplica el resultado parcial por el divisor.	$\begin{array}{r} 11 \\ 11_2 \overline{)1010_2} \\ 11 \\ \hline 100 \\ - 11 \\ \hline 001 \end{array}$ 9. Para la comprobación, se multiplica cociente por divisor, más el residuo para obtener el dividendo.

Suma en octal

$$\begin{array}{r} 261 \\ + 353 \\ \hline 634 \end{array}$$

MSD ← → LSD

La suma se inicia en el dígito menos significativo, si la suma es menor que 10_8 , la suma se lleva a cabo utilizando el conteo numérico normal. Si la suma es mayor o igual 10_8 , entonces al llegar a 10 se comienza el conteo numérico en cero y se realiza un acarreo.

Resta en octal

De la misma forma que se realiza la resta en decimal, se lleva a cabo la resta en octal. La diferencia entre los diferentes sistemas de numeración es hasta qué número se lleva a cabo el conteo regresivo inicia en 7 hasta llegar a 0.

Se inicia la resta con el dígito LSD, si el minuendo es mayor que el sustraendo, entonces se restan las cifras como en el sistema decimal. Si el minuendo es menor que el sustraendo, entonces se “pide prestado” al dígito de la izquierda. Este dígito, tiene un valor de 8 unidades (10_8), por lo que se suman 8 unidades al dígito que pidió prestado. Al realizar la suma, ahora el minuendo es mayor que el sustraendo, y la resta se puede llevar a cabo.

$$\begin{array}{r} 526 \\ - 243 \\ \hline 263 \end{array}$$

MSD ← → LSD

Se inicia la resta en $6 - 3 = 3$, como en la resta decimal, posteriormente se continua con los siguientes dígitos menos significativos, $2 - 4$, pero como el minuendo es menor que el sustraendo, se “pide una cifra” al siguiente dígito a la izquierda. La cifra que se pidió prestada se suma al minuendo, en este caso $8_{10} + 2_{10} = 10_{10}$. Ahora se resta $10 - 4 = 6$. Se continua con el siguiente dígito, debido a que se “pidió prestado” se resta $5 - 1 = 4$. Y se resta $4 - 2 = 2$.

Multiplicación en octal

La multiplicación de números en el sistema octal, es similar que la multiplicación en el sistema decimal. Se inicia la multiplicación con el LSD. Si la multiplicación es menor que 8, entonces se realiza la multiplicación como en el sistema decimal y se continua con el siguiente dígito a la izquierda.

$$\begin{array}{r} 031_8 \\ \times 2_8 \\ \hline 062_8 \end{array}$$

Si al multiplicar, el resultado es mayor o igual a 8, entonces se realiza de la siguiente forma.

$$\begin{array}{r} 237 \\ \times 4 \\ \hline 1304 \end{array}$$

$$7 \times 4 = 28_{10} = 34_8$$

Se realiza la multiplicación en decimal con el LSD $7_{10} \times 4_{10} = 28_{10}$, se convierte el 28_{10} a octal y se anota el dígito menos significativo. Continuar la multiplicación con el siguiente dígito a la izquierda y sumar el 3 del acarreo. Continuar de la misma forma hasta terminar con el MSD.

División en octal

La división de dos números en el sistema octal, se realiza de la misma forma que la división decimal. Ejemplo:

$$3_8 \overline{)7245_8}$$

$3_8 \overline{)7245_8}$ 1. Para poder dividir un número, el dividendo debe ser mayor que el divisor, en otro caso, el resultado de la división será una fracción.	$\begin{array}{r} 2 \\ 3_8 \overline{)7245_8} \end{array}$ 2. Similar al sistema decimal, se analiza cuántas veces cabe el 3 en el 7. En este caso, 2 veces.	$\begin{array}{r} 2 \\ 3_8 \overline{)7245_8} \\ 6 \end{array}$ 3. Se multiplica el resultado parcial, ($2 \times 3 = 6$), y se convierte a octal.	$\begin{array}{r} 2 \\ 3_8 \overline{)7245_8} \\ 6 \\ 1 \end{array}$ 4. Se realiza la resta $7 - 6 = 1$. El resultado se anota en octal. En este caso, el resultado en decimal es el mismo que en octal.
$\begin{array}{r} 2 \\ 3_8 \overline{)7245_8} \\ 6 \\ \hline 12 \end{array}$ 5. Se baja el 2, para dividir 12 entre 3.	$\begin{array}{r} 24 \\ 3_8 \overline{)7245_8} \\ 6 \\ \hline 12 \end{array}$ 6. Se analiza cuántas veces cabe el 3 en el 12, en este caso, 4.	$\begin{array}{r} 24 \\ 3_8 \overline{)7245_8} \\ 6 \\ \hline 12 \\ 14 \end{array}$ 7. El multiplica el resultado parcial por el divisor. Se observa que el resultado obtenido es mayor, por lo que al realizar la resta se obtendría un resultado negativo.	$\begin{array}{r} 23 \\ 3_8 \overline{)7245_8} \\ 6 \\ \hline 12 \end{array}$ 8. Se intenta otra vez calcular cuántas veces cabe el 3 en el 12, se propone 3.

$\begin{array}{r} 23 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 01 \end{array}$ 9. Se multiplica $3 \times 3 = 9_{10} = 11_8$	$\begin{array}{r} 23 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 01 \end{array}$ 10. Se realiza la resta en octal	$\begin{array}{r} 23 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 014 \end{array}$ 11. Se baja el 4, y se calcula la división 14 entre 3.	$\begin{array}{r} 234 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 014 \end{array}$ 12. Con base en el cálculo anterior, se decide 4.
$\begin{array}{r} 234 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 014 \\ \underline{14} \\ 0 \end{array}$ 13. Se multiplica $3_8 \times 4_8$ y se anota el resultado debajo de el resultado de la resta realizada en el paso 12.	$\begin{array}{r} 234 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 014 \\ \underline{14} \\ 0 \end{array}$ 14. Se realiza la resta.	$\begin{array}{r} 234 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 014 \\ \underline{14} \\ 05 \end{array}$ 15. Se baja el siguiente dígito del divisor y se calcula cuantas veces cabe el 3 en el 5, en este caso 1.	$\begin{array}{r} 2341 \\ 3_8 \overline{)7245_8} \\ \underline{6} \\ 12 \\ \underline{- 11} \\ 014 \\ \underline{14} \\ 05 \\ \underline{3} \\ 2 \end{array}$ 16. Se multiplica y se resta como en los pasos anteriores.

Suma en hexadecimal

$$\begin{array}{r} 2 \text{ A } 6 \\ + \text{ F } 9 \text{ 6} \\ \hline 1 \text{ 2 } 3 \text{ C} \end{array}$$

MSD ← → LSD

1. De la misma manera que los demás sistemas de numeración posicional, la suma en el sistema hexadecimal inicia con el LSD. Se suman los dos dígitos hexadecimales, realizando mentalmente la conversión de aquellos dígitos mayores que 9.
2. Si la suma es menor o igual a F_{16} (15_{10}), entonces se expresa en forma directa como un dígito hexadecimal.
3. Si la suma es mayor a F_{16} (15_{10}), se restan 16 a la suma y se realiza un acarreo hacia la siguiente posición a la izquierda.
4. Se continua de esta forma hasta completar la suma de todos los dígitos del número hexadecimal.

Resta en hexadecimal

1. De la misma forma que se realiza la resta en decimal, se lleva a cabo la resta en hexadecimal. La diferencia es que cuando el minuendo es menor que el sustraendo, al “pedir prestado” al número que está a la izquierda, esta cantidad “prestada” es 10_{16} (es decir 16_{10}).
2. Se inicia la resta con el LSD, si el minuendo es mayor que el sustraendo, entonces se restan las cifras como en el sistema decimal.
3. Si el minuendo es menor que el sustraendo, entonces “pide prestado” al dígito de la izquierda. Este dígito tiene un valor de 16 unidades (10_{16}), por lo que se suman 16 al dígito que pidió prestado.
4. Después de realizar la suma, el minuendo es mayor que el sustraendo, por lo que la resta se lleva a cabo en forma directa.

$$\begin{array}{r} 52E \\ - 246 \\ \hline 2E8 \end{array}$$

MSD ← → LSD

En el ejemplo, se inicia la resta en $E - 6 = 8$, realizando la sustitución mental de E por su valor hexadecimal que es 14, como en la resta decimal, se continua la resta con los siguientes dígitos a la izquierda, $2 - 4$, pero como el minuendo es menor que el sustraendo, se “pide una cifra” al siguiente dígito a la izquierda. La cifra que se pidió prestada se suma al minuendo, en este caso $16_{10} + 2_{10} = 18_{10}$. Ahora se resta $18 - 4 = E$. Se continua con el siguiente dígito, debido a que se “pidió prestado” se resta $5 - 1 = 4$, y después $4 - 2 = 2$.

Multipliación en hexadecimal

La multiplicación de números en el sistema hexadecimal es similar que la multiplicación en el sistema decimal. Se inicia la multiplicación en el LSD. Si la multiplicación es menor que 16, entonces se realiza la multiplicación como en el sistema decimal y se continua con el siguiente dígito a la izquierda.

$$\begin{array}{r} 125_{16} \\ \times 3_{16} \\ \hline 16F_{16} \end{array}$$

Si al multiplicar, el resultado es mayor o igual a 16, entonces se realiza de la siguiente forma.

$$\begin{array}{r} 237_{16} \\ \times 9_{16} \\ \hline 13EF_{16} \end{array}$$

Se inicia la multiplicación con el LSD $9 \times 7 = 63_{10} = 3F_{16}$, se convierte el 63_{10} a hexadecimal y se anota el dígito menos significativo, en este caso el 15_{10} que es F_{16} . El dígito a la izquierda del número multiplicado, en este caso el 3_{16} se multiplica por 9 y se suma con el acarreo ($3 \times 9 = 27 + 3$), por lo que se obtiene $27 + 3 = 30_{10} = 1E_{16}$. Se anota E y se suma el acarreo de 1 en la siguiente multiplicación. Se multiplica $2 \times 9 = 18_{10} + 1 = 19_{10} = 13_{16}$.

División en hexadecimal

La división, se realiza de la misma forma que la división larga, aquí se aplican las operaciones de suma, resta y multiplicación.

$$5_{16} \overline{) 82A5_{16}}$$

$5_{16} \overline{) 82A5_{16}}$ 1. Para dividir un número, el dividendo debe ser mayor que el divisor, en otro caso, el resultado de la división será una fracción.	$5_{16} \overline{) 82A5_{16}}$ 2. Se analiza cuántas veces cabe el 5 en el 8, 1 vez.	$5_{16} \overline{) 82A5_{16}}$ 3. Se multiplica (1×5) y se anota debajo del dividendo.	$5_{16} \overline{) 82A5_{16}}$ 4. Se realiza la resta $8 - 5 = 3$ y se baja el siguiente dígito.
$5_{16} \overline{) 82A5_{16}}$ 5. Se calcula cuántas veces cabe el 32 en el 5. Recordar que los números son hexadecimales.	$5_{16} \overline{) 82A5_{16}}$ 6. Se analiza cuántas veces cabe el 5 en el 32, A veces. Se multiplica $5 \times A = 50_{10} = 32_{16}$	$5_{16} \overline{) 82A5_{16}}$ 7. Se realiza la resta y se baja el siguiente dígito, en este caso A.	$5_{16} \overline{) 82A5_{16}}$ 8. Se divide A entre 5, el resultado es 2.

1.4 Aplicación de los sistemas numéricos en la computación

Para la representación de todos los caracteres que se utilizan en el teclado de una computadora se utiliza el código ASCII (American Standard Code for Information Interchange). Este es el código alfanumérico más utilizado para la representación de los símbolos utilizados en la computadora.

El código ASCII utiliza 8 bits para almacenar cada uno de los caracteres del teclado, por lo que puede almacenar $2^8 - 1 = 255$ símbolos. Esta cantidad es suficiente para almacenar todos los caracteres estándar del teclado, así como las teclas especiales.

El sistema binario es el lenguaje ideal para las computadoras porque utiliza dos dígitos para su representación, lo que facilita las operaciones en los sistemas digitales, los cuales operan con voltaje o ausencia de voltaje en su funcionamiento. Los sistemas octal y hexadecimal facilitan la interacción humana con el sistema binario, pues en el sistema binario las cifras son grandes secuencias de 0's y 1's, lo que provoca errores y dificultad para la lectura y manipulación en los seres humanos.

Tema 2

Conjuntos y relaciones

La necesidad del ser humano de establecer una forma de comunicación clara y eficiente con los dispositivos digitales ha desencadenado interesantes investigaciones como la Teoría de conjuntos que aporta un gran avance en las ciencias computacionales. Georg Cantor fue el primero en notar la utilidad potencial de investigar las propiedades de los conjuntos, aunque hoy en día es un trabajo ampliamente reconocido, muchos matemáticos de su tiempo se resistieron a aceptar la validez y utilidad de estas investigaciones. “La abstracta teoría de conjuntos es considerada como el fundamento del pensamiento matemático. Todos los objetos matemáticos (¡aún los numéricos!) pueden definirse en términos de conjuntos y el lenguaje de la teoría de conjuntos se utiliza en todos los temas matemáticos” (Sussana Epp, pag. 336).

“La segunda vez que se resuelve un problema, se hace una tarea o actividad, es más fácil que cuando se realiza por primera vez, porque se recuerdan, repiten y mejoran soluciones previas. El ser humano permite organizar las experiencias pasadas para que puedan usarse durante la solución de nuevos problemas. La clasificación se define como el proceso de reconocer una entidad como una instancia de un concepto conocido”. Los elementos de un conjunto se reconocen por sus propiedades.

Competencia específica: Conoce y aplica las operaciones y propiedades de los conjuntos y relaciones para la resolución de los problemas reales.

2.1 Características de los conjuntos y subconjuntos

El concepto de conjunto

“Se entiende por conjunto la agrupación en un todo de **objetos bien diferenciados** de nuestra intuición o de nuestra mente” (Georg Cantor). También se puede definir un conjunto como una **colección bien definida de objetos** llamados elementos o miembros del conjunto.

La frase “bien definida” es esencial para determinar el conjunto. Ya que para que una colección se denomine conjunto, no debe existir ambigüedad o subjetividad entre los miembros del conjunto.

Ejemplos de conjuntos válidos:

“El conjunto de niños menores de 4 años del grupo de Kinder “

“El conjunto de mexicanos entre 15 y 20 años”

“El conjunto de pelotas de básquetbol que tiene el equipo de IT de Ciudad Madero”

“El conjunto de alumnos de ISC de IT de Ciudad Madero que cursan Matemáticas discretas”

“El conjunto de alumnos inscritos en primer semestre en el IT de Ciudad Madero en agosto 2025”

Los conjuntos se indican por medio de una letra mayúscula y los elementos o miembros del conjunto se simbolizan por medio de letras minúsculas. Ejemplo:

El conjunto A tiene como elementos a las vocales del alfabeto

$A = \{a, e, i, o, u\}$

Características de un conjunto.

- Si en un conjunto aparece un elemento del conjunto más de una vez, entonces los elementos repetidos se pueden eliminar porque sólo se consideran una sola vez, es decir, un elemento pertenece o no al conjunto.
- El orden en que aparecen los elementos no es relevante, pueden aparecer en cualquier orden.
- Un elemento x pertenece a un conjunto A , si se puede comprobar que el elemento forma parte del conjunto y se denota $x \in A$
- Un elemento x no pertenece a un conjunto A si se demuestra que el elemento no forma parte del conjunto. Se denota $x \notin A$

$A = \{a, e, i, o, u\} = \{e, o, i, u, a, a, a, i, e\}$

Se dice que dos conjuntos son iguales si y sólo sí, tienen los mismos elementos, sin importar el orden en que aparecen, por ejemplo:

$A = \{1, 6, 7\}$

$B = \{7, 1, 6\}$

$\therefore A = B$

En el caso del conjunto de las vocales del alfabeto, se pueden listar los elementos o miembros del conjunto; pero en muchas ocasiones, es imposible o no es conveniente listar todos los miembros del conjunto. Entonces se utiliza la notación abstracta que consiste en especificar de manera clara y precisa la característica (o características) de los elementos que pertenecen al conjunto. Por ejemplo,

$$A = \{x \mid P(x)\}$$

Y se lee como A es el conjunto de las x, tal que x cumple con la condición (o condiciones) P(x).

Por ejemplo, sea el conjunto B que tiene como elementos a todos los nombres de las personas que tienen su credencial del INE con registro en el estado de México. En este caso, sería posible hacer un listado de todos los miembros del conjunto, sin embargo, es inconveniente, porque la cantidad de elementos que pertenecen al conjunto es demasiado extensa.

En lugar de realizar el listado, el conjunto se puede expresar como sigue:

$$B = \{x \mid x \text{ es el nombre de una persona que tiene credencial de elector del estado de México}\}$$

Conjunto finito. En matemáticas un conjunto finito A es aquel conjunto que tiene como cantidad de elementos un número natural, por lo que todo conjunto finito es un conjunto numerable.

Conjunto infinito. Es un conjunto cuyo número de elementos que pertenecen al conjunto son infinitos, por lo que no se pueden contar.

En matemáticas existen conjuntos importantes que se distinguen por sus características y que se usan continuamente por lo que se denotan con alguna letra especial. Algunos de los conjuntos más utilizados, se muestran a continuación.

N = Conjunto de los números naturales = $\{1, 2, 3, \dots\}$ Z^+ = Conjunto de los números enteros no negativos = $\{0, 1, 2, 3, \dots\}$ Z = Conjunto de los números enteros = $\{\dots, -2, -1, 0, 1, 2, 3, \dots\}$ Q = Conjunto de los números racionales = $\{a/b \mid a, b \in Z; b \neq 0\}$ R = Conjunto de los números reales U = Conjunto Universo $\emptyset$ = Conjunto vacío

Subconjuntos

Si todos los elementos del conjunto A también son elementos del conjunto B, entonces se dice que A es un subconjunto de B o que el conjunto A está contenido en B, esto se denota como:

$$A \subseteq B$$

Por otro lado, si A no es subconjunto de B (o A no está contenido en B) se escribe: $A \not\subseteq B$

Se dice que dos conjuntos son iguales cuando tienen los mismos elementos, es decir se cumple que:

$$A \subseteq B \text{ y } B \subseteq A$$

Por ejemplo

$$A = \{\text{fresa, kiwi, limón}\}$$

$$B = \{\text{limón, fresa, kiwi}\}$$

Entonces

$$A = B$$

Considere los siguientes conjuntos:

$$A = \{x \mid x \in \mathbb{Z}; 50 \leq x \leq 100\}$$

$$B = \{x \mid 2, 13, 21, 51, 85, 90, 100\}$$

$$C = \{51, 85, 90\}$$

Entonces se tiene que:

$C \subseteq A$	$B \not\subseteq A$	$A \not\subseteq B$
$C \subseteq B$	$B \not\subseteq C$	$A \not\subseteq C$

Aplicando la definición de subconjunto

$A \subseteq A$	Un conjunto A es siempre subconjunto de sí mismo
$\emptyset \subseteq A$ $\emptyset \subseteq U$ $\emptyset \subseteq \emptyset$	Se dice que el conjunto vacío es un subconjunto de todos los conjuntos y en particular es un subconjunto de sí mismo.
$A \subseteq U$ $\emptyset \subseteq U$ $U \subseteq U$	Todos los conjuntos, son subconjuntos del conjunto universo

Conjunto potencia

Se le llama conjunto potencia al conjunto que se integra por todos los subconjuntos del conjunto A y se denomina $P(A)$. Ejemplo, se tiene el conjunto A como sigue

$$A = \{1, 2, 3, 4\}$$

El conjunto potencia de A es:

$$P(A) = \{\{\emptyset\}, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,3\}, \{1,2,4\}, \{2,3,4\}, \{3,4,1\}, \{1,2,3,4\}\}$$

El número de elementos del conjunto potencia (es decir, el número de subconjuntos que se pueden formar con los elementos del conjunto A) se determina por la cardinalidad del conjunto A

$$|P(A)| = 2^n$$

Donde n es el número de elementos del conjunto A:

$$|P(A)| = 2^4 = 16$$

Diagramas de Venn

Los diagramas de Venn son representaciones gráficas para mostrar la relación entre los elementos de los conjuntos. Por lo general cada conjunto se representa por medio de un círculo, óvalo o rectángulo. La forma en que se entrelazan las figuras que representan a los conjuntos, muestra la relación que existe entre los elementos de los respectivos conjuntos.

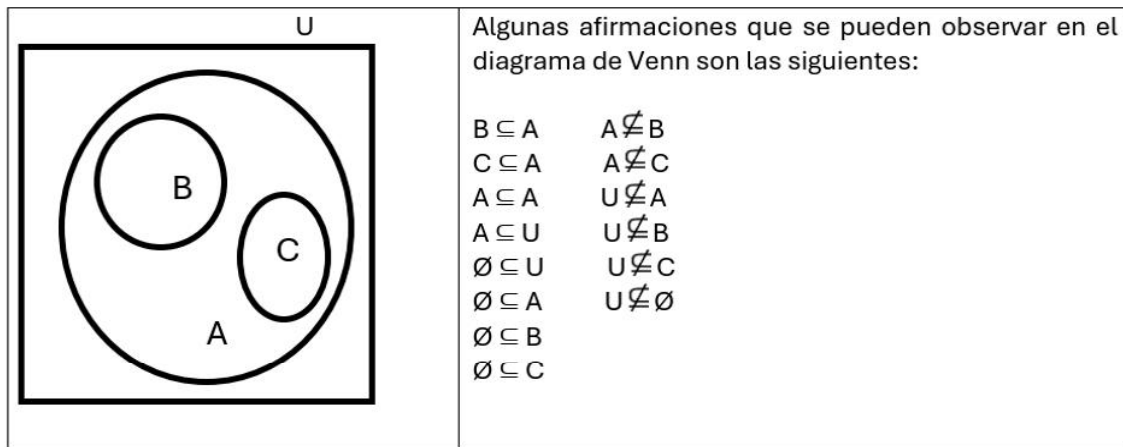


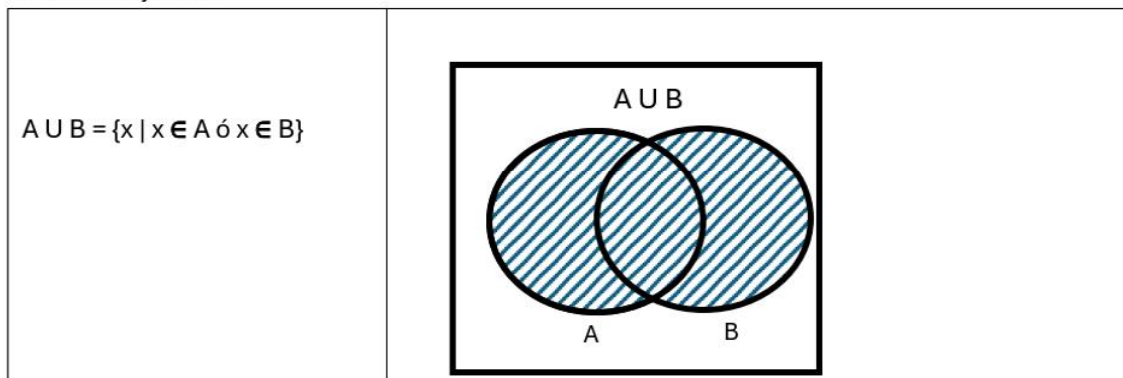
Figura 2.1 Diagrama de Venn

2.2 Operaciones con conjuntos

Los conjuntos permiten realizar operaciones con ellos, estas operaciones son: unión, intersección, diferencia, diferencia simétrica, complemento.

Unión de conjuntos

La unión del conjunto A y el conjunto B es el conjunto que contiene a todos los elementos de ambos conjuntos.



Ejemplo

Sea

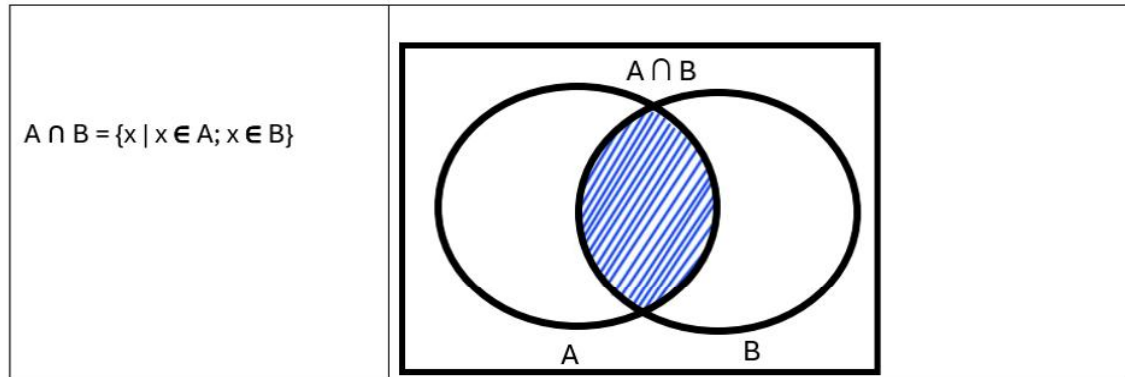
$$A = \{1, 2, 3, 4, 5\}$$

$$B = \{x \mid x \in \mathbb{Z}; x \leq 10; x \geq 2\}$$

$$A \cup B = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

Intersección de conjuntos

La intersección del conjunto A y el conjunto B es el conjunto que contiene a todos los elementos que están en el conjunto A y también están en el conjunto B, es decir, aquellos elementos que son comunes a los conjuntos A y B.



Ejemplo

Sea

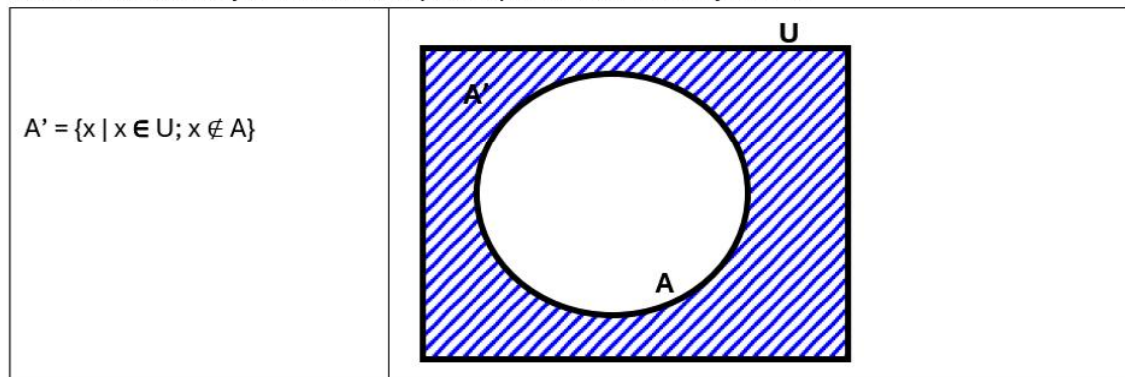
$$A = \{1, 2, 3, 4, 5\}$$

$$B = \{x \mid x \in \mathbb{Z}; x \geq 2; x \leq 10\}$$

$$A \cap B = \{2, 3, 4, 5\}$$

Complemento de un conjunto

El complemento de un conjunto A, que se denota como A' , es el conjunto que tiene a todos los elementos del conjunto universo que no pertenecen al conjunto A.



Ejemplo

Sea

$$U = \{x \mid x \in \mathbb{Z}; x \geq 1; x \leq 10\}$$

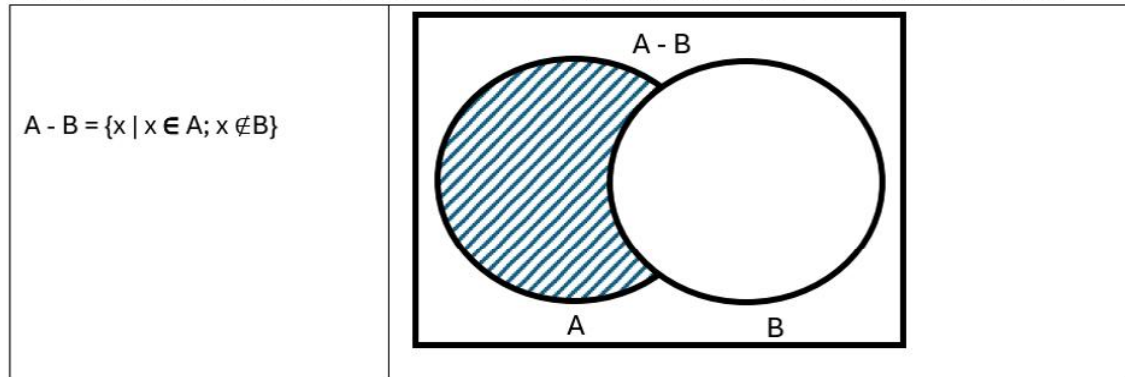
$$A = \{1, 2, 3, 4, 5, 6, 7\}$$

$$A' = \{8, 9, 10\}$$

Diferencia de conjuntos

La diferencia entre dos conjuntos (se denota $A - B$), es el conjunto que contiene a todos los elementos del conjunto A que no se encuentran en el conjunto B . El conjunto diferencia también se conoce como complemento de B con respecto a A .

El orden en que se lleva a cabo la operación de diferencia produce resultados distintos.



Ejemplo

Sea

$$A = \{1, 2, 3, 4, 5\}$$

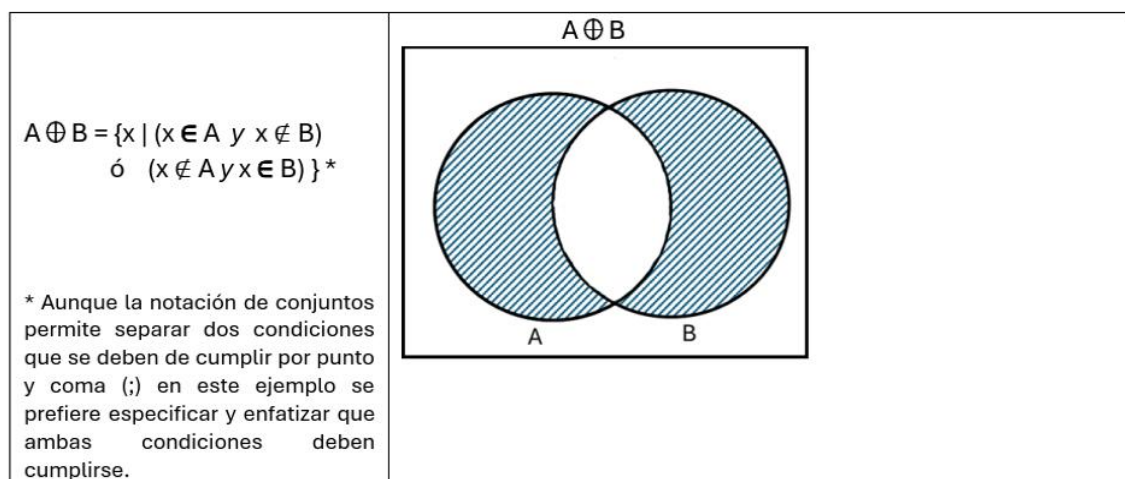
$$B = \{x \mid x \in \mathbb{Z}; x \geq 4; x \leq 10\}$$

$$A - B = \{1, 2, 3\}$$

$$B - A = \{6, 7, 8, 9, 10\}$$

Diferencia simétrica de conjuntos

La diferencia simétrica de los conjuntos A y B , que se denota como $A \oplus B$, es el conjunto que contiene a todos los elementos que se encuentran en A , pero que no se encuentran en B y también los elementos del conjunto B que no están en A . Es decir, el conjunto $A \oplus B$ contiene a todos los elementos que se encuentran en $A \cup B$, pero que no están en $A \cap B$.



Ejemplo

Sea

$$A = \{1, 2, 3, 4, 5\}$$

$$B = \{x \mid x \in \mathbb{Z}; x \geq 4; x \leq 10\}$$

$$A \oplus B = \{1, 2, 3, 6, 7, 8, 9, 10\}$$

2.3 Propiedades y aplicaciones de los conjuntos

El conjunto es una estructura discreta que sirve como fundamento para construir otras estructuras discretas como son el álgebra booleana y la lógica matemática. A través de sus principios y leyes se cuenta con un lenguaje de los conjuntos que permite estudiar colecciones de objetos de forma organizada. Mediante el uso de estructuras discretas como los conjuntos que, a través de diversas estrategias, permiten almacenar y gestionar datos.

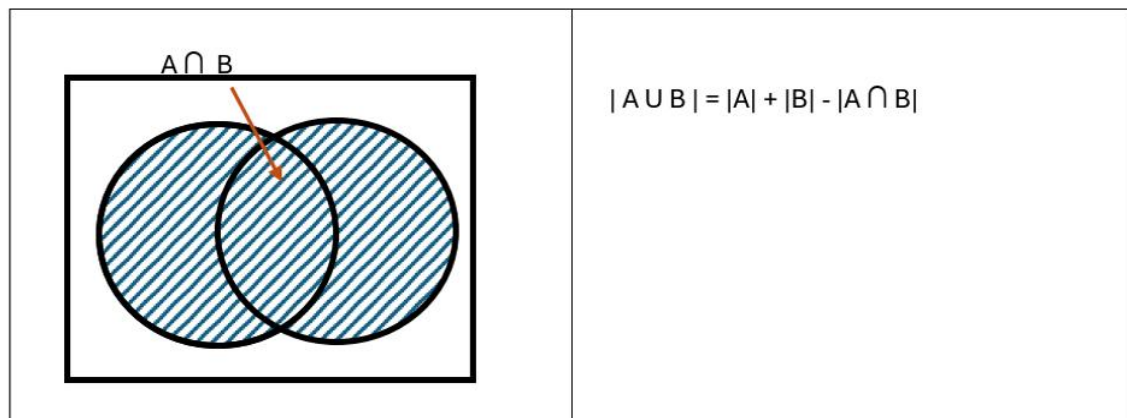
Aplicación de conjuntos finitos

Cuando un conjunto es finito, es decir, cuando se conoce el número de elementos que pertenecen al conjunto, existen situaciones en donde un elemento de un conjunto también pertenece a otro conjunto y se desea conocer el número de elementos en total, considerando una sola vez aquellos elementos que pertenecen a dos o más conjuntos. Por ejemplo:

Sean A y B dos conjuntos finitos, entonces:

$$|A \cup B| = |A| + |B| - |A \cap B|$$

Donde $|A|$ es la cardinalidad de A y $|B|$ es la cardinalidad de B. Al analizar la fórmula anterior, se puede notar que $|A \cup B|$ corresponde a la suma de las áreas de los conjuntos A y B, pero como el área de la intersección de A y B ($A \cap B$) se sumó dos veces, entonces es necesario restarla una vez.



Ejercicio 1

Las escuelas secundarias de la zona norte de Tampico participan en un torneo deportivo. La escuela secundaria “Ignacio Zaragoza” cuenta con 130 estudiantes en total. Del total de estudiantes que tiene la escuela 10 alumnos están participando en Básquetbol, 12 alumnos están participando en Voleibol, pero 5 estudiantes participan en ambas disciplinas.

¿Cuántos alumnos en total participarán en el evento?

¿Cuántos alumnos juegan solamente Básquetbol?

¿Cuántos alumnos juegan exclusivamente Vóleibol?

Para resolver este ejemplo, se puede utilizar la fórmula o Diagramas de Venn. Se especifican los conjuntos involucrados:

U = Universo 130 alumnos

A = El conjunto de alumnos que juegan Básquetbol (10 alumnos)

B = El conjunto de alumnos que juegan Vóleibol (12 alumnos)

$|A \cap B|$ = Alumnos que participan en ambas disciplinas (5 alumnos)

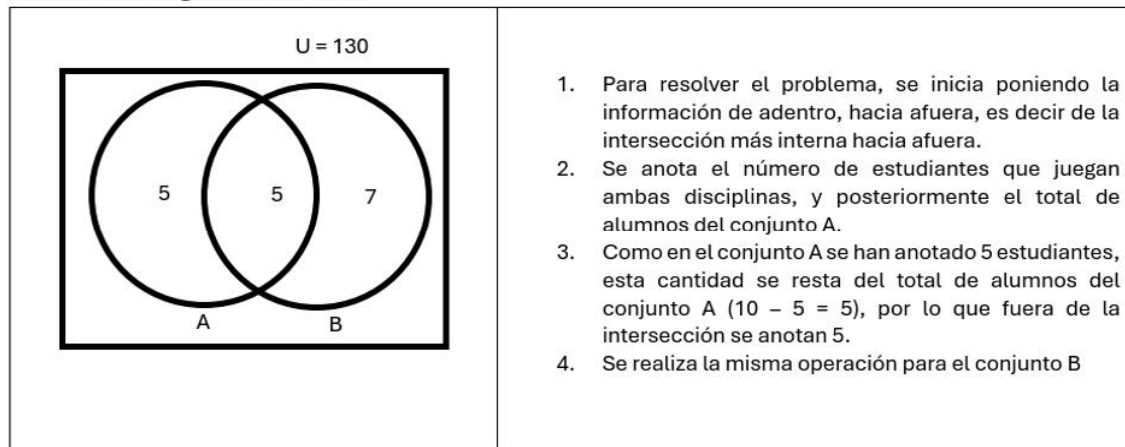
Utilizando la fórmula

$$|A \cup B| = |A| + |B| - |A \cap B|$$

$$= 10 + 12 - 5$$

$$|A \cup B| = 17$$

Utilizando Diagramas de Venn



Ahora se pueden contestar las preguntas:

¿Cuántos alumnos en total participarán en el evento?

17 alumnos

¿Cuántos alumnos juegan solamente Básquetbol?

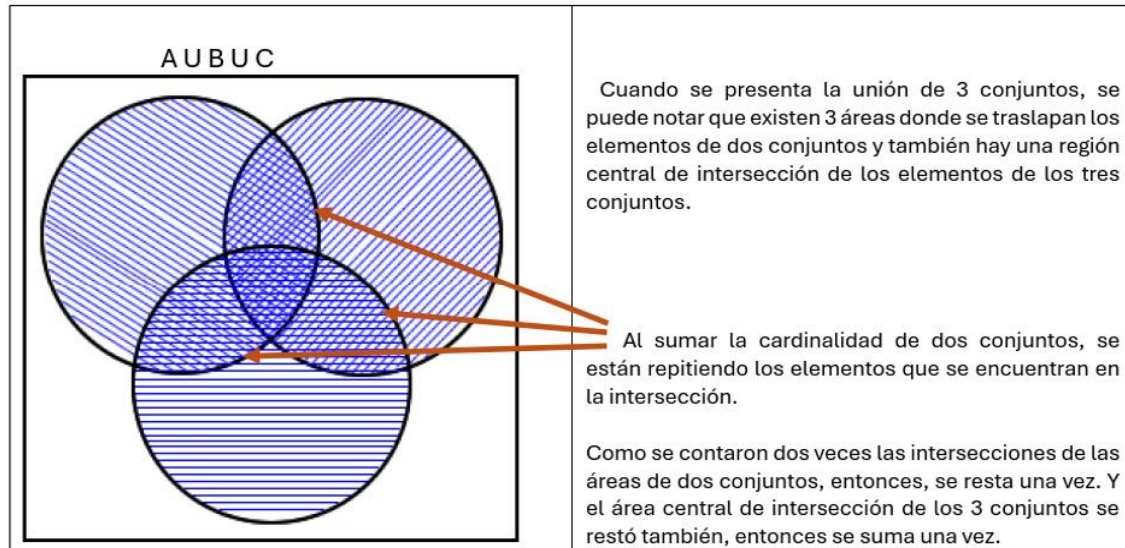
5 alumnos

¿Cuántos alumnos juegan exclusivamente Vóleibol?

7 alumnos

Cuando se involucran 3 conjuntos finitos, la expresión para resolver el problema es:

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$



Ejercicio 2

Las escuelas secundarias de la zona norte de Tampico participan en un torneo deportivo. La escuela secundaria “Ignacio Zaragoza” cuenta con 130 estudiantes en total. Del total de estudiantes que tiene la escuela 10 alumnos están participando en Básquetbol, 12 alumnos están participando en Vóleibol, 24 participan en Fútbol. Algunos alumnos participan en dos disciplinas, 5 alumnos participan en Básquetbol y Voleibol, 3 en Básquetbol y Fútbol, 4 en Voleibol y Fútbol. Hay 2 alumnos que participan en las 3 disciplinas.

¿Cuántos alumnos en total participan en el evento?

¿Cuántos alumnos juegan solamente Básquetbol?

¿Cuántos alumnos juegan exclusivamente Vóleibol?

¿Cuántos alumnos juegan solamente Fútbol?

¿Cuántos alumnos no participan en ninguna disciplina?

Solución del problema:

Especificar los conjuntos del problema

U = Universo 130 alumnos

A = El conjunto de alumnos que juegan Básquetbol (10 alumnos)

B = El conjunto de alumnos que juegan Vóleibol (12 alumnos)

C = El conjunto de alumnos que juegan Fútbol (24 alumnos)

$|A \cap B|$ = Alumnos que participan en Básquetbol y Vóleibol (5 alumnos)

$|A \cap C|$ = Alumnos que participan en Básquetbol y Fútbol (3 alumnos)

$|B \cap C|$ = Alumnos que participan en Vóleibol y Fútbol (4 alumnos)

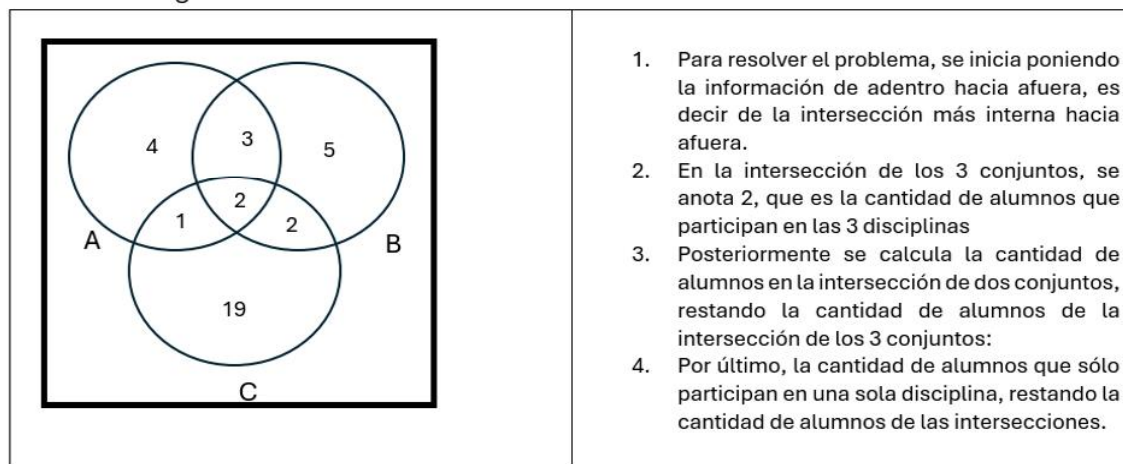
$|A \cap B \cap C|$ = Alumnos que participan en Básquetbol, Vóleibol y Fútbol (2 alumnos)

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |B \cap C| - |A \cap C| + |A \cap B \cap C|$$

$$|A \cup B \cup C| = 10 + 12 + 24 - 5 - 4 - 3 + 2$$

$$|A \cup B \cup C| = 32 \text{ alumnos de la Escuela "Ignacio Zaragoza" participan en el torneo}$$

Utilizando diagramas de Venn:



Con los datos del problema:

$$|A \cap B \cap C| = 2$$

$$|A \cap B| = 5$$

$$|A \cap C| = 3$$

$$|B \cap C| = 4$$

Para calcular la cantidad de alumnos que participan en 2 disciplinas exclusivamente, se resta la cantidad de alumnos que participan en 3 disciplinas.

$$5 - 2 = 3$$

$$3 - 2 = 1$$

$$4 - 2 = 2$$

Para conocer el número de elementos que participan exclusivamente en una sola disciplina, se resta el número de elementos de las intersecciones.

a) El número de alumnos que juegan exclusivamente Básquetbol es $10 - 2 - 3 - 1 = 4$

b) El número de alumnos que juegan exclusivamente Voleibol es $12 - 2 - 3 - 2 = 5$

c) El número de alumnos que juegan exclusivamente Fútbol es $24 - 2 - 1 - 2 = 19$

Como se puede observar, cada círculo en el diagrama de Venn, contiene la cantidad de elementos de cada conjunto, considerando los elementos que pertenecen a la intersección.

Otras aplicaciones de la teoría de conjuntos

Las bases de datos son muy importantes en computación, pues permite almacenar y gestionar grandes volúmenes de información. El funcionamiento de las bases de datos se basa en la teoría de conjuntos por lo que es posible llevar a cabo operaciones como unión o intersección de información entre otras operaciones.

2.4 Conceptos básicos: producto cartesiano y relación binaria

Una relación es una correspondencia entre dos elementos que tienen ciertas características para mantener dicha relación. Una relación entre conjuntos se genera cuando un elemento de un conjunto “se relaciona” con un elemento de otro conjunto. Si los elementos de un conjunto se pueden relacionar con los elementos de otro conjunto, entonces se dice que los conjuntos integran una relación y a esta relación se le llama “relación de orden”.

Por ejemplo, se tienen dos conjuntos:

Mujer = {Ana, Bety, Mirna}

Hombre = {Juan, Luis, Paco}

Y se tiene una relación de matrimonio que se genera cuando un hombre se casa con una mujer
Matrimonio = {(Ana, Juan), (Mirna, Luis)}

El conjunto Matrimonio especifica que Ana está casada con Juan y que Mirna está casada con Luis. Existen también elementos del conjunto Mujer y elementos del conjunto Hombre que no están casados, es decir, no pertenecen al conjunto Matrimonio.

El conjunto Matrimonio es una relación entre elementos de 2 conjuntos. La manera de representar los elementos del conjunto que representa una relación es a través de pares ordenados, entre paréntesis se pone en primer lugar un elemento del conjunto Mujer y separado por una coma “,” en segundo lugar un elemento del conjunto Hombre: (Ana, Juan).

Dados dos conjuntos no vacíos A y B, una relación es un conjunto de pares ordenados en donde un elemento del conjunto A está relacionado con uno y sólo un elemento del conjunto B por medio de una propiedad o característica. La relación se indica como aRb :

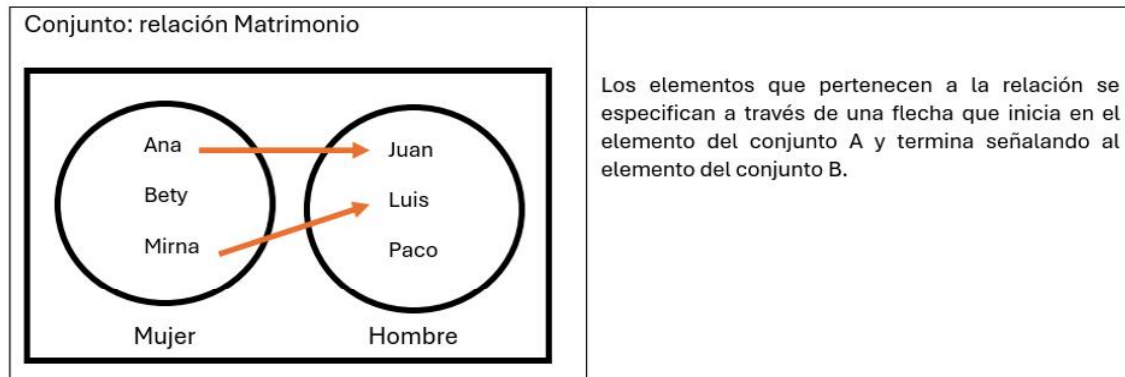
$$R = \{(a,b) | a \in A \text{ y } b \in B\}$$

El par ordenado (a,b) pertenece a la relación R, si **a** es miembro del conjunto A, **b** es miembro del conjunto B y además cumplen con la característica para pertenecer a la relación.

Una segunda alternativa para representar una relación es a través de una tabla que muestra la correspondencia de los elementos del conjunto A y B. Por ejemplo, la relación Matrimonio que se presenta entre una mujer y un hombre.

Relación Matrimonio	
Mujer	Hombre
Ana	Juan
Mirna	Luis

Representación gráfica de un conjunto relación



En términos de relación, se dice que Ana está relacionada con Juan porque están casados. En la representación gráfica la relación se muestra a través de una flecha. El elemento Ana del conjunto Mujer y el elemento Juan del conjunto Hombre pueden tener otros atributos que son importantes como nombre completo, edad, fecha y lugar de nacimiento, etc., pero en el diagrama sólo se muestra el nombre de la persona por simplicidad. Las relaciones se forman si se cumple cierta proposición o condición, la proposición puede ser textual, por ejemplo: “está casada con” o “imparte la materia”, pero también puede ser planteada en lenguaje matemático como “mayor que” $>$ o “menor que” $<$.

Producto cartesiano

El producto cartesiano de los conjuntos A y B, que se denota como $A \times B$, es la combinación de todos los elementos del conjunto A con todos los elementos del conjunto B. En teoría de conjuntos equivale al conjunto universo. Una relación R de A en B ($R: A \rightarrow B$) es un subconjunto del producto cartesiano $A \times B$. Si $R \subseteq A \times B$ y $(a, b) \in R$, entonces a su vez el producto cartesiano también es una relación.

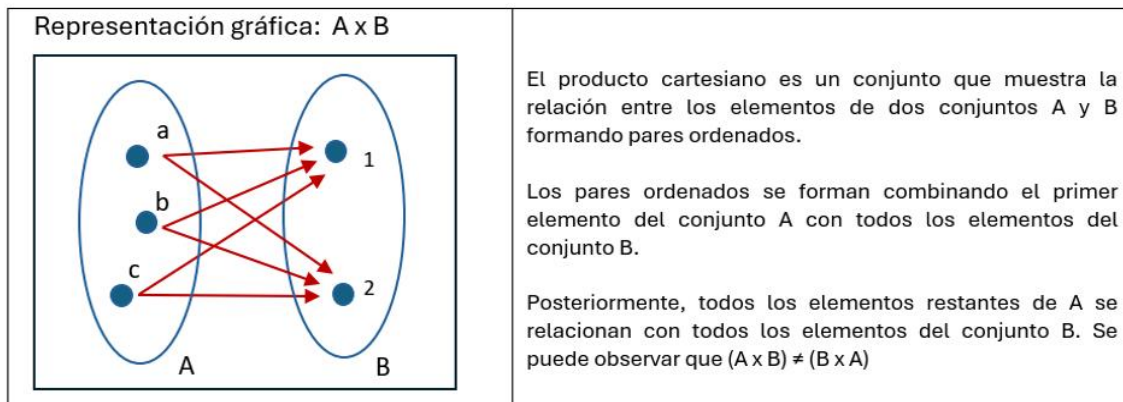
Sean los conjuntos

$$A = \{a, b, c\}$$

$$B = \{1, 2\}$$

“El producto cartesiano $A \times B$ contiene todos los pares ordenados que resultan de relacionar todos los elementos del conjunto A con todos los elementos del conjunto B” (Jiménez, pp. 223).

$$A \times B = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$$



Relación terciaria

No siempre los elementos de la relación son pares ordenados, ya que pueden tener más de dos elementos como el siguiente caso:

$$R = \{(a, 1, \diamond), (b, 2, \circ), (b, 1, \diamond), (c, 3, \circ), (c, 2, \diamond)\}$$

En este caso la relación está formada por tercias de elementos pertenecientes a los conjuntos:

$$A = \{a, b, c\} \quad B = \{1, 2, 3\} \quad C = \{\diamond, \circ\}.$$

Relación binaria

Una de las relaciones más importantes en la computación es la relación binaria, ya que se puede representar por medio de una matriz, una tabla o una gráfica. Se le llama relación binaria porque sus elementos son pares ordenados que se forman a partir de dos conjuntos.

En toda relación de pares ordenados se tienen dos conjuntos: El dominio de R ($\text{Dom}(R)$), que es el conjunto del primer elemento de los pares ordenados de una relación el cual es un subconjunto del conjunto A ($\text{Dom}(R) \subseteq A$) y el codominio de R ($\text{Cod}(R)$), conjunto que está formado por los segundos elementos de los pares de la relación R y que también es un subconjunto de B ($\text{Cod}(R) \subseteq B$)

Ejemplo:

Sean los conjuntos

$$A = \{4, 5, 6, 7, 8, 9\}$$

$$B = \{b | b \in \mathbb{Z}; 1 \leq b \leq 6; b \text{ es par}\}$$

Considere que aRb si y sólo si a es divisible entre 3. Por lo tanto, los elementos de la relación son:

$$R = \{(6,2), (6,4), (6,6), (9,2), (9,4), (9,6)\}$$

$$\text{Dom}(R) = \{6, 9\}$$

$$\text{Cod}(R) = \{2, 4, 6\}$$

2.5 Representación de las relaciones

Con fines de tratamiento computacional, las relaciones deben representarse de tal manera que puedan ser manipuladas por una computadora. Por este motivo, las relaciones se representan como matrices o como grafos.

Representación por medio de una matriz

Si A y B son dos conjuntos finitos con m y n elementos respectivamente, y R es una relación de A en B, entonces, se puede representar R como una matriz $M_R = [m_{ij}]$ cuyos elementos se definen como:

$$M_{ij} = \begin{cases} 1 & \text{si } (a,b) \in R \\ 0 & \text{si } (a,b) \notin R \end{cases}$$

Ejemplo:

Sean los conjuntos

$A = \{1, 2, 3, 4\}$

$B = \{a, b, c, d, e\}$

Y sea la relación R de $A \rightarrow B$

$R = \{(1,a), (3,b), (3,e), (4,d), (4,e)\}$

Representación por medio de una matriz					
	a	b	c	d	e
1	1	0	0	0	0
2	0	0	0	0	0
3	0	1	0	0	0
4	0	0	0	1	1

Se coloca 1 si el par ordenado pertenece a la relación y 0 en caso contrario.

Los elementos del conjunto A se representan como filas y los elementos del conjunto B como columnas.

Representación por medio de grafos

Consiste en representar los elementos de los conjuntos A y B como nodos o círculos y la relación que existe entre los elementos de los dos conjuntos se indica por medio de una flecha que va del elemento del conjunto A hacia el elemento del conjunto B con el que está relacionado. Se utiliza el mismo ejemplo que en matrices para comparar ambas representaciones.

Ejemplo:

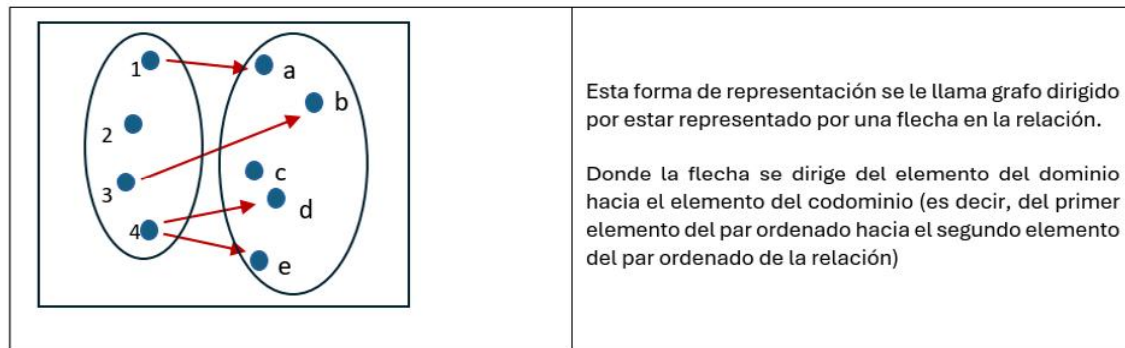
Sean los conjuntos

$$A = \{1, 2, 3, 4\}$$

$$B = \{a, b, c, d, e\}$$

Y sea la relación R de $A \rightarrow B$

$$R = \{(1,a), (3,b), (3,e), (4,d), (4,e)\}$$



Es común que los conjuntos A y B tengan los mismos elementos (ejemplo $A = B = \{1, 2, 3, 4\}$), en este caso se dice que $R \subseteq A \times B$ y es una relación de A , en vez de decir que es una relación de A en A .

Ejemplo:

Sean los conjuntos

$$A = B = \{1, 2, 3, 4\}$$

Y la relación

$$R = \{(1,3), (2,2), (2,4), (3,1), (3,4)\}$$

Represente las relaciones utilizando matriz y grafos

Representación por medio de matriz	Representación por medio de grafos																									
<table><tr><td></td><td>1</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>2</td><td>0</td><td>1</td><td>0</td><td>1</td></tr><tr><td>3</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>4</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>		1	2	3	4	1	0	0	1	0	2	0	1	0	1	3	1	0	0	1	4	0	0	0	0	<pre>graph TD; 1((1)) --> 3((3)); 2((2)) --> 2; 2 --> 4((4)); 3 --> 1; 3 --> 4;</pre>
	1	2	3	4																						
1	0	0	1	0																						
2	0	1	0	1																						
3	1	0	0	1																						
4	0	0	0	0																						

Se observa que cuando los elementos de ambos conjuntos son los mismos, en la representación a través de un grafo sólo se dibujan una sola vez los nodos del problema, y el sentido de la flecha indica el primer y segundo elemento del par ordenado, simplificando la gráfica.

2.6 Propiedades de las relaciones

Existen propiedades que son importantes y que se utilizan para clasificar las relaciones en un conjunto. En este caso, se considera que los elementos del conjunto A y del conjunto B tienen los mismos elementos, por lo que se representan con matrices cuadradas o con grafos en el que los elementos de los conjuntos A y B son los mismos (por lo que en el grafo sólo se muestran una vez los elementos de los conjuntos). Las propiedades más importantes de las relaciones entre conjuntos son:

- a) Relación reflexiva
- b) Relación irreflexiva
- c) Relación simétrica
- d) Relación asimétrica
- e) Relación antisimétrica
- f) Relación transitiva

Relación reflexiva

Una relación es reflexiva cuando todos los elementos de un conjunto A están relacionados consigo mismo, esto es, se cumple que aRa para todo elemento de A. Una característica de este tipo de relación es que la matriz que corresponde a su representación tiene 1's en la diagonal principal y los elementos restantes pueden ser 0's o 1's. Ejemplo

Sean $A = B = \{1, 2, 3, 4\}$

$R = \{(1,1), (2,2), (2,4), (3,3), (4,4)\}$

Matriz reflexiva					
	1	2	3	4	
1	1	0	0	0	
2	0	1	0	1	
3	0	0	1	0	
4	0	0	0	1	

Una matriz reflexiva contiene 1's en la diagonal principal, sin importar el contenido en las demás posiciones. Esto indica que un elemento está relacionado consigo mismo, es decir, $1R1$, $2R2$, etc.

Relación irreflexiva

Se dice que una relación es irreflexiva cuando ningún elemento del conjunto A está relacionado consigo mismo, es decir, $(a,a) \notin R$. En este caso la matriz de la relación deberá contener únicamente 0's en la diagonal. Si la diagonal principal de la matriz tiene 0's y 1's, la relación no es reflexiva ni irreflexiva.

Sean $A = B = \{1, 2, 3, 4\}$

$R = \{(1,2), (1,3), (2,4), (3,2), (4,3)\}$

Matriz irreflexiva					
	1	2	3	4	
1	0	1	1	0	
2	0	0	0	1	
3	0	1	0	0	
4	0	0	1	0	

Una matriz irreflexiva tiene solamente 0's en la diagonal principal, indicando que ningún elemento está relacionado consigo mismo, $(a,a) \notin R$. En las demás posiciones no importa si hay 0's o 1's.

Relación simétrica

Esta matriz tiene unos y ceros en los pares colocados simétricamente, es decir, si $(a,b) \in R$ entonces $(b,a) \in R$. Pero si $(a,b) \notin R$ entonces $(b,a) \notin R$. La condición de simetría se debe cumplir para todos los pares colocados simétricamente.

Para saber si una relación es simétrica se compara la matriz de la relación con su transpuesta: si son iguales entonces se concluye que la relación R es simétrica.

Cada matriz de relación (M_R), tiene una matriz transpuesta (M_R^T). La matriz transpuesta (M_R^T) es una matriz que se obtiene al cambiar las columnas de M_R en filas. En el ejemplo que se presenta a continuación $M_R = M_R^T$ entonces se concluye que la relación R no es simétrica. La condición de simetría se debe cumplir para todos los pares colocados simétricamente.

Sean $A = B = \{1, 2, 3, 4\}$

$R = \{(1,2), (1,3), (2,4), (3,2), (4,3)\}$

Matriz simétrica

Matriz de relación M_R					Transpuesta de la matriz de relación M_R^T					Al cambiar las filas por columnas se obtiene la misma matriz por lo que es una matriz simétrica. $M_R = M_R^T$
	1	2	3	4		1	2	3	4	
1	1	1	0	1	1	1	1	0	1	
2	1	0	1	1	2	1	0	1	1	
3	0	1	0	1	3	0	1	0	1	
4	1	1	1	0	4	1	1	1	0	

Para saber si una relación es simétrica se compara la matriz de relación con su transpuesta: si son iguales se concluye que la relación es simétrica.

Relación asimétrica

Una relación R de A en B es asimétrica si cuando $(a,b) \in R$ entonces $(b,a) \notin R$, además de que ningún elemento deberá estar relacionado consigo mismo; esto significa que la diagonal principal de la matriz de relación deberá contener solamente ceros. Ejemplo:

En los pares simétricos de la siguiente matriz se observa que, si uno de ellos vale 1, su simétrico debe valer 0. Por otro lado, la diagonal principal debe tener solamente 0's, lo cual indica que ningún elemento está relacionado consigo mismo.

Matriz asimétrica

	1	2	3	4	Una matriz irreflexiva tiene solamente 0's en la diagonal principal, indicando que ningún elemento está relacionado consigo mismo, $(a,a) \notin R$. En las demás posiciones no importa si hay 0's o 1's.
1	0	0	1	0	
2	1	0	0	0	
3	0	1	0	0	
4	0	0	1	0	

Relación antisimétrica

Una relación es antisimétrica cuando uno de los pares colocados simétricamente no está en la relación, lo cual significa que o bien $(a,b) \notin R$ o $(b,a) \notin R$, es decir, por lo menos uno de los pares colocados simétricamente no debe estar en la relación. En este caso la diagonal

principal en la matriz no es importante, ya que pueden estar o no relacionados los elementos con ellos mismos.

Matriz antisimétrica

	1	2	3	4
1	0	0	1	0
2	0	1	1	1
3	0	0	0	0
4	0	0	1	0

En esta matriz, cuando menos uno de los pares simétricos de la relación es 0, lo cual significa que $(a,b) \notin R$ o bien $(b,a) \notin R$. En la diagonal principal puede haber 0's o 1's, y también puede haber pares de 0's colocados simétricamente, pero no pares de 1's. Por lo tanto, es una relación antisimétrica.

Relación transitiva

Una relación de A en B tiene la propiedad de ser transitiva si cuando se tienen los pares aRb y bRc en la relación, entonces también existe un par aRc . Por ejemplo:

Si se tienen los pares (2,3) y (3,4), entonces debe existir también el par ordenado (2,4).

Si se tienen los pares (3,1) y (1,3), entonces también debe existir el par ordenado (3,3).

Matriz transitiva

	1	2	3	4
1	0	0	1	0
2	0	1	1	1
3	1	0	1	1
4	0	0	1	0

Para conocer si una matriz es transitiva, se deben revisar todos los pares para ver si cumple con la condición de transitividad. Se debe realizar en forma ordenada, empezando en el primer renglón, primera columna, seguir con las columnas siguientes.

Al terminar el renglón, continuar con el siguiente renglón, y así sucesivamente hasta terminar de revisar todos los pares ordenados que tengan 1.

Revisar todos los pares ordenados de la relación (aquellos que existe un 1 en la matriz).

$(1,3) \rightarrow (3,1), (3,3), (3,4) = (1,1)^*, (1,3), (1,4)^*$ $(2,2) \rightarrow (2,2), (2,3), (2,4) = (2,2), (2,3), (2,4) \odot\odot$ $(2,3) \rightarrow (3,1), (3,3), (3,4) = (2,1)^*, (2,3), (2,4)$ $(2,4) \rightarrow (4,3) = (2,3)$ $(3,1) \rightarrow (1,3) = (3,3)$ $(3,3) \rightarrow (3,1), (3,3), (3,4) = (3,1), (3,3), (3,4)$ $(3,4) \rightarrow (4,3) = (3,3)$ $(4,3) \rightarrow (3,1), (3,3), (3,4) = (4,1)^*, (4,3), (4,4)^*$	Al revisar el primer elemento de la relación (1,3) se compara con la columna 3, porque son los elementos que en su primer elemento del par ordenado contienen un 3. Los elementos 1R3 y 3R1 existen, por lo que debería existir 1R1. En este caso, se observa que el par ordenado (1,1) no existe. Con una sola condición que no se cumpla, es suficiente para determinar que no es una matriz transitiva. En el ejercicio se revisan todos los elementos. Los pares ordenados que tienen asterisco son aquellos que no existen en la matriz de relación, por lo que no es una matriz transitiva.
$\odot\odot$ Cuando el renglón y la columna es la misma en el par ordenado, los elementos con los cuales se debe revisar y los que se deben obtener son los mismos.	

Resumen de las propiedades de las relaciones

Propiedad	Condición
Reflexiva	$aRa, \forall a \in A$
Irreflexiva	$(a,a) \notin R, \forall a \in A$
Simétrica	Si $(a,b) \in R$ entonces $(b,a) \in R$, o bien si $(a,b) \notin R$ entonces $(b,a) \notin R$
Asimétrica	Si $(a,b) \in R$ entonces $(b,a) \notin R$, o bien si $a=b$ entonces $(a,a) \notin R$
Antisimétrica	Si $(a,b) \notin R$ o bien $(b,a) \notin R$. No importa la diagonal principal
Transitiva	Si $(a,b) \in R$ y $(b,c) \in R$ entonces $(a,c) \in R$

2.7 Relaciones de equivalencia

Una relación de equivalencia es aquella que tiene 3 propiedades:

- Es reflexiva
- Es simétrica
- Es transitiva

Para saber si una matriz de relación es una relación de equivalencia se deben revisar cada una de las 3 propiedades para saber si la matriz es reflexiva, simétrica y transitiva. Ejemplo:

Sea la matriz:

	1	2	3	4
1	1	0	1	0
2	0	1	1	0
3	1	1	1	1
4	0	0	1	1

Revisión de las propiedades de equivalencia:

1. Se observa que todos los elementos de la matriz principal tienen 1's, por lo que cada elemento está relacionado consigo mismo. Por lo tanto, la matriz es reflexiva.
2. Para saber si la matriz es simétrica, se intercambian renglones por columnas. Al cambiar renglones por columnas tiene como resultado la misma matriz por lo que se comprueba que es simétrica.
3. Es una matriz transitiva pues para cada elemento que pertenece a la relación, es decir que hay 1, Si $(a,b) \in R$ y $(b,c) \in R$ entonces $(a,c) \in R$.

Cerraduras

No todas las relaciones son de equivalencia, pero es posible hacer que tengan esta propiedad agregando los pares ordenados mínimos necesarios para que sean reflexivas, simétricas y transitivas usando para ello las cerraduras.

Debido a que las relaciones de equivalencia tienen tres propiedades, entonces se requieren 3 cerraduras, una para cada una de las propiedades: reflexiva, simétrica y transitiva. En el siguiente ejemplo, se utilizan las cerraduras para construir una relación de equivalencia.

Cerradura reflexiva

La cerradura reflexiva agrega a la relación (R la matriz original) la matriz identidad para obtener una relación reflexiva (R U I). La relación identidad (I) es una matriz cuadrada cuyos elementos de la diagonal son únicamente 1's y los elementos restantes son 0's.

$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Cerradura reflexiva La suma de la matriz de relación con la matriz identidad (R U I) asegura agregar los elementos mínimos necesarios para que la matriz resultante sea reflexiva. Al unir la matriz de relación y la matriz identidad, sólo se agregan 1's en la diagonal principal, los demás elementos permanecen sin cambio.
--	--

Cerradura simétrica

Para obtener la cerradura simétrica se toma como matriz de relación, la matriz que se obtuvo al aplicar la cerradura reflexiva. A esta relación R, se agrega la relación inversa R^{-1} para que la relación resultante tenga la propiedad de simetría (R U R^{-1}). Al unir la matriz inversa (o relación inversa), se agregan los pares ordenados necesarios, si $(a,b) \in R$ entonces se agrega el par ordenado (b,a) .

$ \begin{array}{c} R \\ \begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{vmatrix} \end{array} $	$ \begin{array}{c} R^{-1} \\ \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{vmatrix} \end{array} $	Cerradura simétrica Se suma la matriz de relación con la matriz inversa, la cual resulta de intercambiar las filas por las columnas. De esta forma se obtiene una matriz de relación que cumple con la propiedad de simetría ($R \cup R^{-1}$)
---	--	--

Cerradura transitiva

La matriz que se obtuvo en la cerradura simétrica se une con la matriz que resulta de multiplicar la relación por ella misma M_R^2 . Después de unir la matriz de relación obtenida de la cerradura simétrica se une con la multiplicación de la matriz de relación por ella misma ($M_R \cup M_R^2$), se revisa nuevamente que cumpla con la propiedad de transitividad, esto es, si $(a,b) \in R$ y $(b,c) \in R$ entonces debe de cumplirse que $(a,c) \in R$. Si alguno de los pares ordenados no cumple con la propiedad, entonces se debe de multiplicar nuevamente la matriz de relación por sí misma, y este proceso se repite hasta que se obtenga la matriz transitiva.

Multiplicar la matriz de relación por sí misma $ \begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{vmatrix} $	Cerradura transitiva 1. Multiplicar la matriz obtenida de la cerradura simétrica por sí misma para obtener M_R^2 2. Unir la matriz obtenida de la cerradura simétrica con M_R^2 3. Revisar que la matriz resultante cumpla con la propiedad de transitividad: si $(a,b) \in R$ y $(b,c) \in R$ entonces $(a,c) \in R$. 4. Si no cumple con la propiedad de transitividad, entonces repetir nuevamente los pasos.
Unir ($M_R \cup M_R^2$) $ \begin{vmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix} $	

Revisión de la propiedad de transitividad en la matriz

$(1,1) \rightarrow (1,1), (1,3), (1,4)$ $(1,3) \rightarrow (3,1), (3,3), (3,4) = (1,1), (1,3), (1,4)$ $(1,4) \rightarrow (4,2), (4,3), (4,4) = (1,2)^*, (1,3), (1,4)$ $(2,2) \rightarrow (2,2), (2,3), (2,4)$ $(2,3) \rightarrow (3,1), (3,3), (3,4) = (2,1)^*, (2,3), (2,4)$ $(2,4) \rightarrow (4,2), (4,3), (4,4) = (2,2), (2,3), (2,4)$ $(3,1) \rightarrow (1,1), (1,3), (1,4) = (3,1), (3,3), (3,4)$ $(3,3) \rightarrow (3,1), (3,3), (3,4)$ $(3,4) \rightarrow (4,2), (4,3), (4,4) = (3,2)^*, (3,3), (3,4)$ $(4,2) \rightarrow (2,2), (2,3), (2,4) = (4,2), (4,3), (4,4)$ $(4,3) \rightarrow (3,1), (3,3), (3,4) = (4,1)^*, (4,3), (4,4)$ $(4,4) \rightarrow (4,1), (4,3), (4,4) = (4,1)^*, (4,3), (4,4)$	Debido a que aún existen pares ordenados que no cumplen con la propiedad de transitividad, se deben volver a realizar los pasos del 1 al 4 para aplicar la cerradura transitiva. Considerar ahora la nueva matriz de la relación como $M_R = M_R \cup M_R^2$ y encontrar la nueva matriz M_R^2 y con esta nueva M_R^2 , se debe aplicar nuevamente la cerradura transitiva.
--	--

Obtención de la nueva cerradura transitiva

$ \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} $	La matriz de relación que se obtiene de la cerradura transitiva se vuelve a multiplicar por sí misma para obtener M_R^2 . Posteriormente, se vuelve a obtener la nueva matriz de relación $M_R = M_R \cup M_R^2$.
$ \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix} + \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix} $	

2.8 Funciones

Una función es un caso específico de una relación. Una función f es una relación que asigna a cada elemento x de un conjunto X a un único elemento y de un conjunto Y . Se define una función como un tipo especial de pares ordenados.

Sean A y B conjuntos no vacíos. Una función f de A en B se escribe como

$f: A \rightarrow B$

Todas las funciones son relaciones, pero no todas las relaciones son funciones. Para que una relación pueda considerarse como una función, se deben cumplir las siguientes condiciones:

1. El dominio $\text{Dom}(f) = A$. Esto es, el conjunto de los primeros elementos de todos los pares ordenados de la relación es el dominio de la función y también es igual al conjunto A . Significa que todos los elementos del conjunto A forman parte de la función y es el dominio. Por otro lado, el segundo elemento del par ordenado y , es un elemento del conjunto B (que se denomina el codominio de la función) no necesariamente todos los elementos del conjunto B deben ser parte de la función.
2. Si (a,b) y (a,c) que pertenecen a f , entonces $b=c$. Esto significa que cada elemento del dominio deberá estar relacionado sólo con un elemento del codominio.

Ejemplo:

Sean

$A = \{1, 2, 3, 4, 5\}$

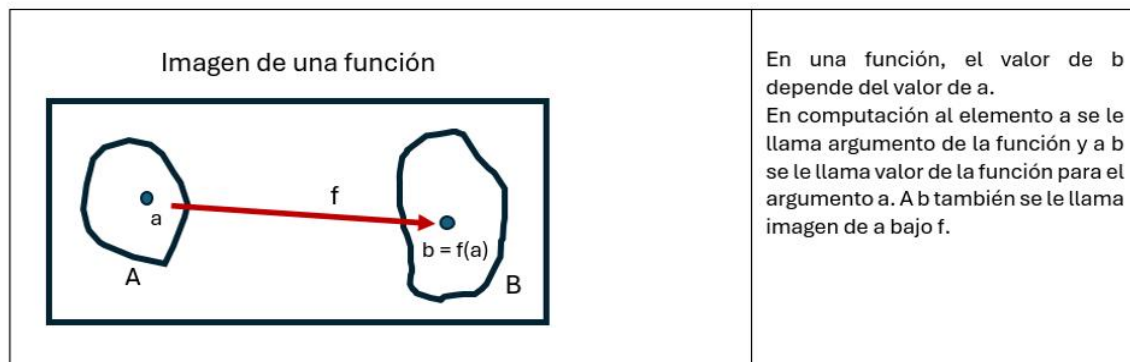
$B = \{a, b, c, d\}$

Determinar si las siguientes relaciones son también funciones:

- a) $R = \{(1,b), (2,c), (3,d), (4,b), (5,d)\}$
 - b) $R = \{(1,a), (2,c), (1,b), (3,a), (4,c), (5,c)\}$
 - c) $R = \{(1,a), (2,a), (3,a), (4,a), (5,a)\}$
 - d) $R = \{(1,b), (2,c), (3,d)\}$
-
- a) Sí es una función. La relación del inciso a) sí es una función, pues cumple con las dos condiciones, todos los elementos del conjunto A forma parte de la relación, y cada elemento del dominio está relacionado con un solo elemento del codominio.
 - b) No es una función. Aunque satisface la primera condición, pues todos los elementos del dominio forman parte de la función, la segunda condición no se cumple, pues existen pares ordenados como $(1,a), (1,b)$ cuyo primer elemento del par ordenado tiene diferente codominio $a \neq b$.
 - c) Sí es una función, pues en los pares ordenados se incluyen todos los elementos del dominio y no existe un elemento del dominio que tenga dos codominios distintos.
 - d) No es una función, pues el dominio de la función es diferente al conjunto A . $\text{Dom}(f) \neq A$.

La propiedad 2 de la definición de función afirma que si $(a,b) \in f$, entonces b se determina de manera única por a . Por lo tanto, se puede escribir $b = f(a)$.

Esto significa que el valor de b está en función de a , es decir, el valor de b depende del valor de a . También se dice que a es la variable independiente y b la variable dependiente.



Dado el valor de la variable independiente y la función, se puede determinar el valor de la variable dependiente.

$$y = \cos(x)$$

$$y = x^2 + 3x - 1$$

donde

x es la variable independiente
 y es la variable dependiente

Una función puede estar definida para 2 o más parámetros o variables independientes, además de incluir la condición de que para éstos siempre se obtenga como resultado un solo valor. Algunos ejemplos:

$y = \text{mod}(a,b)$ Donde “mod” es la función que obtiene el residuo de dividir a/b
 $y = \text{divide}(a,b)$ Donde “divide” es la función que obtiene el resultado de dividir a/b
 $y = \text{resta}(a,b)$ Donde “resta” es la función que obtiene el resultado de restar $a - b$

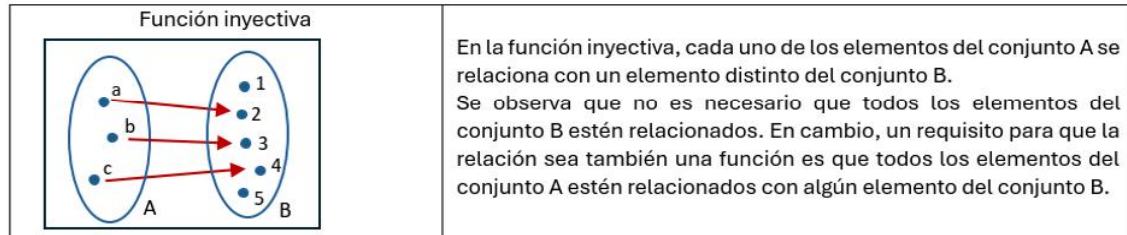
Por lo general, los nombres de las funciones son coherentes con la acción que realizan. Por ejemplo, “divide(a,b)” es una función que devuelve el resultado de dividir a/b . Aunque es recomendable que un nombre de función sea significativo, también se puede utilizar como nombre de una función alguna letra del alfabeto como f , g , h , etc. Como las funciones son relaciones, se pueden representar de distintas formas: como conjuntos, matrices, ecuaciones, algoritmos y grafos, lo cual facilita su gestión.

Tipos de funciones

De acuerdo con las características de las funciones, éstas se pueden clasificar en: inyectiva, sobreyectiva y biyectiva.

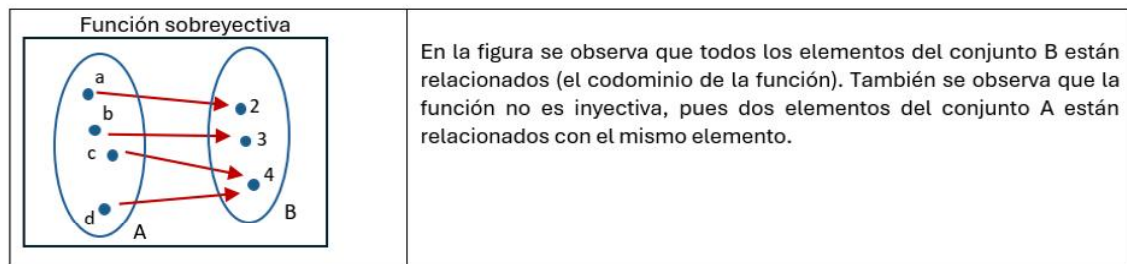
Función inyectiva

Una función $f: A \rightarrow B$ se llama uno a uno (o inyectiva) cuando a cada elemento distinto del conjunto A le corresponde un elemento distinto del conjunto B, esto es, para toda $a, a' \in A$, si $f(a) = f(a')$ implica que $a = a'$.



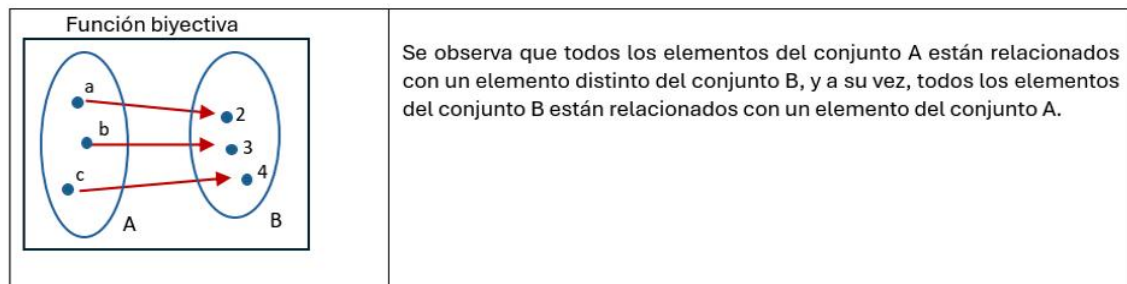
Función sobreyectiva

Una función $f: A \rightarrow B$ se llama sobreyectiva (o suprayectiva) si el conjunto de los segundos elementos de los pares ordenados de la función es igual al conjunto B, dicho de otra forma, la función es sobreyectiva si $\text{Cod}(f) = B$.



Función biyectiva

Una función es biyectiva cuando se cumple con la condición de ser inyectiva y sobreyectiva a la vez.



Ejemplo:

Determinar si las siguientes funciones son inyectiva, sobreyectiva o biyectiva.

- a) $A = \{a, b, c, d, e\}$
 $B = \{1, 2, 3\}$
 $f = \{(a,2), (b,3), (b,2), (c,3), (e,1)\}$
- b) $A = B = \{1, 2, 3, 4, 5\}$
 $f = \{(1,3), (2,5), (4,1), (3,2), (5,4)\}$

El inciso a) no es inyectiva, los pares ordenados $(a, 2)$ y $(b,2)$ tienen el mismo elemento en el codominio. Sí es una función sobreyectiva, pues todos los elementos del codominio, están en la relación. No es biyectiva porque no cumple con la condición de ser inyectiva y sobreyectiva.

El inciso b) Es una función inyectiva pues cada elemento del dominio tiene exclusivamente una imagen del codominio. También es sobreyectiva porque todos los elementos de codominio forman parte de la función. Por tanto, es una función biyectiva.

2.9 Aplicaciones de las relaciones y las funciones en la computación

Las funciones son una clase especial de relación y se utilizan prácticamente en todas las áreas de la matemática. Una de las fortalezas de todos los lenguajes de programación es contar con funciones integradas que realizan procedimientos específicos como $\sin(x)$, $\text{abs}(x)$, $\text{mod}(x,y)$, $\text{div}(a,b)$, etc. Estas funciones incrementan la potencia de los lenguajes.

Los lenguajes de programación también permiten crear funciones con características especiales para resolver problemas específicos. Estas funciones se utilizan con mucha frecuencia porque dan claridad en la codificación de un programa y hacen más compacto y legible el código fuente. La comprensión de la teoría de conjuntos, de las relaciones y de las funciones facilitan la gestión de las funciones y bases de datos en programación.

Tema 3

Lógica matemática

El ser humano expresa y comunica sus ideas, pensamientos y sentimientos a través de un lenguaje verbal, pero también se comunica por medio de gestos, miradas, movimientos, etc. Cuando la comunicación se produce a través de un lenguaje verbal, se utiliza un conjunto de palabras y reglas gramaticales para formar oraciones. De acuerdo con su estructura e intención, las oraciones se clasifican en declarativas, imperativas, exclamativas e interrogativas. Para la lógica matemática se destacan las oraciones declarativas que expresan una realidad concreta y que pueden evaluarse como falsas o verdaderas. Las oraciones declarativas son el bloque básico de construcción de la lógica matemática que se encarga del estudio formal de la lógica para estructurar oraciones en forma coherente y con un significado específico.

En este tema, el estudiante aprenderá a expresar oraciones declarativas como proposiciones lógicas y verificar si un razonamiento es correcto o incorrecto. Como menciona Jiménez: "Al desarrollar la lógica matemática se ejercita el pensamiento abstracto, es posible generalizar la información usando el razonamiento tanto inductivo como deductivo y se pueden llevar a cabo cálculos matemáticos complejos" (Jiménez, p. 116).

Competencia específica: Analiza y resuelve problemas computacionales utilizando las técnicas básicas de lógica e inducción matemática.

3.1 Lógica proposicional

En filosofía y lógica, el término proposición se utiliza para referirse a entidades portadoras de un valor de verdad. Para formar las proposiciones lógicas se utilizan las oraciones declarativas o enunciativas como por ejemplo “El sol es una estrella”. En lógica, se considera a la proposición como la expresión verbal de un juicio y al emitir un juicio se puede evaluar como falso o verdadero. Algunos ejemplos de proposiciones son:

- El equipo de fútbol mexicano ganó el partido
- El jugador de Barcelona juega como defensa
- Los españoles llegaron a México en 1519
- La luna es una estrella

En los ejemplos anteriores se puede observar que las proposiciones pueden ser falsas o verdaderas. El concepto de proposición implica que la expresión se pueda evaluar, sin importar su valor de verdad.

La lógica estudia la forma del razonamiento humano, su disposición para pensar en forma coherente. El razonamiento utiliza reglas y técnicas para determinar si un teorema es falso o verdadero. La lógica tiene un lenguaje exacto que busca precisión en la expresión de las oraciones. Algunos conceptos importantes que se requieren conocer para abordar el tema de lógica matemática se mencionan a continuación.

Axioma. Proposición tan clara y evidente que se admite sin demostración. En matemáticas, se refiere a cada uno de los principios fundamentales e indemostrables sobre los que se construye una teoría (por ejemplo, $2+2=4$).

Postulado. Proposición cuya verdad se admite sin pruebas para servir de base en posteriores razonamientos. Idea que sustenta una persona, un grupo o una organización.

Teorema. Proposición demostrable lógicamente partiendo de axiomas, postulados o de otras proposiciones ya demostradas.

Hipótesis. Se refiere a una o más proposiciones iniciales en las cuales se sustenta la conclusión de un argumento o teorema. Las hipótesis deben ser convincentes y explícitas.

Argumento. Un argumento consiste en una o más hipótesis y una conclusión, de forma que la conclusión se apoye en las hipótesis. También, un argumento es una serie de proposiciones interrelacionadas que conforman una proposición más compleja a la cual se le llama teorema. Por lo general, los argumentos lógicos tienen la forma de un condicional.

3.1.1 Proposiciones simples y compuestas

Una proposición es un enunciado, frase o expresión matemática que puede evaluarse como falsa o verdadera. Las proposiciones son los bloques básicos de construcción de la lógica matemática, se clasifican en proposiciones simples y proposiciones compuestas.

Proposiciones simples. Las proposiciones simples también se denominan proposiciones atómicas, son las proposiciones más básicas que existen, aquellas que no se pueden dividir en proposiciones más simples. Por ejemplo: “Hoy es un día lluvioso”, si se intenta dividir aún más este enunciado, se pierde el sentido de lo que se quiere comunicar.

Proposiciones compuestas. Las proposiciones compuestas o moleculares se forman al unir una o más proposiciones simples utilizando conectores u operadores lógicos que permiten expresar proposiciones más complejas.

Operadores lógicos o términos de enlace

Un operador lógico es un símbolo o palabra que sirve para conectar proposiciones, los operadores lógicos también reciben el nombre de términos de enlace o conectores lógicos.

Los operadores lógicos que permiten formar proposiciones compuestas son:

- El operador and (y)
- El operador or (o)
- El operador not (no)
- El operador condicional (Si ... entonces)

Al unir una o más proposiciones atómicas con un término de enlace, se obtiene una proposición compuesta (o proposición molecular).

Por ejemplo, se tienen las proposiciones simples:

- Hoy es viernes
- Hoy hay clase

Estas proposiciones simples que no se pueden dividir más y que pueden ser evaluadas como ciertas o falsas, se pueden combinar para formar proposiciones más complejas utilizando los términos de enlace, por ejemplo:

La proposición simple: “Hoy es viernes”, se une con el operador lógico “no” y se forma la proposición compuesta: “Hoy no es viernes”. Las proposiciones simples: “Hoy es viernes” y “Hoy hay clase”, se pueden combinar con el operador lógico “y” para formar la proposición compuesta “Hoy es viernes y hay clase”. Como se puede ver en la proposición compuesta “Hoy no es viernes”, el término de enlace aparece después de la palabra “Hoy”, pero pudiera aparecer en la posición en la que se escuche mejor. En el caso de la proposición compuesta “Hoy es viernes y hay clase”, se puede notar que, aunque la palabra “hoy” aparece en cada una de las proposiciones simples, no es necesario repetir la palabra en la proposición compuesta: “Hoy es viernes y hoy hay clase”.

En el lenguaje coloquial es posible cambiar el orden de las palabras o eliminar algunos términos para dar claridad a la frase. Por lo general el término de enlace “y” se coloca entre las dos proposiciones simples, aunque existen algunas variantes para expresar la conjunción de proposiciones.

Proposiciones válidas y no válidas

Una proposición válida, es un enunciado declarativo, mientras que las proposiciones no válidas son enunciados exclamativos, interrogativos o imperativos; es decir, aquellos que no pueden evaluarse como falsos o verdaderos. De los siguientes ejemplos, especifique cuáles son proposiciones válidas y cuáles son proposiciones no válidas:

P = Este refresco no tiene azúcar

Q = Haz la tarea por favor

R = La ballena azul no vive en el mar de Cortés

S = ¿Cuántos años tienes?

T = Las manzanas son verdes

U = Por favor, digitaliza las páginas del libro que están marcadas con color azul

V = Mi mamá vende bocoles en el mercado

W = ¡Qué paisaje tan hermoso!

El operador and (y)

El operador and, también llamado el operador de conjunción permite conectar dos proposiciones que tienen que ser ambas verdaderas para que el resultado de la conjunción sea verdad. Su símbolo es $\mathbf{\wedge}$. En caso de que alguna de las dos proposiciones unidas por el operador de conjunción sea falso, el resultado de la conjunción es falso.

Por ejemplo, En la proposición compuesta: “Mi mamá prepara la comida si y sólo si la estufa está encendida y hay gas en la estufa”. Sean las proposiciones simples:

P = La estufa está encendida

Q = Hay gas en la estufa

R = Mi mamá prepara la comida

En esta proposición compuesta, se tienen 3 proposiciones simples. La proposición simple: “Mi mamá prepara la comida” es verdadera solamente en el caso de que las dos proposiciones simples P y Q sean verdaderas. Es decir, es necesario que haya gas en la estufa y también que la estufa esté encendida, las dos condiciones tienen que ser verdaderas.

Este es el comportamiento del operador “Y”.

Tabla de verdad del operador AND o “Y”

P	Q	$P \wedge Q$
0	0	0
0	1	0
1	0	0
1	1	1

El operador or (o)

El operador or, también llamado operador de disyunción permite conectar dos proposiciones. Para que el resultado de la disyunción sea verdadero, sólo se requiere que alguna de las dos proposiciones sea verdadera, aunque también ambas proposiciones pueden ser verdaderas.

Su símbolo es **V**. En caso de que las dos proposiciones unidas por el operador de disyunción sean falsas, el resultado de la disyunción también es falso.

Por ejemplo, en el enunciado: “Matías juega Básquetbol o juega Fútbol”.

De la proposición compuesta anterior, se obtienen las siguientes proposiciones simples:

P = Matías juega básquetbol

Q = Matías juega fútbol

En este ejemplo, para que la oración compuesta sea verdad, Matías debe jugar básquetbol o jugar fútbol, con una sola de las proposiciones simples que sea verdadera, es suficiente para que la proposición compuesta sea verdadera; pero también la proposición compuesta es verdadera si Matías juega ambos deportes. Este es el comportamiento del operador de disyunción (operador OR).

Tabla de verdad del operador OR

P	Q	$P \vee Q$
0	0	0
0	1	1
1	0	1
1	1	1

El operador not (no)

El operador lógico de negación (not) tiene como función negar una proposición. Esto significa que, al aplicar el operador de negación a una proposición verdadera, la proposición resultante es falsa, y si la proposición es falsa, al negarla el resultado es una proposición verdadera. A diferencia de los operadores AND y OR, el operador NOT funciona con una sola proposición.

Por ejemplo, la proposición:

P = “El carro es azul”, se puede negar para obtener la proposición “El carro no es azul”.

Tabla de verdad del operador NOT

P	P'
1	0
0	1

El operador condicional (si...entonces)

Una proposición condicional o implicación está formada por dos proposiciones las cuales pueden ser simples o compuestas, estas proposiciones se denominan antecedente y consecuente.

Sean P y Q dos proposiciones. Una proposición condicional se expresa: $P \rightarrow Q$.

Y se lee “Si P entonces Q”.

Al antecedente también se denomina hipótesis o premisa, mientras que el consecuente se denomina tesis o conclusión. Una proposición condicional es falsa, cuando la premisa es falsa y la conclusión es verdadera. La proposición condicional es verdadera en cualquier otro caso.

Tabla de verdad del operador condicional (Si.. entonces)

P	Q	$P \rightarrow Q$
0	0	1
0	1	1
1	0	0
1	1	1

Por ejemplo, en la proposición “Si Juan mide 1.2 m o más, entonces puede subirse a este juego mecánico”. Sean las proposiciones:

P = “Juan mide 1.2m o más”

Q = “Juan puede subirse a este juego mecánico”

Para entender mejor el comportamiento del operador condicional se analizan los casos:

1. En el caso de que la proposición sea falsa, es decir, $P = 0$, significa que Juan no mide 1.2 o más, por lo que no hay ninguna condición que se deba de cumplir. Si puede subirse o no al juego mecánico no tiene nada que ver con el antecedente, puede subirse por otras causas, por tanto, la condición es verdad. Cuando no se cumple el antecedente, no importa el consecuente, igual la condicional es verdadera.
2. Si $P=1$ y $Q = 1$, significa que Juan mide 1.2m o más y Juan puede subirse al juego mecánico, por lo que la condición se cumple.
3. En el caso de $P=1$ y $Q=0$ significa que Juan mide 1.2m o más y a pesar de cumplir el antecedente, Juan no puede subirse al juego mecánico, por lo que el resultado de la condición es falso. Cuando el antecedente se cumple y el consecuente no se cumple, la condición es falsa. Este es el único caso en que el resultado de una condición es falso.

En una proposición condicional es importante el orden en que aparecen el antecedente y el consecuente. Cuando no se cumple el antecedente, no se puede cumplir el consecuente:

Si María va a la torre Eiffel, entonces María visita Francia.

Pero no necesariamente se cumple que:

Si María va a Francia, entonces María visita la torre Eiffel.

El operador or exclusivo (XOR)

Además de los operadores básicos (AND, OR, NOT, SI..ENTONCES) existen otros operadores como el operador XOR, cuyo funcionamiento es similar al OR, con la diferencia de que el resultado es verdadero solamente en el caso de que una sola de las proposiciones sea verdadera, pues si ambas son verdaderas, el resultado es falso. El símbolo que representa al operador XOR es $\oplus$. Una equivalencia del operador XOR es la siguiente:

$$P \oplus Q = (P \wedge Q') \vee (P' \wedge Q)$$

Al operador OR exclusivo también se le llama disyunción exclusiva y se lee: “P o Q, pero no ambas”. Por ejemplo, se tienen las proposiciones:

P = “La medalla es de oro”

Q = “La medalla es de acero”

Para el caso de la disyunción exclusiva sólo una de las dos proposiciones que conforman la proposición compuesta es verdadera, es decir: “O la medalla es de oro, o la medalla es de acero, pero no es de ambos metales a la vez”.

Tabla de verdad del operador OR exclusivo

P	Q	$P \oplus Q$
0	0	0
0	1	1
1	0	1
1	1	0

Precedencia de los operadores lógicos

La precedencia de los operadores lógicos determina el orden en que se llevan a cabo las operaciones en una expresión de acuerdo con la jerarquía de los operadores. La prioridad y la forma en que se asocian los operadores lógicos afectan el resultado de la evaluación de una expresión lógica o aritmética. Cuando dos operadores tienen la misma jerarquía, entonces las operaciones se empiezan a realizar de izquierda a derecha, conforme aparecen en la expresión.

Jerarquía	Operador
1°	()
2°	'
3°	$\wedge$
4°	$\vee$
5°	$\rightarrow$

Para modificar la jerarquía de los operadores o simplemente para hacer más evidente el orden en que se llevan a cabo las operaciones siempre es posible utilizar los paréntesis. Es importante entender el orden en que se realizan las operaciones porque podría interpretarse en forma errónea. Según Rosen: “Traducir oraciones del lenguaje natural, como el español, a expresiones lógicas es una parte esencial de la especificación tanto de sistemas de hardware como de software. Los ingenieros de software y de sistemas reciben los requerimientos en lenguaje natural y producen especificaciones precisas y sin ambigüedades que pueden usarse como base para el desarrollo de sistemas” (Rosen, p. 10).

3.1.2 Tablas de verdad

Una tabla de verdad es una herramienta que muestra todas las posibles combinaciones de los valores de una proposición simple o compuesta. Para una tabla de verdad con dos proposiciones, se tienen 4 resultados, si la tabla tiene 3 proposiciones distintas se producen 8 resultados, para 4 proposiciones se tienen 16 resultados. El número de renglones en la tabla de verdad es 2^n , que son todas las combinaciones posibles de los valores de verdad de las

proposiciones; donde n representa el número de proposiciones simples que forman parte de la proposición compuesta.

Para poder asignar valores de verdad a una proposición compuesta, es necesario conocer el comportamiento de los términos de enlace y utilizar tablas de verdad para analizar los valores de un conjunto de proposiciones simples que forman la proposición compuesta. Cada uno de los renglones de la tabla muestra las combinaciones de los valores de las proposiciones. El número de renglones en la tabla de verdad depende del número de variables o proposiciones involucradas, así como de la cantidad de proposiciones requeridas por el conector y el número de conectores utilizados. Las tablas de verdad proporcionan un método mecánico para comprobar la validez y el resultado de una proposición lógica. El método es el siguiente:

1. Se escriben todas las combinaciones posibles de los valores de verdad para las proposiciones atómicas (siguiendo la secuencia binaria a partir de 0).
2. Se determinan los valores de verdad para todas las proposiciones compuestas y la conclusión del razonamiento.
3. Se buscan las líneas que presentan todas las premisas como proposiciones ciertas, si la conclusión también es cierta para cada una de estas líneas, entonces el razonamiento es válido. “Pero si hay alguna línea para la que todas las premisas son ciertas y la conclusión es falsa, el razonamiento no es válido y la conclusión no es una consecuencia lógica”.

En lógica matemática, se utilizan reglas de inferencia, las cuales tienen un comportamiento predefinido. Por ejemplo, existe una regla de inferencia que dice que, si hay una disyunción de dos proposiciones P o Q , y la proposición P no se cumple, entonces forzosamente Q ha de ser cierta, para que la disyunción sea verdadera; esta regla se expresa:

$$P \vee Q \text{ y } \neg P$$

Cuando se utiliza una tabla de verdad para el análisis de una regla de inferencia el primer paso es especificar todas las combinaciones posibles de los valores de verdad para cada proposición atómica, en este caso P y Q . Posteriormente, con base en los valores de las proposiciones atómicas, se registran en la tabla los valores de las proposiciones moleculares.

Tabla de verdad de la proposición compuesta $(P \vee Q) \wedge \neg P$

P	Q	$P \vee Q$	$\neg P$
0	0	0	1
0	1	1	1
1	0	1	0
1	1	1	0

Una vez que se han registrado los valores tanto de las proposiciones atómicas como de las proposiciones moleculares. Como siguiente paso se analizan los renglones en donde todas las premisas son verdaderas (en este caso $p \vee q$ y $\neg p$). Al observar la tabla de verdad, se puede notar que el renglón 2 es el único renglón que cumple con esta condición.

De acuerdo con el ejemplo planteado, se puede notar que el valor de q es verdad, cuando las dos proposiciones moleculares son ciertas.

3.1.3 Tautologías, contradicción y contingencia

Una tautología es aquella proposición molecular que es cierta para todos los valores de verdad de las proposiciones atómicas que la componen. Un ejemplo típico es que se cumple la proposición P , o no se cumple la proposición P . alguna de las dos proposiciones ha de ser cierta. Su representación formal:

$$P \vee \neg P$$

Representando la proposición molecular mediante la representación de tablas de verdad se tiene

P	$\neg P$	$P \vee \neg P$
0	1	1
1	0	1

En la tabla se puede observar que, si la proposición P es verdadera, la negación de P es falsa, sin importar el valor de P el resultado de $P \vee \neg P$ siempre es verdadero, por lo tanto, se concluye que es una tautología.

Las tautologías son importantes en lógica matemática porque al tener un resultado verdadero para todos los valores de verdad, se consideran reglas de inferencia que se pueden utilizar para realizar demostraciones de teoremas o para inferir resultados de proposiciones desconocidas.

Una contradicción o también llamada una proposición absurda es aquella proposición molecular cuyo resultado es falso para todos sus valores de verdad. Una de las contradicciones más conocidas es: “No es cierto que se cumple P y $\neg P$ al mismo tiempo”, lo cual se puede simbolizar de la siguiente manera:

P	$\neg P$	$P \wedge \neg P$
0	1	0
1	0	0

En la tabla se puede observar que, sin importar el valor de P , el resultado final de la proposición molecular ($p \wedge \neg p$) siempre será falso. A esto se le llama contradicción.

Por otro lado, una contingencia es una proposición compuesta cuyos valores en sus diferentes líneas de la tabla de verdad, dan como resultado unos y ceros, por lo general, cualquier proposición molecular es una contingencia. También se le llama inconsistencia o falacia.

3.1.4 Equivalencias lógicas

Dos proposiciones son equivalentes lógicamente si ambas proposiciones muestran los mismos valores en sus tablas de verdad. Para demostrar si hay una equivalencia lógica entre dos proposiciones se debe demostrar que la tabla de verdad de la primera proposición es equivalente a la tabla de verdad de la segunda proposición.

En lógica matemática los símbolos que se pueden utilizar para representar una equivalencia lógica son los siguientes:

$\iff$ $\longleftrightarrow$ $\Leftrightarrow$ $\equiv$

Por ejemplo, demostrar que $\neg(P \rightarrow Q) \equiv (P \wedge \neg Q)$

Para demostrar que ambas proposiciones son equivalentes, se utiliza el método especificado en el punto 3.1.2, primeramente, se diseña una tabla de verdad con las proposiciones simples y proposiciones moleculares involucradas, como se muestra en la tabla 3.1; posteriormente, se verifica que las dos proposiciones sean equivalentes revisando que en cada renglón de la tabla de verdad, el resultado sea el mismo en ambas proposiciones.

P	Q	$P \rightarrow Q$	$\neg Q$	$\neg(P \rightarrow Q)$	$P \wedge \neg Q$
0	0	1	1	0	0
0	1	1	0	0	0
1	0	0	1	1	1
1	1	1	0	0	0

Tabla 3.1 Proposiciones equivalentes

3.1.5 Reglas de inferencia

La identificación de proposiciones simples y compuestas fundamenta la base para realizar razonamientos inductivos y deductivos dentro de un proceso de análisis. Las frases y oraciones que se expresan en lenguaje natural pueden ser traducidas a expresiones lógicas para su análisis y determinar su valor de verdad, así mismo se pueden manipular y utilizar reglas de inferencia para obtener resultados con base en un razonamiento válido. Las proposiciones se simbolizan y los resultados que se obtienen son los mismos independientemente del contenido de cada proposición. Por ejemplo, la proposición $P \rightarrow Q$ es válida para cualquier proposición en lenguaje español.

Cuando se simbolizan las proposiciones utilizando un lenguaje formal, se avanza hacia una parte importante de la lógica matemática: la inferencia lógica. Las reglas de inferencia son un conjunto de reglas que permiten partir de las premisas para llegar a una conclusión mediante un proceso de deducción o inducción, “La idea de la inferencia se puede expresar de la manera siguiente: de premisas verdaderas se obtienen sólo conclusiones que son verdaderas”.

Para llevar a cabo un proceso de razonamiento deductivo se debe iniciar con la identificación de las proposiciones y reglas de inferencia a utilizar. Las principales reglas de inferencia que se pueden utilizar son:

Modus Ponendo Ponens

Este vocablo proviene del latín que significa: “Método que afirmando el antecedente, afirma el consecuente”. Para esta regla se tienen dos premisas $P \rightarrow Q$ y P que son verdaderas. Recordando la tabla de verdad de una proposición condicional, se puede notar que, si se cumple P , entonces también se cumple Q . Por lo tanto, de estas dos proposiciones se puede deducir Q . Se dice que Q es consecuencia lógica de las premisas $P \rightarrow Q$ y P .

Ejemplo 1:

Si llueve, entonces llevo paraguas.

Llueve, por lo tanto, llevo paraguas.

Lo anterior se simboliza:

P: Llueve

Q: Llevo paraguas

1) $P \rightarrow Q$ P

2) P P

3) Q PP 1,2

Primeramente, se identifican y simbolizan las proposiciones atómicas y moleculares, como se muestra en el ejemplo 1, se enumeran las proposiciones dadas, y se agrega "P" para indicar que es una premisa, y en la siguiente línea numerada, se agrega la conclusión, indicando la regla de inferencia (en este caso PP, Ponendo Ponens) y el número de línea de cada proposición utilizada en la regla de inferencia (líneas 1 y 2).

A continuación, simbolice los ejercicios 2 y 3, tomando como base el ejemplo 1, identifique proposiciones atómicas y moleculares, posteriormente utilice la regla de inferencia PP para obtener una conclusión.

Ejercicios 2 y 3:

2. "Si hoy es lunes, entonces Juan va a trabajar"
Se cumple que: "Hoy es lunes", ¿Qué se puede deducir?
3. "Si México colinda con Estados Unidos, entonces Estados Unidos está en América"
Se cumple que: "México colinda con Estados Unidos", ¿Qué se puede deducir?

Doble negación.

Es una regla que niega dos veces una proposición. A diferencia de las reglas Modus Ponendo Ponens esta regla sólo requiere una premisa. La abreviatura formal para la doble negación es DN.

1. Juan es mi mejor amigo
DN: No es cierto que Juan no es mi mejor amigo
2. Antony pinta un cordero
DN: No es cierto que Antony no pinta un cordero
3. María conoció el desierto de Sahara
DN: No es verdad que María no conoció el desierto de Sahara

Simbolización:

P = Antony pinta un cordero

- | | |
|-----------------|------|
| 1) P | P |
| 2) $\neg\neg P$ | DN 1 |

Dado que $\neg P$ es la negación de P, así también $\neg\neg P$ es la negación de $\neg P$. Utilizando la regla de la doble negación, se obtiene una proposición equivalente a P, que es $\neg\neg P$.

Modus Tollendo Tollens

La regla de Modus Tollendo Tollens requiere de dos premisas: una proposición condicional y una proposición que niega el consecuente del condicional. Por medio de esta regla, se obtiene la negación del antecedente.

Ejemplo:

1. Si dibujo un cordero entonces tengo una hoja de papel
2. No tengo una hoja de papel
3. Por lo tanto, no dibujo un cordero

Si la luna es una estrella, entonces la luna tiene luz propia

La luna no tiene luz propia

∴

La luna no es una estrella

Si Joel es mi hermano, entonces Joel tiene una hermana que se llama Carmen.

Joel no tiene una hermana que se llama Carmen

∴

Joel no es mi hermano

Si dibujo un cordero, entonces tengo una hoja de papel.

No tengo una hoja de papel

∴

No dibujo un cordero

Simbolización:

P = Yo dibujo un cordero

Q = Tengo una hoja de papel

- | | |
|----------------------|--------|
| 1) $P \rightarrow Q$ | P |
| 2) $\neg Q$ | P |
| 3) $\neg P$ | TT 1,2 |

Ley de adjunción

Cuando se tienen dos proposiciones que son verdaderas, entonces, se pueden unir en una proposición molecular utilizando el término de enlace “y”, para obtener una proposición verdadera. La regla de adjunción se simboliza como A.

Ejemplo:

Premisa 1. María es alta

Premisa 2. María juega básquetbol

Entonces utilizando la regla de adjunción, se puede concluir la proposición verdadera

María es alta y María juega básquetbol

Simbolización:

Q= María es alta

R= María juega básquetbol

- | | |
|-----------------|-------|
| 1) Q | P |
| 2) R | P |
| 3) $Q \wedge R$ | A 1,2 |

Ley de simplificación

La regla de simplificación se simboliza como “S” y es la regla opuesta a la regla de adjunción.

En este caso, se tiene una premisa que contiene un término de enlace “y” de la siguiente manera:

María tiene 18 años y estudia Ingeniería en Sistemas Computacionales

De esta proposición se pueden obtener dos proposiciones, ambas verdaderas. ¿Qué se puede concluir?

P= María tiene 18 años

Q= María estudia Ingeniería en Sistemas Computacionales

Si la premisa es cierta, entonces cada una de las conclusiones que se derivan de la conjunción, también son ciertas.

Simbolización:

P=María tiene 18 años

Q= María estudia Ingeniería en Sistemas Computacionales

- | | |
|-----------------|-----|
| 1) $P \wedge Q$ | P |
| 2) P | S 1 |
| 3) Q | S 1 |

¿A qué conclusiones se puede llegar con base en la Ley de simplificación a partir de las siguientes proposiciones?

El ser humano vive en sociedad y el ser humano necesita el agua para vivir

∴

Se concluye que:

P = El ser humano vive en sociedad

Y también se concluye que:

Q = El ser humano necesita agua para vivir

A Francisco le gusta esquiar y Francisco viaja a la montaña nevada

∴

Se concluye que:

P = A Francisco le gusta esquiar

Y también se concluye que:

Q = Francisco viaja a la montaña nevada

Paloma habla español y Pedro habla inglés

∴

Se concluye que:

P = Paloma habla español

Y también se concluye que:

Q = Pedro habla inglés

Modus Tollendo Ponens

El nombre de la regla Modus Tollendo Ponens proviene del latín que significa: “Negando un elemento de la disyunción, se afirma el otro elemento de la disyunción” y esta frase tiene sentido pues en una disyunción, uno de los dos elementos debe ser verdadero para que la disyunción sea verdadera, y si uno de los elementos es falso, entonces el segundo elemento de la disyunción debe ser verdadero. La disyunción dice que por lo menos una de las dos partes se cumple, y si en este caso uno de los elementos no se cumple, se afirma que el segundo elemento es verdadero.

¿A qué conclusiones se pueden llegar con base en las siguientes proposiciones?:

El globo tiene Helio o el globo tiene aire

El globo no tiene aire

∴

Juan Carlos estudia Medicina o estudia Mecatrónica

Juan Carlos no estudia Mecatrónica

∴

Pepe ha terminado de leer el libro de “El Principito” o ha terminado de leer el libro de “Marianela”

Pepe no ha terminado de leer el libro de "El Principito"

∴

Simbolización:

T = Juan Carlos estudia Medicina

U = Juan Carlos estudia Mecatrónica

- 1) $T \vee U$ P
- 2) $\neg U$ P
- 3) T TP 1, 2

Ley del silogismo hipotético (HS)

En lógica proposicional, el silogismo hipotético es una regla de inferencia válida que requiere de dos proposiciones condicionales y que ofrece como conclusión una nueva proposición condicional. Esta regla también se denomina regla de cadena, argumento cadena o principio de transitividad de la implicación. En el caso de la Ley del silogismo hipotético, y a diferencia de las otras reglas, la conclusión es una nueva proposición condicional.

Si hoy es un día soleado, entonces Juan va a la playa

Si Juan va a la playa, entonces Juan lleva una sombrilla

Se puede concluir que:

Si hoy es un día soleado entonces Juan lleva una sombrilla

Simbolización

O = Si hoy es un día soleado

P = Juan va a la playa

Q = Juan lleva una sombrilla

- 1) $O \rightarrow P$ P
- 2) $P \rightarrow Q$ P
- 3) $O \rightarrow Q$ HS 1,2

Ley del silogismo disyuntivo (DS)

La ley del silogismo disyuntivo es una regla de inferencia válida que requiere una disyunción y dos condicionales para obtener la conclusión.

Ejemplo:

O es un día soleado o está lloviendo

Si es un día soleado entonces Pedro juega básquetbol

Si está lloviendo, entonces Pedro juega videojuegos en casa

¿A qué conclusión se puede llegar con base en las siguientes proposiciones?

Pedro juega básquetbol o Pedro juega videojuegos en casa

Simbolización

P = Hoy es un día soleado

Q = Está lloviendo

R = Pedro juega básquetbol

S = Pedro juega videojuegos en casa

- | | |
|----------------------|------------|
| 1) $P \vee Q$ | P |
| 2) $P \rightarrow R$ | P |
| 3) $Q \rightarrow S$ | P |
| 4) $R \vee S$ | SD 1, 2, 3 |

3.1.6 Argumentos válidos y no válidos

Un argumento es una secuencia de afirmaciones por medio de las cuales se llega a una conclusión. Todas las afirmaciones que conforman el argumento se llaman premisas, suposiciones o hipótesis, y la proposición final que se deduce a partir de las premisas se denomina conclusión. Es decir, un argumento consiste en una o más hipótesis y una conclusión, de forma que la conclusión se apoye en las hipótesis.

Se dice que un argumento es válido si para cualquier valor de las variables proposicionales involucradas en la simbolización que hacen verdaderas a las premisas, hace que la conclusión también sea verdadera. Es decir, de un argumento válido se obtienen conclusiones verdaderas, es imposible por razones formales que las premisas sean verdaderas y la conclusión sea falsa. Cuando se utilizan argumentos inválidos, no se garantiza una conclusión verdadera.

3.1.7 Demostración formal

Utilizando las reglas de inferencia, se puede pasar de un conjunto de afirmaciones lógicas a otra afirmación, por ejemplo, de las proposiciones $(P \rightarrow Q)$ y (P) se puede concluir la proposición Q . En muchas ocasiones, no es posible llegar directamente de las premisas a la conclusión en un solo paso, pero utilizando las reglas de inferencia y realizando pasos sucesivos en forma sistemática, siguiendo cada uno de los pasos permitidos por la regla, se puede alcanzar la conclusión deseada. Bajo este esquema es posible demostrar que la conclusión es una consecuencia lógica de las premisas proporcionadas.

Como se menciona en el párrafo anterior, el propósito de cada movimiento que se realiza es avanzar un paso hacia la meta deseada. La demostración formal se inicia con la especificación de un conjunto de premisas. La inclusión de las premisas está justificada por la regla de las premisas que dice: “Una premisa puede ser introducida en cualquier punto de una deducción” (Suppes, p. 71).

La demostración formal tiene como objetivo alcanzar la conclusión establecida, el propósito de deducir otras proposiciones utilizando las reglas de inferencia es avanzar un paso hacia la conclusión. Generalmente los argumentos lógicos utilizados en la demostración formal son razonamientos resultantes del enunciado del problema que es posible representar, usando notación lógica, como una proposición condicional integrada por varias proposiciones simples, siempre y cuando se identifiquen claramente las proposiciones simples y los conectores lógicos que unen dichas proposiciones. Como se mencionó anteriormente, un argumento o teorema es la proposición que resulta del planteamiento de un problema.

Un argumento o teorema tiene la forma $(P \rightarrow Q)$, en donde P y Q son proposiciones compuestas. A las proposiciones que integran a P y que están conectadas por operadores se les llama hipótesis o premisas, y a la proposición Q se le llama conclusión o consecuente.

La lógica proposicional tiene una limitación en cuanto a su capacidad de representar conocimiento. Existen situaciones en donde las sentencias complejas pierden mucho de su significado cuando se representan mediante la lógica proposicional. Por este motivo se desarrolló una forma lógica más general que tiene la capacidad de representar mayor cantidad de los detalles expresados en las sentencias, a esta forma de representar las sentencias se le denomina lógica de predicados.

3.2 Lógica de predicados

La lógica de predicados, también llamada lógica de conjuntos se fundamenta en considerar que las proposiciones son conjuntos de elementos que tienen una propiedad o característica llamada “predicado” y desde este contexto una proposición puede ser verdadera para un grupo de elementos del conjunto, pero a la vez puede ser falsa para otro grupo de elementos del conjunto.

La lógica de predicados estudia las frases declarativas con mayor grado de detalle, de tal forma que considera la estructura interna de las proposiciones. La lógica de predicados distingue como elementos básicos:

- a) A los objetos (que son los objetos de quien se afirma)
- b) Y el predicado (que se refiere a la relación con el objeto de lo que se está afirmando, es decir, lo que se afirma del objeto)

3.2.1 Cuantificadores

En lógica de predicados se define un conjunto universo o dominio del discurso que contiene a todos los elementos a los cuales está sometido el predicado. Además, se incluyen los cuantificadores “Todos” y “Algunos”. Los cuantificadores se refieren a la cantidad de elementos que pertenecen al conjunto, que pueden ser ningún elemento, más de un elemento

o todos los elementos del universo o del predicado pertenecen al conjunto. Los símbolos para representar los cuantificadores son:

$\forall$ = “Para todo o para todos”

$\exists$ = “Existe alguno, algunos, o al menos un elemento”

3.2.2 Representación y evaluación de predicados

Además del predicado se indica cuál es el dominio para esa proposición, de forma que los enunciados anteriores pueden plantearse de la siguiente manera:

$\forall x p(x) \quad x \in U$

(Para todo x; tal que p, donde x pertenece a U)

$\exists x p(x) \quad x \in U$

(Existe algún elemento x; tal que p, donde x pertenece a U)

La teoría de conjuntos y sus propiedades es la base en lógica de predicados. Además, es importante mencionar que los operadores lógicos que se utilizan en la lógica proposicional también son válidos en lógica de predicados.

3.3 Algebra declarativa

El álgebra declarativa, también llamada álgebra proposicional, se refiere a la estructura algebraica que se forma con expresiones utilizando los conectores lógicos. A una expresión sintácticamente correcta se le llama fórmula bien formada (fbf) o simplemente fórmula. La construcción de fórmulas bien formadas representa la base para realizar los razonamientos lógicos. Una fórmula en lógica de proposiciones se obtiene al aplicar una o más veces las siguientes reglas:

1. Si P es una proposición lógica, entonces es una fórmula bien formada (fbf)
2. Si F es una fbf, entonces también lo es $\neg F$
3. Si P y Q son fbf, entonces también lo es $(P * Q)$ donde * es uno de los operadores binarios válidos en lógica proposicional ($\wedge \vee \rightarrow \leftrightarrow$)
4. Cualquier expresión que no se pueda obtener mediante la aplicación finita de estas reglas, no es una fbf en lógica de proposiciones

3.4 Inducción matemática

La inducción matemática es un razonamiento que permite demostrar que una proposición es falsa o verdadera, partiendo de axiomas, postulados y otras proposiciones ya demostradas.

El proceso de razonamiento inicia con la correcta simbolización de las proposiciones que representan la base para la inducción matemática; posteriormente el análisis y la utilización de reglas de inferencia adecuadas que guíen hacia la conclusión del razonamiento.

Cuando las premisas son ciertas, cada una de las conclusiones que se obtienen también son ciertas. Los pasos que se siguen para realizar demostraciones lógicas son:

1. Iniciar con el paso básico. Leer el problema planteado para identificar información relevante que pueda ser formalizada como proposiciones válidas, es decir, simbolizar las proposiciones a partir del problema planteado. A cada una de las proposiciones, se le asigna una variable distinta.
2. Enumerar las premisas. Las proposiciones que son verdaderas se denominan premisas, las cuales se enumeran y se identifican con la letra “P”, para indicar que la línea está justificada como una premisa válida.
3. Continuar con el paso inductivo. Utilizar las reglas de inferencia y las proposiciones válidas, para identificar nuevas proposiciones que se deriven de las premisas.
4. Enumerar premisas y reglas. Las reglas de inferencia utilizadas, así como las premisas se enumeran, cada línea representa un paso en la inferencia realizada.

Ejemplo:

“Si compro una bicicleta entonces llego temprano a la escuela. Si no llego temprano a la escuela entonces repruebo el semestre. O compro una bicicleta o me levanto temprano. Pero, si no me levanto temprano entonces repruebo el semestre. No me levanto temprano, por lo tanto, compro una bicicleta”.

Simbolización:

P = Compro una bicicleta

Q = Llego temprano a la escuela

R = Repruebo el semestre

S = Me levanto temprano

Inferencia lógica:

- | | | |
|----|------------------------|---------|
| 1) | $P \rightarrow Q$ | P |
| 2) | $\neg Q \rightarrow R$ | P |
| 3) | $P \vee S$ | P |
| 4) | $\neg S \rightarrow R$ | P |
| 5) | $\neg S$ | P |
| 6) | R | PP 4, 5 |
| 7) | P | TP 3, 5 |

3.5 Aplicaciones de la lógica matemática en la computación

La historia de la lógica tiene sus inicios en el siglo III A.C. con la “Teoría silogista” de Aristóteles, quien introdujo los cuantificadores $\forall$ y $\exists$, así como las reglas de inferencia como el silogismo hipotético (Jiménez, p. 163). El uso de la lógica matemática se ha consolidado como una herramienta fundamental para mejorar el hardware y software.

Uno de los usos más comunes de la lógica matemática es el análisis y diseño de los algoritmos que guían el comportamiento de los dispositivos electrónicos como computadoras, teléfonos celulares, electrodomésticos con dispositivos electrónicos, tabletas y laptops entre otros

dispositivos. En términos generales, se puede decir que la lógica matemática fundamenta el proceso de programación de computadoras. Una computadora está construida con circuitos electrónicos que responden a señales eléctricas y que siguen patrones de comportamiento similares a los que estudia la lógica matemática, por lo que una forma de involucrarse en el entendimiento y comprensión del funcionamiento de la computadora es el estudio de la lógica matemática.

Actualmente, las computadoras y otros equipos electrónicos integran módulos de inteligencia artificial para realizar tareas que normalmente requieren inteligencia humana, como el aprendizaje y la toma de decisiones.

Tema 4

Álgebra booleana

Las reglas de la lógica matemática fueron desarrolladas por primera vez por George Boole en 1854, pero debido a que sus principios eran abstractos, no tuvieron aplicación directa, aunque son el fundamento del álgebra de Boole. El primer paso para la construcción de un circuito digital es representar su función booleana, la cual se puede expresar mediante operandos y operadores básicos del álgebra de Boole. Esta función booleana puede ser analizada y optimizada para que requiera menor cantidad de elementos en su representación.

Mediante una interesante aportación, Claude Shannon un siglo después, en 1938, demostró cómo se podían utilizar las reglas de la lógica matemática para diseñar circuitos electrónicos (Rosen, p. 653). Los circuitos de las computadoras y otros componentes electrónicos reciben datos como entrada en formato de dos elementos distintos como 0's y 1's, así, producen una salida binaria utilizando cualquier elemento básico que posea dos diferentes estados.

El álgebra booleana hoy en día es el fundamento para analizar, diseñar y optimizar circuitos eléctricos y electrónicos. En este tema se describe la relación existente entre el álgebra booleana y los circuitos eléctricos.

Competencia específica: Aplica los conceptos y propiedades del álgebra booleana, para optimizar expresiones booleanas y diseñar circuitos básicos con compuertas lógicas.

Introducción

Los sistemas digitales procesan información en formato binario, esta es la importancia del sistema de numeración binario en el uso de las computadoras. Las cantidades binarias pueden representarse mediante cualquier dispositivo que sólo tenga dos estados de operación o dos condiciones posibles. En la vida diaria se pueden mencionar muchos ejemplos donde un objeto puede tener dos estados posibles, por ejemplo, un foco puede estar encendido o apagado, una puerta puede estar abierta o cerrada, un botón puede estar presionado o no, un espejo refleja luz o no refleja luz, puede estar lloviendo o no, el termómetro puede funcionar o no, etc.

Para almacenar música, o información en un CD o en un pendrive también se utiliza el formato binario, por ejemplo, en un CD, la superficie interior del disco, que se encuentra bajo una capa de plástico transparente, se recubre con una capa de aluminio con alta capacidad de reflexión, para grabar se queman hoyos en la cubierta reflectora de aluminio formando “pozos” que no reflejan la luz de la misma manera que las áreas que no fueron quemadas. Estas áreas en las que se queman los “pozos” se consideran 1's y las áreas reflectoras de luz se consideran 0's.

Un pendrive o dispositivo de memoria USB también funciona con un sistema binario, el dispositivo recibe pulsaciones de cargas eléctricas en un rango de 0 a 5 volts (aunque permite tolerancia en los rangos de voltajes, por ejemplo, de 0 a 0.8 volts para representar el 0, y de 3.5 a 5 volts para representar el 1), en los sistemas digitales el valor exacto de un voltaje no es importante. Al recibir las pulsaciones eléctricas procedentes del dispositivo al cual está conectado el pendrive, se establece la conexión entre la computadora y la memoria USB lo que permite que dicha memoria se mantenga en marcha y a la espera de la lectura o grabación de datos. Una vez establecida la conexión, existe un microcontrolador que actúa como una “puerta” entre la memoria de la computadora y el USB, también existe un área en donde los impulsos eléctricos que entran en la conexión se transforman en datos, es decir, en 0's y 1's.

Esencialmente, una memoria USB contiene una cadena de celdas de almacenamiento contiguas, donde se almacena un bit en cada una de sus celdas abriendo o cerrando puertas electrónicas microscópicas. Una puerta puede almacenar 1 bit (abierta es 1, cerrada es 0). Al leer en un pendrive o memoria USB, la corriente eléctrica intenta pasar a través de cada puerta, interpretando las puertas abiertas como 1's y las puertas cerradas como 0's, estos datos son transferidos a la memoria de la computadora. Al grabar datos en un pendrive, de la memoria de la computadora se envían las pulsaciones eléctricas que pasan a través de las puertas correspondientes del pendrive, modificando el estado de las celdas.

Álgebra Booleana

El álgebra booleana, es el álgebra en la cual se utiliza el sistema de numeración binario. Al igual que el álgebra tradicional, en el álgebra booleana también se utilizan letras del alfabeto para denominar variables y formar ecuaciones, con el objetivo de obtener resultados de ciertas operaciones mediante ecuaciones o expresiones booleanas.

En 1854, el matemático George Boole en su libro “Una investigación sobre las leyes del pensamiento” describe la manera en que realizamos decisiones lógicas con base en circunstancias falsa o verdaderas. George Boole propone la idea de que las proposiciones lógicas pueden ser tratadas con herramientas matemáticas. Los métodos que describió se conocen como lógica booleana y el sistema de utilizar símbolos y operadores para describir estas decisiones se llama álgebra booleana.

Las proposiciones lógicas sólo pueden tomar el valor de falso o verdadero (sólo pueden tomar dos valores que pueden ser 0 para representar falso y 1 para representar verdadero). El álgebra booleana constituye un sistema matemático deductivo centrado en los valores 0 y 1, que proporciona operaciones y reglas para trabajar. Las operaciones booleanas básicas son la suma, el producto y el complemento booleano. De la misma forma que en el álgebra tradicional se utilizan los símbolos como “x” y “y” para representar valores numéricos desconocidos, en el álgebra booleana se utilizan símbolos para representar una expresión lógica que tiene uno de los dos valores posibles: falso o verdadero.

Señal eléctrica

Una señal eléctrica es la representación de información que puede aparecer en forma de un valor, o de una cadena de valores de cierta magnitud física. El álgebra booleana trabaja con señales binarias. Los sistemas digitales utilizan señales binarias que se pueden interpretar como falso o verdadero, que provienen de los sensores que mandan información al circuito de control, mismo que lleva a cabo la evaluación para obtener un valor que indicará si se lleva a cabo o no determinada acción como encender un foco, arrancar un equipo de ventilación en un cine o ejecutar una operación matemática en una computadora. Los sensores pueden ser ópticos como los que se usan en tiendas departamentales (de proximidad), magnéticos como los que detectan armas en los aeropuertos, de temperatura como los que se utilizan en sistemas de calefacción, aires acondicionados, refrigeradores o termostatos que controlan el sistema de enfriamiento del motor de un vehículo. También existen sensores de nivel, que utilizan un flotador para controlar la cantidad de líquido existente en un tinaco o cisterna. Todos estos tipos de sensores tienen la capacidad de enviar datos a un circuito de control.

Circuitos lógicos

Un circuito lógico, también llamado circuito digital, es un componente que responde a una o varias entradas de una forma específica. La forma en que un circuito digital responde a una entrada se conoce como lógica del circuito. Existen varios tipos de circuitos digitales, los cuales obedecen a un conjunto de reglas lógicas. Por esta razón a los circuitos digitales también se les conoce como circuitos lógicos.

Casi todos los circuitos lógicos que se utilizan en los sistemas digitales modernos son circuitos integrados (CI). La amplia variedad de circuitos integrados lógicos disponibles en el mercado, ha hecho posible la construcción de sistemas digitales complejos. Los circuitos se clasifican en circuitos combinatorios y circuitos secuenciales. Los circuitos combinatorios no tienen una memoria para almacenar información, por lo que las entradas anteriores y el estado del sistema no afectan su salida. Por otro lado, en los circuitos secuenciales, las entradas del circuito y el estado del sistema retroalimentan al circuito por lo que contienen una memoria

donde se almacena la salida del sistema. Para construir los circuitos combinatorios se utilizan dispositivos de estado sólido llamados compuertas, los cuales son capaces de cambiar los niveles de voltaje (bits). “Los circuitos digitales, llamados compuertas lógicas, pueden construirse a partir de diodos, transistores y resistencias conectados de manera que la salida del circuito sea el resultado de una operación lógica básica (OR, AND y NOT) que se lleva a cabo con las entradas” (Tocci, p. 57).

Con el uso de sistemas digitales es posible resolver problemas de la vida cotidiana mediante la automatización de procesos, para lo cual es necesario realizar un análisis detallado de lo que se quiere lograr y tipos de sensores que se requieren para obtener las señales de entrada para la solución del problema. Una vez analizado el problema, se plantea el funcionamiento del circuito lógico en una expresión matemática, la cual recibe el nombre de función booleana, y cada una de sus variables representa un sensor o dispositivo de entrada que provee una señal de entrada al circuito, modelando de esta forma una situación de la vida cotidiana.

4.1 Teoremas y postulados

En el álgebra booleana, al igual que en la lógica matemática, existen un conjunto de propiedades, las cuales permiten analizar y optimizar las funciones booleanas. Estas propiedades son particularmente útiles en la simplificación del diseño de circuitos. Las propiedades de las expresiones booleanas se mencionan a continuación:

- Están compuestas por literales (A, B, C, ...) y cada una de ellas representa la señal. Un ejemplo es $F = A'B + BC$.
- El valor de las señales o de la función sólo puede ser 0 o 1, o en todo caso falso o verdadero.
- Además de literales, en la expresión booleana se puede tener el valor de 0 o 1. Por ejemplo: $F = A'BD1 + AB'CD + 0$.
- Las literales de las expresiones booleanas pueden estar conectadas por medio de los operadores lógicos AND ($\wedge$), OR ($\vee$) y NOT ($'$). El operador AND es una multiplicación lógica que se representa por medio de un paréntesis, un punto o simplemente poniendo las variables juntas como $(A)(B) = A.B = AB$. El operador OR es una suma lógica que se indica con el signo más “+”, por ejemplo, $A+B$. El operador NOT es la negación o complemento de una señal, el cual se indica por un apóstrofo (A') o una tilde sobre la variable: $\bar{A}$.
- El valor de una expresión booleana se obtiene sustituyendo en cada una de las variables el valor de 0 o 1, teniendo en cuenta el comportamiento de los operadores lógicos.
- Además de las operaciones básicas, también es posible aplicar la Ley de Morgan de forma semejante a como se aplica en la teoría de conjuntos:

$$(ABCD)' = A' + B' + C' + D'$$

$$(A + B + C + D)' = A'B'C'D'$$

El álgebra booleana es un sistema algebraico que consiste en un conjunto B que contiene dos o más elementos y en el que están definidas dos operaciones denominadas suma (operación OR, "+") y producto (operación AND, "."), las cuales satisfacen las siguientes propiedades:

- Existencia de neutros. En B existe el elemento neutro de la suma (0) y el elemento neutro del producto (1), tales que para cualquier elemento x de B:

$$x + 0 = x \quad y \quad x \cdot 1 = x$$
- Propiedad conmutativa. Esta propiedad del álgebra booleana es válida para cada x, y en B:

$$x + y = y + x \quad y \quad x \cdot y = y \cdot x$$
- Propiedad asociativa. Esta propiedad del álgebra booleana es válida para cada x, y, z en B:

$$x + (y + z) = (x + y) + z$$
- Propiedad distributiva. Para cada x, y, z en B:

$$x + (y \cdot z) = (x + y) \cdot (x + z)$$

$$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$$
- Existencia de complementos. Para cada x en B existe un elemento x', llamado complemento de x, tal que:

$$x + x' = 1$$

$$x \cdot x' = 0$$

A partir de las propiedades de las operaciones del álgebra booleana se tienen los siguientes teoremas:

Teorema 1. Idempotencia

$$x + x = x \quad x \cdot x = x$$

Teorema 2. Identidad de los elementos 0 y 1

$$x + 1 = 1 \quad x \cdot 0 = 0$$

Teorema 3. Absorción

$$x + (x \cdot y) = x \quad x \cdot (x + y) = x$$

Teorema 4. Complemento de 0 y 1

$$0' = 1 \quad 1' = 0$$

Teorema 5. Involución

$$(x')' = x$$

Teorema 6. Leyes de Morgan

$$(x + y)' = x' \cdot y' \quad (x \cdot y)' = x' + y'$$

Los teoremas del álgebra de Boole son ecuaciones que son válidas en todos los casos. Las ecuaciones con elementos del álgebra booleana vienen en pares, estos pares se conocen como duales y pueden ser intercambiados entre ellos con el propósito de analizar y simplificar

las ecuaciones. Para obtener el dual de una expresión o ecuación booleana se sustituye el 0 por 1 y el 1 por el 0, al mismo tiempo que se cambia el operador “+” por “.” Así también se cambia el “.” por el “+”. Por ejemplo, el dual de $(x + y)' = x'y'$. Las variables x, y se niegan, cada una por separado, y el operador “+” se cambia por “.” (La operación producto se representa con el símbolo “.”, o simplemente colocando las variables juntas como en este caso)

Para demostrar que los duales pueden ser intercambiados sin alterar los resultados en la figura 4.1 se muestran la tabla con todas las combinaciones posibles de las variables y operadores, observando que las tablas de resultados son iguales. La figura 4.2 muestra los teoremas del álgebra de Boole y sus duales.

x	y	$x + y$	x'	y'	$(x + y)'$	$x' \cdot y'$
0	0	0	1	1	1	1
0	1	1	1	0	0	0
1	0	1	0	1	0	0
1	1	1	0	0	0	0

Como se observa en la tabla:

$$(x + y)' = x' \cdot y'$$

Figura 4.1 Tabla de la función $(x + y)' = x' \cdot y'$

Teoremas del álgebra de Boole

No.	Teorema	Dual
1	$0A = 0$	$1 + A = 1$
2	$1A = A$	$0 + A = A$
3	$AA = A$	$A + A = A$
4	$AA' = 0$	$A + A' = 1$
5	$AB = BA$	$A + B = B + A$
6	$ABC = A(BC)$	$A + B + C = A + (B + C)$
7	$(AB \dots Z)' = A' + B' + \dots + Z'$	$(A + B + \dots + Z)' = A'B'C' \dots Z'$
8	$AB + AC = A(B+C)$	$(A + B)(A + C) = A + BC$
9	$AB + AB' = A$	$(A + B)(A + B') = A$
10	$A + AB = A$	$A(A + B) = A$
11	$A + A'B = A + B$	$A(A' + B) = AB$
12	$CA + CA'B = CA + CB$	$(C+A)(C+A'+B) = (C+A)(C+B)$
13	$AB + A'C + BC = AB + A'C$	$(A+B)(A'+C)(B+C) = (A+B)(A'+C)$

Figura 4.2 Teoremas del Álgebra de Boole

4.3 Optimización de expresiones booleanas

Cuando se plantea un problema, en general la expresión booleana obtenida no necesariamente es la óptima, pues existen diversas formas de escribir una misma expresión algebraica. Lo ideal, es escribir la expresión algebraica de la manera más fácil, clara y sencilla posible para que pueda ser implementada con un menor costo utilizando compuertas lógicas.

La expresión que resulta del planteamiento del problema puede ser simplificada utilizando los teoremas y postulados del álgebra booleana o bien utilizando mapas de Karnaugh.

Cuando se optimiza una expresión booleana, se utilizan los teoremas y postulados del álgebra booleana para simplificar las expresiones lógicas o transformarlas en otras que son equivalentes. Una expresión simplificada se puede implementar con menos equipo y menos circuitos. Para simplificar u optimizar las expresiones booleanas, se aplican los teoremas, se comparan fragmentos de la expresión con los teoremas que permitan hacer más simple la expresión, este paso se realiza hasta que no sea posible simplificar más la expresión.

Ejemplo: Simplificar la expresión booleana

$$\begin{aligned}
 F &= A'B + (ABC)' + C(B'+A) + CB' + A \\
 F &= A'B + A'+B'+C' + C(B'+A) + CB' + A && \text{por 7} \\
 F &= A'B + A'+B'+C' + CB'+CA + CB' + A && \text{por 8} \\
 F &= A'B + A'+B'+C' + CB'+AC + CB' + A && \text{por 5} \\
 F &= A'B + A'+B'+C' + CB' + CB' + AC + A && \text{por 5b} \\
 F &= A'B + A'+B'+C' + CB' + AC + A && \text{por 3b} \\
 F &= A'B + A' + A + B' + C' + CB' + AC && \text{por 5b} \\
 F &= A'B + 1 + B' + C' + CB' + AC && \text{por 4b} \\
 F &= 1 && \text{por 1b}
 \end{aligned}$$

Ejemplo 2. Simplificar la función booleana

$$\begin{aligned}
 F &= A'B'C'D + A'B'CD + AB'C'D + AB'CD + AB'CD' \\
 F &= A'B'(C'D + CD) + AB'(C'D + CD) + AB'CD' && \text{por 8} \\
 F &= A'B'(D(C' + C)) + AB'(D(C' + C)) + AB'CD' && \text{por 8} \\
 F &= A'B'(D(1)) + AB'(D(1)) + AB'CD' && \text{por 4b} \\
 F &= A'B'(D) + AB'(D) + AB'CD' && \text{por 1} \\
 F &= B'D(A' + A) + AB'CD' && \text{por 8} \\
 F &= B'D(1) + AB'CD' && \text{por 4b} \\
 F &= B'D + AB'CD' && \text{por 2}
 \end{aligned}$$

4.4 Aplicación del álgebra booleana

El álgebra booleana tiene muchas aplicaciones, y una de sus principales funciones es la simplificación de circuitos lógicos. La construcción de los circuitos lógicos es muy cara, mientras mayor sea el número de compuertas lógicas utilizadas en la construcción de un circuito lógico, se aumenta su complejidad y es más propenso a errores. Los circuitos lógicos son la base para construir equipos de control, computadoras, calculadoras, un reloj, etc. Las operaciones que son implementadas en los sistemas digitales utilizan las operaciones definidas en el álgebra de Boole.

Otro uso también muy importante, es la criptografía cuya función principal es codificar y decodificar mensajes. Mediante operaciones booleanas es posible analizar y revisar mensajes de manera que sólo puedan ser decodificados por alguien que tenga la clave correcta.

En las empresas, muchos de sus procesos están automatizados, los sistemas telefónicos, los robots y cualquier dispositivo para automatización de los procesos utilizan el álgebra de Boole.

4.4.1 Minitérminos y maxitérminos

Cuando se quiere resolver un problema que involucra la implementación de sistemas digitales, se debe realizar un análisis detallado de lo que se quiere lograr. Esto incluye definir los sensores de entrada que se necesitan para obtener las señales, definir las condiciones que se deben cumplir para llevar a cabo ciertas acciones, definir las operaciones a realizar, así como los resultados que se esperan. Una vez que se plantea el funcionamiento del circuito lógico se debe definir una expresión matemática, que se denomina función booleana.

En álgebra booleana, se conoce como término canónico de una función lógica a todo producto o suma en la cual aparecen todas las variables en su forma directa o inversa, es decir, las variables pueden aparecer afirmadas o negadas. Todas las funciones lógicas se pueden expresar en forma canónica, ya sea como suma de minitérminos o como producto de maxitérminos. Esto permite un mejor análisis para la simplificación de dichas funciones.

Un minitérmino es una expresión lógica de n variables que contiene únicamente el operador de conjunción (AND). En esta expresión lógica aparecen todas las variables que forman parte de la función ya sea como variable normal o como su complemento. Por ejemplo, en una función de 3 variables A , B , y C , los minitérminos pueden ser: $\bar{A}BC$, $A\bar{B}\bar{C}$, ABC , etc.

Un maxitérmino es una expresión lógica de n variables donde el único operador que forma parte de la expresión lógica es el operador de disyunción (OR). En esta expresión lógica aparecen todas las variables que forman parte de la función ya sean como variable normal o como su complemento. Por ejemplo, en una función de 3 variables, los maxitérminos pueden ser $\bar{A}+B+C$, $A+B+\bar{C}$, $\bar{A}+\bar{B}+C$, $A+B+C$, etc.

A la suma de minitérminos también se le llama forma normal disyuntiva (pues los minitérminos están separados únicamente por el operador disyunción), al producto de maxitérminos también se le llama forma normal conjuntiva, pues los maxitérminos están separados únicamente por el operador de conjunción.

Un ejemplo de utilización de funciones booleanas en la industria

Suponga que una empresa dedicada a la purificación y embotellamiento de agua tiene un sistema automático para el control de calidad de las botellas de agua antes de que sean distribuidas. En el sistema automático, existen 3 diferentes sensores de entrada que verifican características importantes en la calidad del producto.

Cuando la botella de agua pasa por una banda transportadora, los sensores detectan los criterios de calidad requeridos y si no cumplen con estos criterios, las botellas son retiradas de la banda.

La tabla de verdad que determina las características mínimas de calidad en la botella se muestra en la figura 4.3.

Tabla de verdad que representa a la función:				Función lógica o expresión booleana que representa a la tabla de verdad:
A	B	C	F	
0	0	0	1	$F = A'B'C' + A'BC' + AB'C' + ABC$
0	0	1	0	
0	1	0	1	Esto significa que la botella con agua se extrae de la banda transportadora, en las siguientes situaciones: A=0, B=0, C=0 A=0, B=1, C=0 A=1, B=0, C=0 A=1, B=1, C=1
0	1	1	0	
1	0	0	1	
1	0	1	0	
1	1	0	0	
1	1	1	1	

Figura 4.3 Ejemplo de representación de una función booleana

4.3.2 Representación de expresiones booleanas con circuitos lógicos

Todos los programas y datos de una computadora se traducen en bits. A través de la historia, se han utilizado diferentes dispositivos en las computadoras digitales para almacenar bits, desde las tarjetas perforadas en los inicios de la era de la computación hasta las unidades de estado sólido (SSD, Solid State Drive), entre muchos otros dispositivos de almacenamiento.

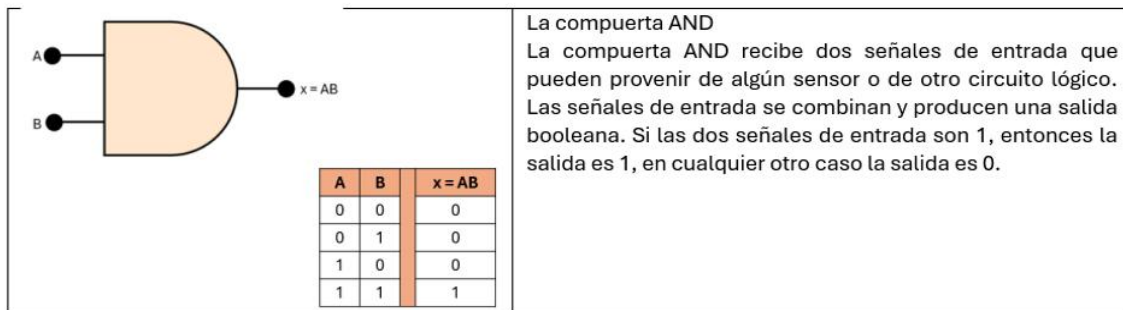
“Los circuitos electrónicos permiten que los dispositivos de almacenamiento se comuniquen entre sí. Un bit en una parte del circuito es transmitido a otra parte del circuito como un voltaje” (Johnsonbaugh, pp.470).

Se llaman compuertas a los circuitos combinatorios que tienen la capacidad de cambiar los niveles de voltaje (bits) y que son el fundamento para la operación de una computadora. Las compuertas fundamentales son AND, OR y NOT.

La compuerta AND

La compuerta AND recibe dos bits como entrada y produce una sola señal como salida. Si ambos bits de entrada son 1, entonces la señal de salida es 1, si cualquiera de los dos bits de entrada son 0, o ambas señales de entrada son 0, entonces la salida también es 0.

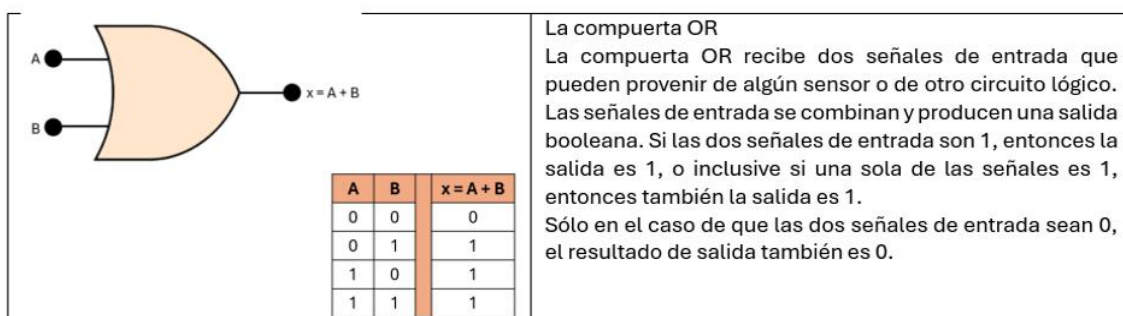
Por ejemplo, en un sistema de control de puertas automáticas, se puede activar la acción de abrir la puerta si y sólo si se presiona el botón de abrir la puerta y el sensor de proximidad indica que el área está despejada. Si el sensor de proximidad detecta objeto o señal que se encuentra cerca del sensor, entonces, aunque se presione el botón de abrir la puerta no se abre. Si cualquiera de las dos condiciones no se cumple, entonces la puerta no se abre.



La compuerta OR

En la compuerta OR se reciben dos señales como entrada y produce una sola señal como salida. En el caso de la compuerta OR, sólo una de las señales de entrada debe ser 1 para que la señal de salida sea 1. También la señal de salida es 1 cuando las dos señales de entrada son 1. Sólo en el caso de que las dos señales de entrada sean falsas, es decir 0, la señal de salida es 0.

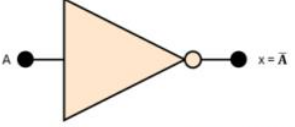
Un ejemplo de la utilización de la compuerta OR, es en un sistema de control de acceso a un edificio, donde el guardia de seguridad se encarga de revisar que la persona que desea ingresar tenga la credencial de empleado que acredite que la persona trabaja en el edificio o en su defecto la credencial para votar con fotografía. No es necesario que la persona que desea entrar al edificio tenga las dos credenciales, sólo con una credencial es suficiente para lograr el acceso al edificio, aunque si trae las dos credenciales de igual forma puede entrar. Sólo en el caso de que la persona no cuente con ninguna de las dos credenciales, a la persona se le niega el acceso al edificio.



La compuerta NOT

El circuito lógico que representa a la compuerta NOT, recibe una señal de entrada y entrega como resultado la señal opuesta a la que se recibió. Esta compuerta es muy útil para describir comportamientos que se presentan frecuentemente en la vida cotidiana.

La compuerta NOT o inversor



A	$x = \bar{A}$
0	1
1	0

La compuerta NOT

La compuerta NOT, también llamada inversor, porque invierte la señal de entrada y devuelve como salida la señal invertida. A diferencia de las compuertas AND y OR, que reciben dos señales como entrada, la compuerta NOT recibe una sola señal y entrega como salida la señal invertida.

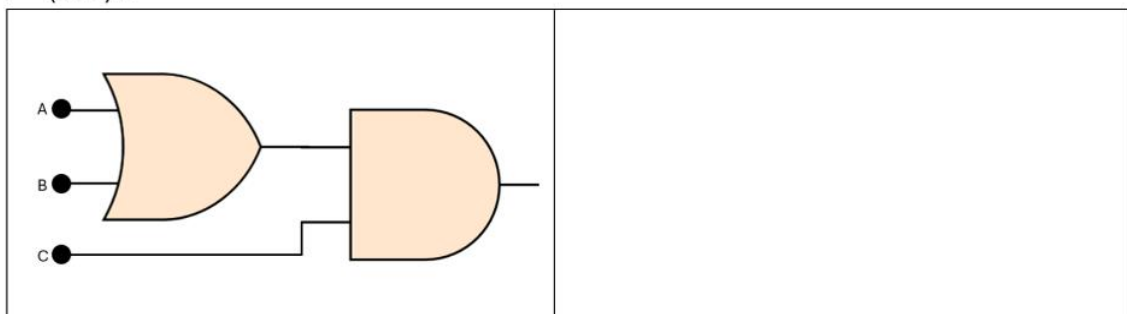
Un ejemplo que muestra el funcionamiento de la compuerta NOT es el apagador de escalera que tiene dos interruptores, si la luz está encendida y se presiona el interruptor, la luz se apaga, pero en caso de luz encendida y se presiona el interruptor, la luz se apaga.

Todos los circuitos lógicos, sin importar su complejidad o comportamiento se pueden describir por completo utilizando los 3 operadores básicos AND, OR, y NOT. La salida de una compuerta se puede utilizar como entrada en otra compuerta, creando circuitos lógicos muy poderosos y complejos. Las funciones lógicas se pueden representar con compuertas lógicas.

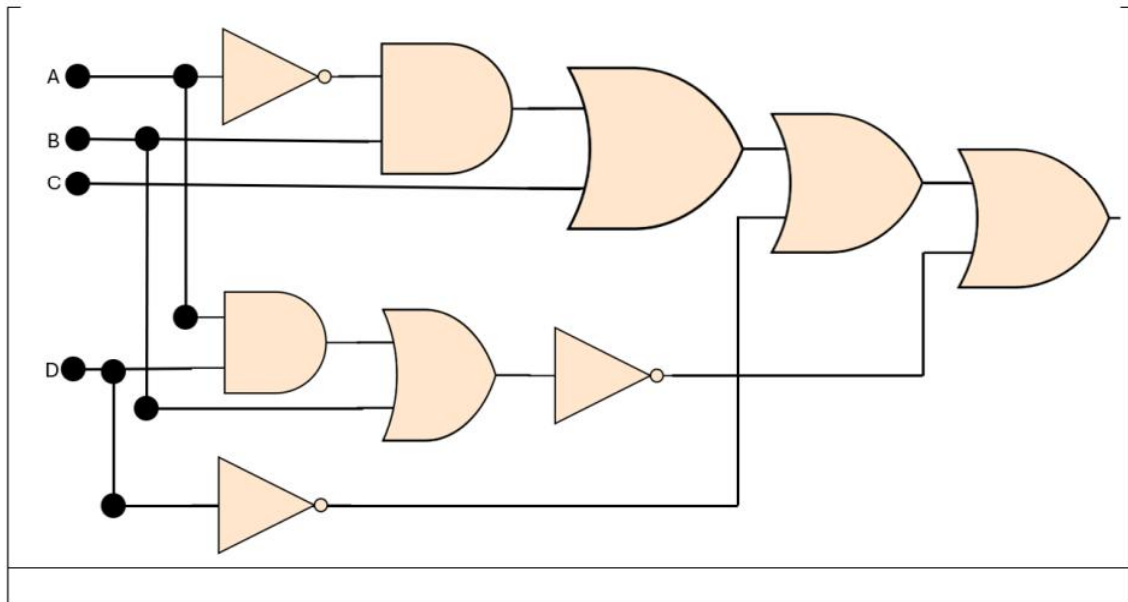
En ocasiones, el orden en que se llevan a cabo las operaciones booleanas puede ser confuso, por ejemplo, en la expresión $A+B.C$ se pudiera interpretar que se realiza primero la operación $A+B$ y posteriormente el resultado se multiplica por C como $(A+B)C$, pero no es así. Cuando las expresiones contienen operaciones AND y OR, la operación AND se realiza primero, y después la operación OR. Si se requiere cambiar el orden en que se realizan las operaciones, se deben utilizar paréntesis. Un inversor que se pone después de una señal, el resultado es una operación de negación, es decir, invierte la señal de entrada.

Por ejemplo, represente mediante compuertas lógicas las siguientes funciones booleanas.

$$F = (A+B)C$$



$$F = ((A'B+C)+D')+(AD+B)'$$



Tema 5

Teoría de grafos

La extrema complejidad de los sistemas actuales requiere la consideración y análisis de gran cantidad de características, algunas de ellas, irrelevantes para el problema en particular. Lograr la abstracción de las características esenciales al problema y plasmar su representación gráfica de manera que aporte una luz hacia la solución del problema es una de las tareas fundamentales de la Teoría de grafos. Aunque la primera publicación en la Teoría de Grafos apareció en el año 1736, fue hasta el año de 1920 cuando se consolidan sus fundamentos. Uno de los principales motivos del interés de la Teoría de Grafos es que permite representar una gran cantidad de problemas de diferentes tipos, siendo fundamental en la representación de topologías de redes y sistemas computacionales, incluyendo también campos como la química, ingeniería eléctrica y electrónica, investigación de operaciones, lingüística, entre muchos otros tópicos. Los grafos son estructuras discretas que se dibujan con puntos y líneas. Las líneas permiten relacionar o conectar puntos que representan elementos importantes al problema.

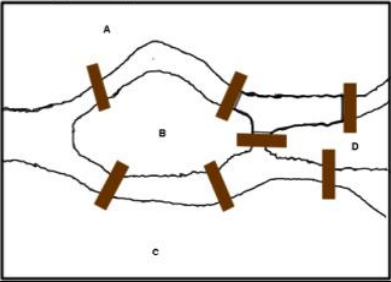
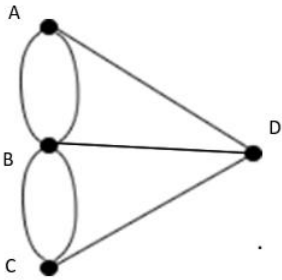
La historia de la teoría de grafos comenzó en Königsberg, quien fuera la capital de Prusia Oriental. A principios del siglo XVIII, en Königsberg existían siete puentes sobre el río Pregel, río que desemboca en el mar Báltico. Los habitantes de Königsberg salían de casa a pasear e intentaban cruzar cada puente una sola vez y regresar a casa. Después de muchos intentos, los habitantes pensaron que la tarea era imposible, por lo que decidieron preguntarle a Leonhard Euler quien fue un matemático y físico suizo, considerado el principal matemático del siglo XVIII y uno de los más grandes y prolíficos de todos los tiempos. Leonhard Euler se interesó en el problema y buscó una estrategia para plantear la solución. Para demostrar que es imposible salir de casa y cruzar los siete puentes una sola vez y regresar al lugar de partida, Euler utiliza lo que se considera los inicios de la teoría de grafos.

Competencia específica: Aplica los conceptos básicos de grafos para resolver problemas afines al área computacional, relacionados con el recorrido, búsqueda y ordenamiento de grafos.

Introducción

Cuando se utiliza la teoría de grafos en el planteamiento de un problema, se utilizan puntos y líneas para su representación. Para modelar una situación como un grafo, la primera acción que se debe realizar es decidir qué es lo que representa cada uno de los puntos, y posteriormente qué tipo de unión o conexión existe entre los puntos y trazar las líneas entre aquellos puntos que están relacionados. En el planteamiento del problema de los puentes de Königsberg se realizan los siguientes pasos:

1. Analizar el río y los segmentos de tierra separados por los puentes
2. Identificar las 4 fracciones de tierra que están separadas por el río y etiquetar cada una como A, B, C y D para distinguirlas.
3. Identificar la ubicación de cada puente especificando cuáles son las fracciones de tierra unidas por los puentes.
4. Representar cada fracción de tierra con un punto.
5. Relacionar los puntos (o las fracciones de tierra) que están conectados a través de un puente con una línea.

El problema de los puentes de Königsberg 	Se identifican las fracciones de tierra separadas por el río Pregel. Como se puede notar en la figura, no es posible cruzar directamente de A hacia C, pero se puede utilizar un puente para cruzar de A hacia B y posteriormente otro puente para cruzar de B hacia C. Las 4 fracciones de tierra están interconectadas. La conexión o la relación es posible a través de los puentes.
	Cuando se realiza una abstracción de información se identifican datos relevantes o esenciales del problema. En el problema de los puentes de Königsberg, se identifican las 4 fracciones de tierra y cuáles de ellas están relacionadas o conectadas por los puentes. Leonhard Euler representó el problema con una figura de puntos y líneas a la que llamó “grafo”. Los puntos representaban fracciones de tierra y las líneas representaban los puentes para cruzar el río.

Los grafos tienen muy diversas aplicaciones, en la industria, en medicina, química, en las relaciones interpersonales, etc. Por ejemplo, el comportamiento de las redes sociales puede ser modelado con grafos, donde los amigos de una persona se conectan a través de un enlace, posteriormente, se enlazan los amigos de los amigos y así sucesivamente.

Para la fabricación de gran cantidad de artículos en la industria se requiere perforar agujeros en placas de metal, para lo cual se utiliza un taladro controlado por computadora. El taladro debe ser posicionado en el lugar que debe perforar la placa. Los grafos se utilizan para revisar la secuencia de perforación en la placa y seleccionar la secuencia adecuada para optimizar el tiempo y esfuerzo requerido por el taladro.

5.1 Elementos, características y componentes de los grafos

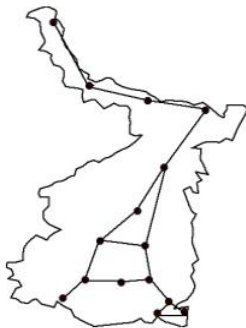
Un grafo se define como un conjunto de **vértices** (representados por puntos o nodos) los cuales están conectados por **aristas** (representadas por enlaces o líneas) donde cada línea representa una relación o conexión entre los vértices y los vértices representan alguna característica esencial al problema de que se trate.

Al número de líneas o aristas que salen de un vértice se le llama “orden del vértice” o más común “**valencia del vértice**”.

Después de analizar el problema de los puentes de Königsberg, Euler llegó a la conclusión de que es imposible obtener un itinerario que salga de un vértice y regrese a él pasando por todas las aristas solamente una vez. Según Euler, si el vértice donde se inicia y termina el recorrido es el mismo, entonces dicho vértice debe tener valencia par, esto es debido a que por un puente se sale y por otro diferente se debe regresar (para no pasar dos veces por el mismo puente), lo mismo sucede con los demás vértices.

Euler estableció que “en un grafo, únicamente se puede establecer un ciclo que pase por todas las aristas sólo una vez si todos los vértices tienen valencia par” (Jiménez, p. 287).

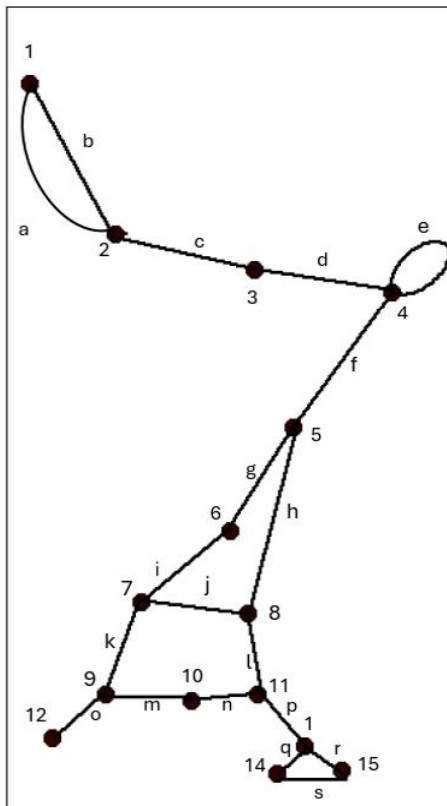
Ejemplo de grafo

Grafo de principales carreteras de Tamaulipas 	En este ejemplo del grafo se muestran las carreteras que conectan las ciudades más importantes en el estado de Tamaulipas. Cómo se puede notar, en algunos casos existen dos carreteras distintas para viajar de un lugar a otro.
--	---

Un grafo es una representación de red o malla por medio de la cual se puede expresar en forma visual y la relación entre los elementos que pueden ser de distinto tipo. Por ejemplo, el organigrama de una empresa donde se muestra la relación entre los diferentes departamentos y los puestos existentes, enfatizando líneas de autoridad-responsabilidad y dependencia entre los mismos.

Un diagrama eléctrico donde se visualice la posición de los contactos y apagadores y las líneas de conexión entre ellos, una red de teléfono que muestre la posición de los postes que distribuyen la señal, la red de carreteras de México donde se pueden ver las ciudades y localidades que son conectadas por las carreteras, la red de computadoras que muestra un conjunto de computadoras conectadas por medio de cables, etc. En los grafos, los vértices pueden ser postes, transformadores, teléfonos, ciudades, válvulas, puestos en una empresa, etc. Las aristas que representan la conexión entre los vértices pueden ser tubos, cables, carreteras, líneas de mando, etc. Por medio de la teoría de grafos se pueden aprovechar mejor los recursos eliminando conexiones redundantes y reduciendo costos y distancias.

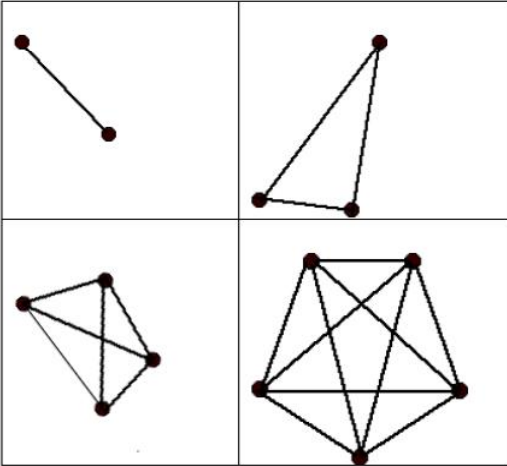
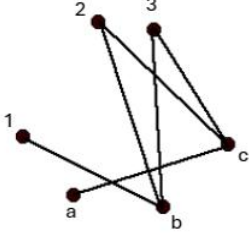
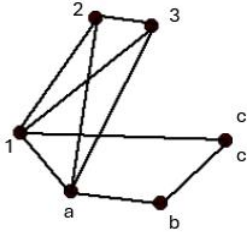
En el área de la computación, los grafos se utilizan para mostrar las relaciones entre los archivos (bases de datos), relaciones entre registros (estructuras de los datos), relación entre computadoras (redes de computadoras). Gracias a la sencillez de los elementos con que se representa un grafo, es más claro ver la relación entre los elementos de un grafo.

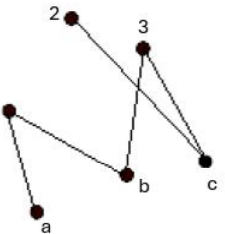
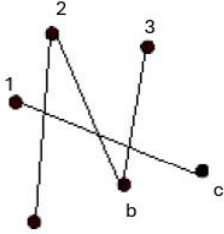
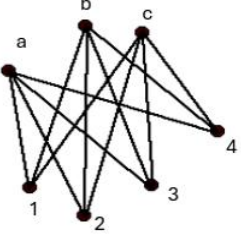


Elementos de un grafo:

- **Vértices (nodos).** Se indican por medio de un pequeño círculo y se les asigna un identificador como un número o letra. $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$
- **Aristas (ramas o lados).** Son las líneas que unen un vértice con otro, también se etiquetan con una letra o un número (diferente a la asignación del vértice para evitar confusiones). $A = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s\}$
- **Lados paralelos.** Son aquellas aristas que tienen relación con un mismo par de vértices $P = \{a, b\}$
- **Lazo.** Es aquella arista que sale de un vértice y regresa al mismo vértice. $L = \{e\}$
- **Valencia de un vértice.** Es el número de lados que salen o entran a un vértice.
- Valencia (1) = 2
- Valencia (4) = 3 ** El lazo se cuenta 1 sola vez
- Valencia (7) = 3
- Valencia (10) = 1

5.1.1 Tipos de grafos

	Grafo simple Es aquel que no tiene lazos ni lados paralelos
	Grafo completo de n vértices (K_n) Es un grafo en donde cada uno de sus vértices está relacionado con todos los demás vértices, sin lazos ni lados paralelos. Se indica como K_n, donde n es el número de vértices del grafo. La valencia en cada uno de los vértices de los grafos completos es $(n-1)$, y Número de lados = $\frac{n(n-1)}{2}$ Por ejemplo, en el caso del grafo de 4 vértices (K_4), cada vértice tiene valencia $(4-1) = 3$ Núm de lados = $\frac{4(4-1)}{2} = 6$
$A = \{1, 2, 3\}$ $B = \{a, b, c\}$ $G = \{(1,b), (1,c), (2,c), (3,b), (3,c)\}$ $G' = \{(1,a), (1,c), (2,a), (2,b), (3,b), (3,c)\}$  Grafo G  Grafo G'	Complemento de un grafo (G') El complemento de un grafo (G'), es el grafo que le falta al grafo (G) de forma que entre ambos formen un grafo completo de n vértices. Este grafo no tiene lazos ni lados paralelos. El grafo completo que se forma con la combinación de todos los pares ordenados de los conjuntos A y B, está dado por los pares ordenados en C. $C = \{(1,a), (1,b), (1,c), (2,a), (2,b), (2,c), (2,a), (3,b), (3,c)\}$

Grafo bipartido	
$A = \{1, 2, 3\}$ $B = \{a, b, c\}$ $ReA = \{(1,a), (1,b), (1,c), (2,c), (3,b), (3,c)\}$ $ReB = \{(1,c), (2,a), (2,b), (3,b), (3,c)\}$  Grafo bipartido 1  Grafo bipartido 2	Es el grafo que está compuesto por dos conjuntos de vértices $A = \{a_1, a_2, a_3 \dots a_n\}$ y $B = \{b_1, b_2, b_3 \dots b_n\}$ Donde los elementos del conjunto A se relacionan con los elementos del conjunto B, pero entre los vértices de un mismo conjunto, no existe arista que los una. En el ejemplo se puede notar que entre los vértices etiquetados con números no se relacionan, ni tampoco entre los vértices etiquetados con letra.
Grafo bipartido completo ($k_{n,m}$)	
 Grafo bipartido completo $K_{3,4}$	Es el grafo que está compuesto por dos conjuntos de vértices, uno de ellos $A = \{a_1, a_2, a_3 \dots a_n\}$ y el otro $B = \{b_1, b_2, b_3 \dots b_n\}$ En el grafo bipartido completo cada vértice de A está unido con todos los vértices de B, pero entre los vértices de un mismo conjunto, no existe arista que los una.

5.2 Representación de los grafos

La utilización de matrices para representar grafos permite gestionar su información, así como determinar algunas propiedades de los grafos que de otra manera serían más difíciles de obtener. Un grafo puede ser representado en forma matemática o en forma computacional, ambas formas se complementan.

5.2.1 Representación matemática de un grafo

La representación matemática más difundida para la gestión de grafos es la matriz de adyacencia, aunque existen otras opciones que dependiendo de las características del grafo también se utilizan ampliamente.

La matriz de adyacencia (M_a) es una matriz cuadrada en la cual los vértices del grafo se indican como filas y columnas: el orden de los vértices es el mismo que guardan las filas y las columnas de la matriz.

Se coloca un 1 como elemento de la matriz cuando existe una relación entre dos vértices, o bien un 0, cuando no existe relación alguna.

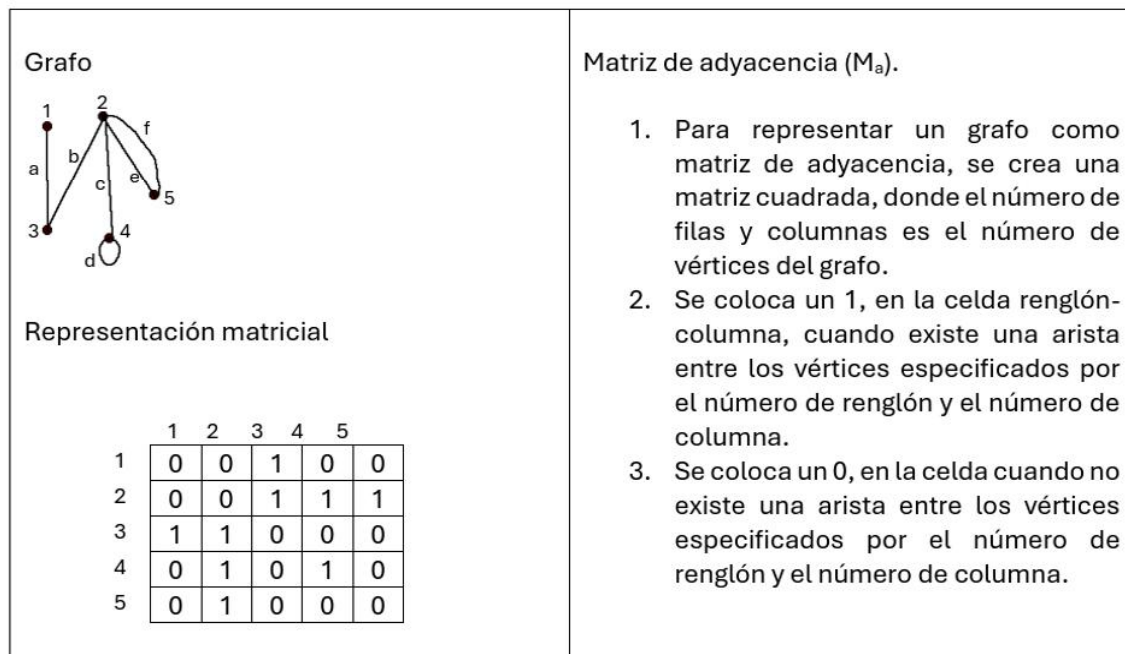


Figura 5.1 Representación de un grafo utilizando matriz de adyacencia (M_a)

Es importante observar que la matriz de adyacencia no permite representar lados paralelos, a menos que se especifique el número de lados en la casilla correspondiente, pero la matriz ya no sería binaria. Por otro lado, la matriz contiene información repetida, porque la celda renglón columna contiene la misma información que columna, renglón.

En conclusión, se tiene que la matriz de adyacencia es una buena herramienta para realizar operaciones con relaciones, pero no permite registrar toda la información del grafo.

Matriz de incidencia (M_i)

En esta matriz se colocan los vértices del grafo como filas y las aristas como columnas.

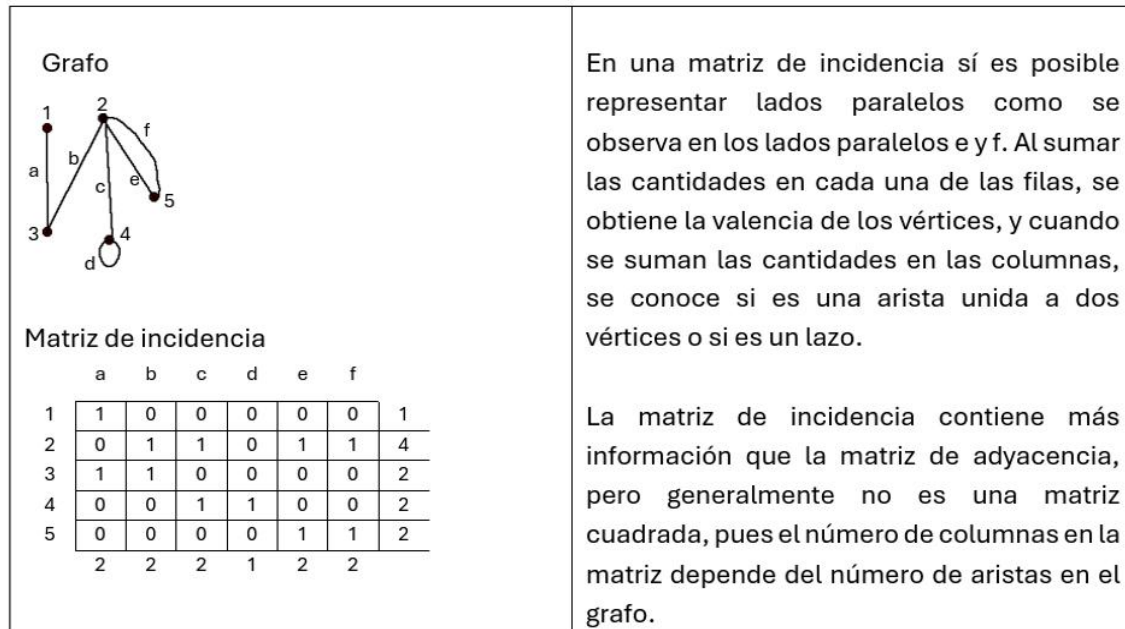


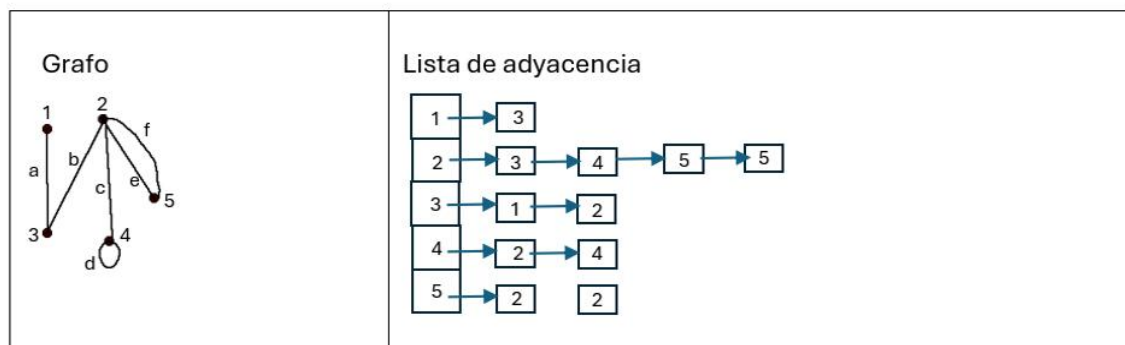
Figura 5.2 Representación de un grafo utilizando matriz de incidencia (M_i)

5.2.2 Representación computacional de un grafo

La utilización de matrices para representar grafos tiene la ventaja de su implementación computacional, lo que facilita el manejo de matrices, ya que se pueden tratar como arreglos o listas doblemente enlazadas.

Otra alternativa para la representación de un grafo es la matriz de adyacencia, que consiste en un arreglo de N posiciones (una por cada vértice), desde el cual se apunta a una lista de adyacencia. Una lista de adyacencia para un vértice i , consiste en una lista que contiene los identificadores de los vértices que son adyacentes al vértice i .

Esta representación es útil cuando el grafo tiene pocas aristas o cuando se requiere conocer con frecuencia si una determinada arista existe en el grafo.



En el caso de la lista de adyacencia se tiene información redundante. Al representar la arista (1,3), en la lista de adyacencia correspondiente al vértice 1, hay un nodo con el valor de 3. A su vez, en la lista de adyacencia correspondiente al vértice 3, hay un nodo con el valor de 1. Cuando se tienen lados paralelos, se pueden especificar repitiendo el número de nodo correspondiente.

5.3 Algoritmos de recorrido y búsqueda

El recorrido de un grafo consiste en realizar una trayectoria por los nodos del grafo. Para recorrer el grafo se deben visitar cada uno de los nodos alcanzables a partir de un nodo inicial. Existen dos formas de recorrer un grafo: recorrido a lo ancho y recorrido en profundidad. El principio del recorrido de un grafo a lo ancho es visitar primero a todos los nodos del mismo nivel, es decir a partir del nodo inicial, recorrer todos los nodos que son adyacentes a él; mientras que, en la búsqueda en profundidad, se visita el primer nodo adyacente y se continua con la visita de sus nodos adyacentes hasta llegar al nodo más lejano del nodo inicial.

Para el recorrido de los nodos de un grafo, se requiere una revisión sistemática de los vértices y aristas de forma que se asegure que se han recorrido todos los nodos. La información que se obtiene de un grafo y la forma en que se llevan a cabo los recorridos de los nodos, depende de la estructura de datos en donde se almacena el grafo, se debe de mantener en memoria la estructura de adyacencia del grafo.

5.3.1 El camino más corto

Se le llama grafo ponderado al grafo cuyas aristas tienen un valor denominado peso o ponderación, este peso representa la distancia o el costo de pasar de un nodo a otro a través de una arista. El objetivo de que todas las aristas en el grafo ponderado tengan un peso es permitir calcular el costo de recorrer las aristas para llegar desde un nodo inicial a un nodo destino. Existen diferentes caminos que se pueden seguir para recorrer los nodos, dependiendo del número de aristas en cada vértice. Determinar el camino más corto es un problema típico de la teoría grafos y consiste en encontrar la ruta de menor costo para llegar del nodo X al nodo W, tomando en cuenta las ponderaciones de las aristas.

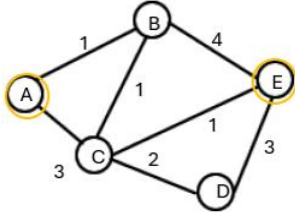
El método más común utilizado para encontrar el menor costo o el camino más corto entre dos nodos de un grafo es el algoritmo de Dijkstra. Este algoritmo es iterativo y permite calcular la distancia más corta al recorrer dos nodos cualesquiera. El procedimiento consiste en sumar los pesos de cada arista recorrida desde el nodo X al nodo W.

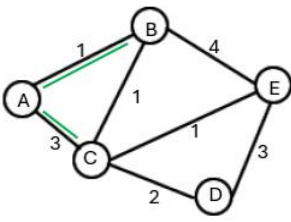
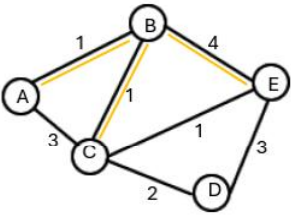
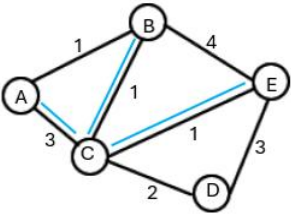
Algoritmo de Dijkstra:

- a. Seleccionar el nodo origen y nodo destino de la ruta (nodo X al nodo W)
- b. Diseñar una matriz que tenga como columnas: el número de iteración, una columna por cada uno de los nodos del grafo, nodo actual, y trayectoria mínima.
- c. Iniciar el algoritmo con la iteración 0. Se especifica la distancia recorrida al nodo inicial que es 0, puesto que aún no se ha recorrido ninguna arista, y a los demás nodos infinito (∞), indicando que no se conoce la distancia.
- d. Colocar en la columna Nodo actual el vértice que tenga el camino más corto, en este caso es el nodo origen por ser la iteración 0. La columna nodo actual contiene el nodo que será utilizado como nodo destino en la siguiente iteración.
- e. Colocar en la columna Trayectoria mínima, el conjunto de nodos recorridos y que forman parte de la ruta mínima. Esta columna se utiliza para mantener un registro de los nodos que forman parte de la ruta mínima, los cuales no se pueden repetir.
- f. Continuar con la siguiente iteración.
- g. Registrar en cada una de las columnas de los nodos del grafo, la distancia más corta que se obtiene al sumar la distancia registrada en el nodo actual y la distancia a los vértices adyacentes a él.
- h. Revisar las distancias de los nodos en la iteración actual, y seleccionar el nodo con la distancia más corta y que aún no forme parte de la trayectoria mínima.
- i. Repetir los pasos de f, g, h hasta que todos los nodos del grafo formen parte de la trayectoria mínima, o hasta los nodos desde el nodo origen hasta el nodo destino.

Ejemplo 1. Calcule el camino más corto para llegar del nodo A al nodo E del siguiente grafo.

Resolución del problema utilizando el Algoritmo de Dijkstra

 Diagrama de un grafo con 5 nodos (A, B, C, D, E) y sus pesos de aristas. El camino más corto de A a E es A-B-E con un costo total de 5.	1. Seleccionar el nodo A como nodo origen y el nodo E como nodo destino. Iniciar con la iteración 0. En esta primera iteración se especifica el costo inicial de A = 0 (el costo para llegar de A hacia A es 0) e infinito hacia los demás nodos (lo que significa que no se conoce el costo para llegar a los demás nodos). Especificar como Nodo actual = A, que es donde inicia el recorrido y Trayectoria mínima también es A, lo que indica que es el único nodo que se ha recorrido.
--	--

	2. Registrar la segunda iteración 1A, calculando las distancias de A hacia sus nodos adyacentes, los nodos adyacentes de A son B y C, $AB = 1$, $AC = 3$, ∞ en las demás posiciones. Seleccionar el camino más corto (AB), seleccionar como nodo actual a B, trayectoria mínima ahora es AB porque se selecciona la ruta AB como camino más corto.
	3. Registrar la tercera iteración a partir del nodo B y calcular la distancia de B hacia todos sus nodos adyacentes. Seleccionar el camino más corto (ABC). Seleccionar como nodo actual a C, y la trayectoria mínima hasta el momento recorrida es ABC (pueden existir dos caminos con el mismo costo, elegir cualquiera).
	4. En la siguiente iteración se calculan las distancias de C hacia todos sus nodos adyacentes, considerando siempre el costo menor y aquellos nodos que no han sido añadidos a la trayectoria mínima. Seleccionar como nodo actual a E y como se ha llegado al nodo destino terminar.

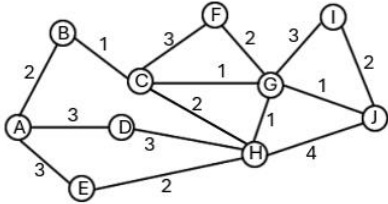
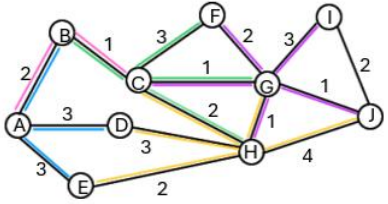
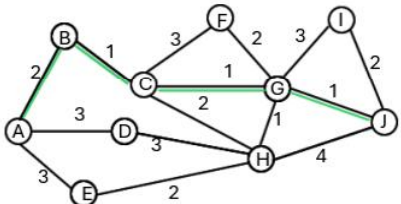
Matriz de cálculo del camino más corto

Iteración	A	B	C	D	E	Nodo actual	Trayectoria mínima
0	0	∞	∞	∞	∞	A	A
1A	0	1	3	∞	∞	B	AB
2B	0	1	2	∞	5	C	ABC
3C	0	1	2	4	3	E	ABCE

El Algoritmo de Dijkstra garantiza la distancia más corta para cualesquiera dos vértices del grafo. En cada iteración del procedimiento, se pueden encontrar caminos más cortos, porque si se encuentra un costo menor, se registra, siempre se anotan pesos menores cuando se anotan distancias más cortas, lo que garantiza que en el recuadro de cada una de las columnas de la tabla de la matriz del Cálculo del camino más corto se encuentre la ruta óptima.

Ejemplo 2. Calcule el camino más corto para llegar del nodo A al nodo J del siguiente grafo.

Resolución del problema utilizando el Algoritmo de Dijkstra

	1. Seleccionar los nodos A como inicial y el nodo G como nodo final. 2. Se inicia la tabla del cálculo del camino más corto, indicando el costo 0 para el nodo A, e infinito (∞) para los demás nodos. 3. Se analizan las 3 aristas adyacentes al nodo A, determinando que la distancia al nodo B es la menor (costo de B=2). Agregar B a la trayectoria
	4. Para una mejor comprensión, se dibujan líneas de color en las aristas adyacentes al nodo que se está analizando. 5. En el nodo B se tienen dos aristas adyacentes, se elige la de menor costo y además que no forme parte de la trayectoria mínima (en este caso C = 1). Se agrega C a la trayectoria. 6. El nodo C tiene 4 aristas adyacentes, se selecciona G=1, por tener menor costo y no forma parte de la trayectoria mínima. Se añade G a la trayectoria. 7. Se procede con los pasos anteriores hasta analizar e incluir en la trayectoria todos los nodos del grafo.
	El camino más corto entre los nodos A y J tiene una distancia de 5, como se observa en la columna del nodo J. En los 3 nodos adyacentes a J se puede observar que el nodo G es el que tiene el menor costo de los 3. Y analizando los nodos adyacentes a G se observa que el nodo C es el nodo de menor costo.

Se genera la matriz de Cálculo del camino más corto tomando en cuenta las siguientes consideraciones:

- Es importante señalar que sólo se pueden calcular los costos de los nodos adyacentes, o nodos cuyo camino más corto haya sido calculado previamente.
- Si no es un nodo adyacente, pero el camino más corto ya fue calculado, entonces se respeta el valor previamente calculado, por lo que la casilla en la matriz no se debe modificar.

- Si al calcular el costo de la ruta del nodo actual se obtiene un costo mayor, el valor no se debe sustituir.
- Si al realizar el cálculo del costo menor de la ruta se obtiene un costo aún menor que el valor actual, este valor se debe sustituir en la matriz.

Matriz de cálculo del camino más corto

Iteración	A	B	C	D	E	F	G	H	I	J	Nodo actual	Trayectoria
0	0	∞	∞	∞	∞						A	A
1A	0	2	∞	3	3	∞	∞	∞	∞	∞	B	AB
2B	0	2	3	∞	∞	∞	∞	∞	∞	∞	C	ABC
3C	0	2	3	∞	∞	6	4	5	∞	∞	G	ABCG
4G	0	2	3	∞	∞	6	4	5	7	5	H	ABCGH
5H	0	2	3	6	5	6	4	5	7	5	E	ABCGHE
6E	0	2	3	6	3	6	4	5	7	5	J	ABCGHEJ
7J	0	2	3	6	3	6	4	5	7	5	D	ABCGHED
8I	0	2	3	6	3	6	4	5	7	5	*	ABCGHEDI

5.3.2 Búsqueda a lo ancho

Se denomina búsqueda a lo ancho o BFS (Breadth-first search) a la revisión sistemática de los nodos de un grafo. La revisión de un nodo, también se conoce como “Visitar un nodo”, proceso por medio del cual se obtiene información relevante del nodo.

El procedimiento de búsqueda a lo ancho comienza en un vértice inicial, a partir del cual se visitan todos sus nodos adyacentes en orden de izquierda a derecha. Si la información que se busca no se encuentra en alguno de los nodos de ese nivel, entonces se procede a buscar en el siguiente nivel siguiendo la trayectoria por las aristas del nodo también en orden de izquierda a derecha, y así sucesivamente hasta encontrar la información o haber recorrido todos los nodos del grafo.

5.3.3 Búsqueda en profundidad

Se denomina búsqueda en profundidad o DFS (Depth-first search) a la revisión sistemática de los nodos de un grafo. En este procedimiento se comienza en un vértice inicial, a partir del cual se busca en el nodo adyacente que se encuentra más hacia la izquierda, y si este nodo tiene nodos adyacentes (no considerar el vértice inicial), entonces se revisa el nodo adyacente más hacia la izquierda y así sucesivamente hasta llegar al nodo más alejado del vértice inicial. Una vez que se visita el nodo más alejado del vértice inicial, se continua con el siguiente nodo a la derecha y así sucesivamente.

En la búsqueda en profundidad se viaja por todos los nodos adyacentes del nodo más hacia la izquierda del vértice inicial, lo cual significa que se recorrió toda la rama de los nodos adyacentes al nodo de la izquierda, posteriormente se regresa al punto de partida para continuar con el siguiente nodo adyacente al vértice inicial y continuar con este proceso mientras que no hayan sido visitados todos los nodos adyacentes del vértice inicial. El objetivo de este proceso es visitar el nodo más cercano al hijo izquierdo y todos sus descendientes, dando preferencia siempre al descendiente de la izquierda. Este proceso se repite hasta encontrar la información o bien recorrer todo el grafo.

Tema 6

Árboles y redes

Conforme el volumen de información aumenta inmensurablemente en los últimos tiempos, las estrategias de búsqueda de información y almacenamiento de información se mejoran. En las ciencias de la computación se utilizan estructuras de datos para organizar información de manera que pueda ser extraída y almacenada en forma eficiente. En las instituciones bancarias, por ejemplo, la búsqueda de los datos de un cliente para tener acceso a su cuenta a través de los servicios de los cajeros automáticos en red debe obtenerse en milésimas de segundo para que el resultado sea eficiente, sin importar la gran cantidad de datos almacenados. El tiempo de acceso a la información es un factor crítico en el funcionamiento de una empresa.

Una estructura de datos ampliamente utilizada para manejar grandes volúmenes de información de manera eficiente y eficaz es el árbol. Uno de los objetivos básicos de la utilización de árboles en las ciencias computacionales es entregar resultados en forma más rápida y ordenada. Las grandes bases de datos y la indización de grandes volúmenes de información son indispensables en el funcionamiento de las empresas modernas. Los árboles se utilizan para el almacenamiento y manipulación de datos.

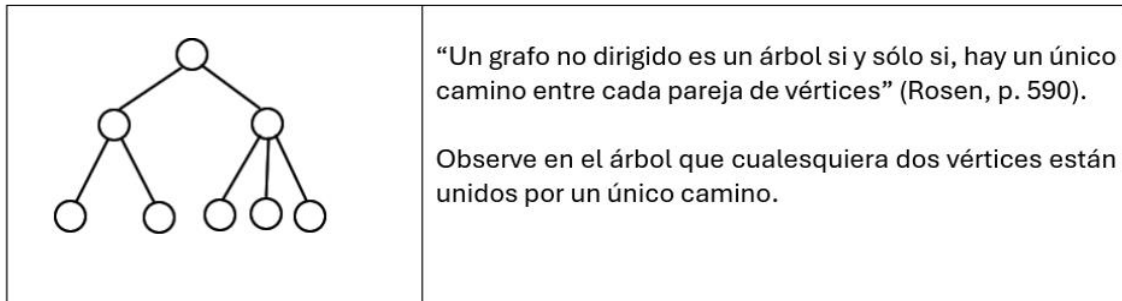
Un árbol es una colección de nodos que están conectados a través de referencias o enlaces. Cada uno de los nodos contiene información valiosa a la cual se puede tener acceso recorriendo trayectorias a través de los enlaces. Un árbol permite mantener la información ordenada y organizada de tal manera que pueda ser recuperada y almacenada de manera eficiente.

Competencia específica: Aplica la organización y relación entre los datos mediante procesos de ordenamiento, para resolver problemas de programación matemática donde se hace uso de las redes.

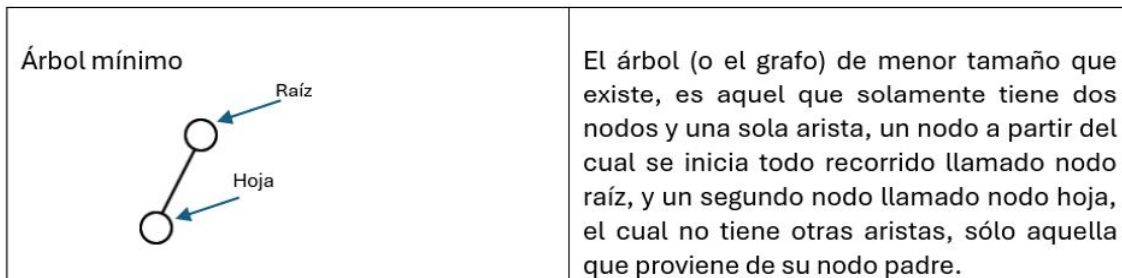
6.1 Árboles

Los árboles son estructuras jerárquicas que permiten una organización ordenada de la información, de manera que cuando se requiera, se pueda encontrar la información en forma rápida y precisa. Los árboles en computación se utilizan en los procesos de clasificación de información, administración de bases de datos, codificación de información, estructuras de datos, reconocimiento de patrones, etc. Con base en la teoría de grafos, se tiene la siguiente definición:

“Un árbol es un grafo no dirigido, conexo, que no tiene ciclos, lazos, ni lados paralelos”



Debido a que un árbol no puede contener ciclos, por lo tanto, se puede decir que un árbol es un grafo sin ciclos, y como un árbol tampoco puede contener lazos o lados paralelos, entonces un árbol es un grafo simple. El árbol más simple que existe es aquel formado por una raíz y una hoja conectados por una arista. Existe una única trayectoria que unen cada par de vértices en el árbol.



6.1.1 Componentes y propiedades

Un árbol, al igual que un grafo, está formado por un conjunto de vértices y aristas. Los vértices de un árbol reciben el nombre de nodos y las arista o lados de un árbol reciben el nombre de ramas. Un árbol está organizado por niveles, y existe un nodo único y el más alto en la jerarquía denominado raíz. La raíz es el nodo a partir del cual se inician todos los recorridos del árbol.

La raíz se encuentra en el nivel 0 del árbol. Los vértices o nodos que son adyacentes a la raíz (y que en la representación gráfica del árbol se dibujan debajo de la raíz) se encuentran en el nivel 1. Los nodos adyacentes al nivel 1, se encuentran en el nivel 2, y así sucesivamente. La altura de un árbol es el valor de su nivel más bajo.

Con excepción de la raíz, todo nodo está vinculado a un nodo de mayor nivel llamado nodo padre, además el nodo puede estar relacionado con uno o más nodos en un nivel más bajo, estos nodos son llamados nodos hijo del nodo. Ningún hijo puede tener más de un padre.

Por ejemplo, en la figura 1, El nodo padre de B, es el nodo A, y los hijos de B son los nodos D y E. El nodo raíz es el nodo A y los nodos hoja son aquellos nodos que no tienen hijos, en la figura 1, los nodos hoja son D, E, F, G y H.

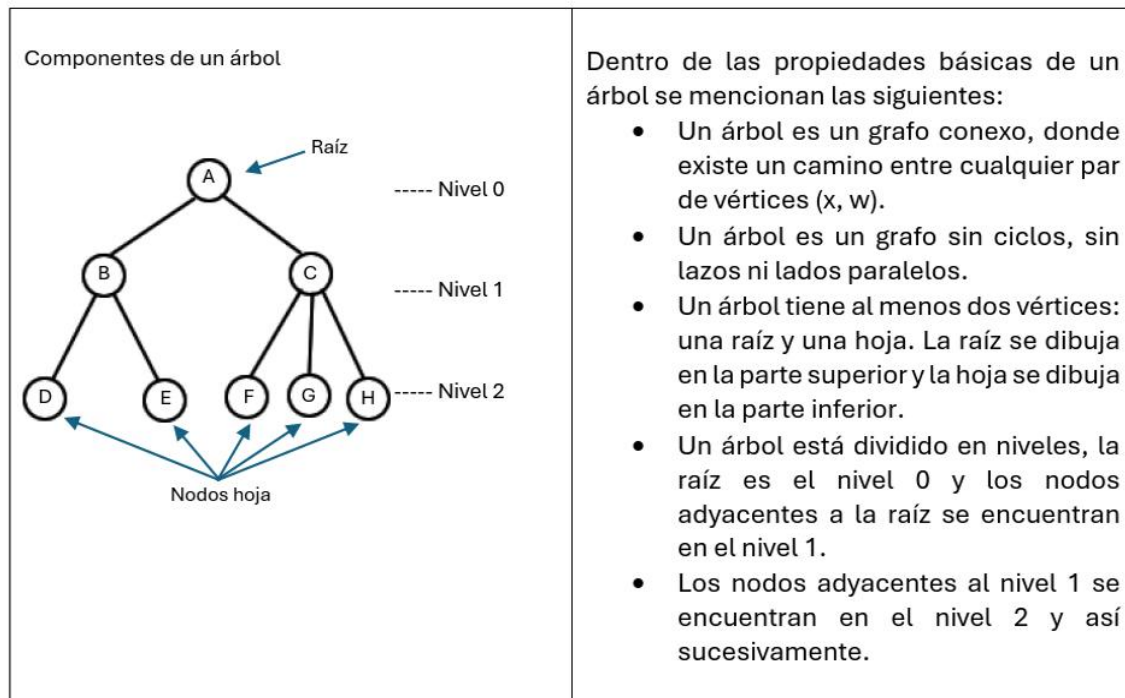


Figura 6.1. Representación gráfica de un árbol

6.1 2 Clasificación por altura y número de nodos

Todos los elementos colocados debajo de un nodo, independientemente de su nivel, se denominan descendientes.

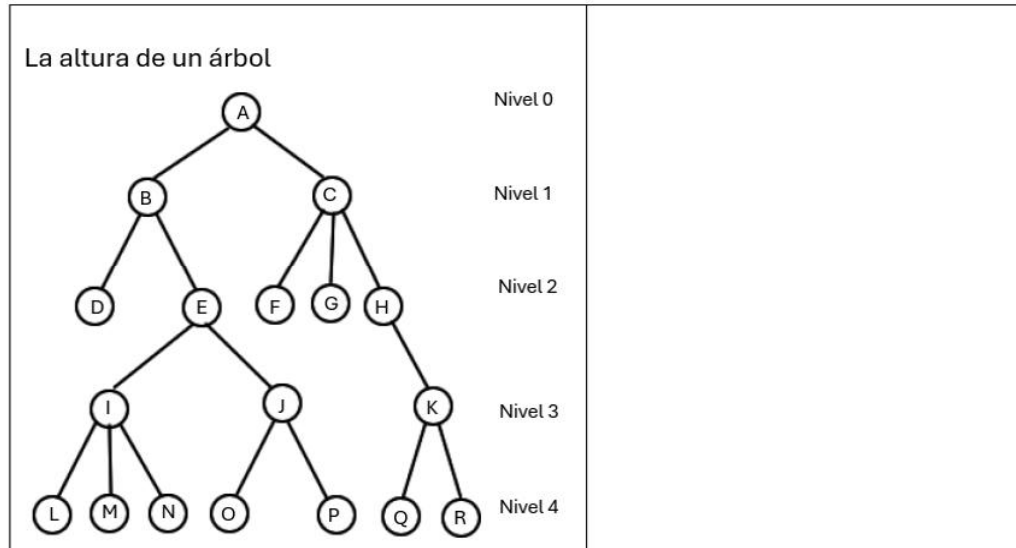


Figura 6.2. La altura de un árbol

En el árbol de la figura 6.2, el nodo H tiene como descendientes a los nodos {K, Q, R}. Los descendientes de H se pueden separar del árbol y formar un subárbol con los nodos {K, Q, R}, un árbol está integrado por varios subárboles.

Los nodos que se encuentran en una misma línea de descendencia, es decir del nodo padre hacia la raíz, se denominan antecesores. Por ejemplo, los antecesores del nodo J son {E, B, A}. Un árbol puede estar integrado por varios subárboles.

Se les llama vértices internos a todos aquellos nodos que no son nodos hoja. En la figura 2 los nodos {A, B, C, E, I, J, H, K} son vértices internos, mientras que los nodos {D, F, G, L, M, N, O, P, Q, R} son nodos hoja, pues no tienen hijos.

Los árboles se pueden clasificar de acuerdo con su número de nodos y en función de su altura.

Clasificación de los árboles por número de nodos

De acuerdo con la cantidad de máxima de hijos que tenga cada nodo padre en un árbol, los árboles se clasifican en:

- Árboles binarios. Los árboles binarios son aquellos en el que cada nodo padre tiene uno o dos hijos como máximo).
- Árboles terciarios. Cada nodo padre tiene máximo 3 hijos.
- Árboles cuaternarios. Cada nodo tiene como máximo 4 hijos.
- Árboles N-arios. Son árboles cuyos nodos pueden tener cualquier cantidad de hijos.

Los árboles binarios son especialmente importantes en el área de la computación por su naturaleza de tener 2 hijos, cada uno de los hijos puede tomar un valor binario razón por la cual son muy útiles en aplicaciones de sistemas digitales.

Árboles binarios

Árbol binario completo

Un árbol binario completo, es aquel en el que cada nodo tiene dos ramas o ninguna.

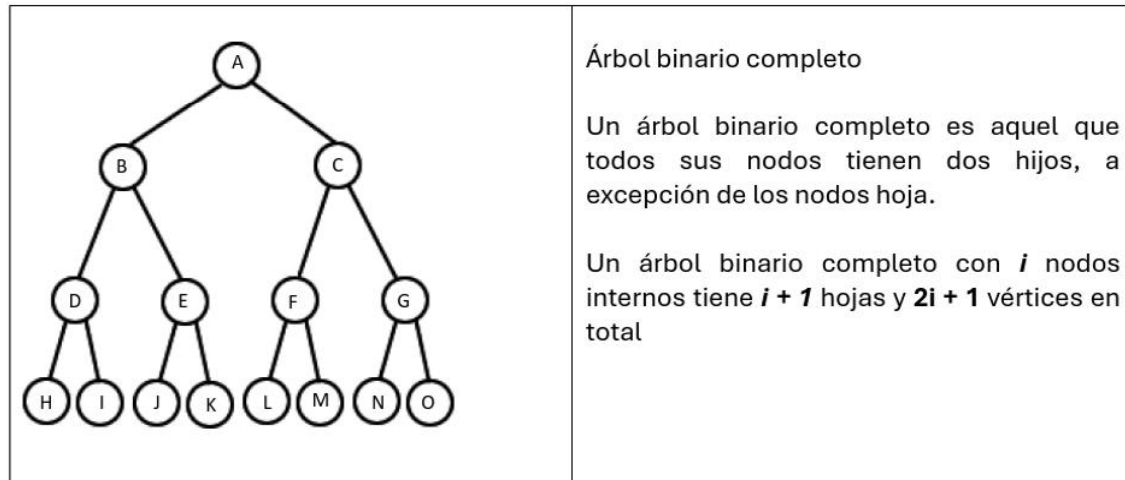


Figura 6.3. Árbol binario completo

En el caso de la figura 6.3, la cantidad de nodos internos son $i=7$

Por lo tanto

Hojas = $i + 1 = 7 + 1 = 8$ Nodos hoja

Total de vértices = $2i + 1 = 2 \times 7 + 1 = 15$ vértices en total

Clasificación de los árboles por su altura

La altura de un árbol se determina por la cantidad de niveles que posee, es el valor de su nivel más bajo, es decir, el número de niveles que conforman el árbol. El mayor nivel de un árbol determina la distancia máxima que se recorre desde la raíz del árbol hasta su hoja más alejada. La raíz tiene un nivel 0, los vértices que son adyacentes a la raíz tienen un nivel 1, los vértices o nodos que son adyacentes a los nodos de nivel 1, tienen un nivel 2, y así sucesivamente. De acuerdo con su altura, los árboles pueden ser:

Balanceados. Cuando la diferencia de altura entre sus ramas es 1, o menor.

Desbalanceados. Cuando la diferencia de altura entre sus ramas es mayor que 1.

Bosques

Un bosque es un conjunto de árboles. De un árbol se pueden obtener varios subárboles, los mismos que conforman el bosque. A su vez un árbol puede considerarse un árbol conectado, sólo se debe tener en cuenta que el árbol más pequeño está integrado por cuando menos dos nodos conectados por una arista.

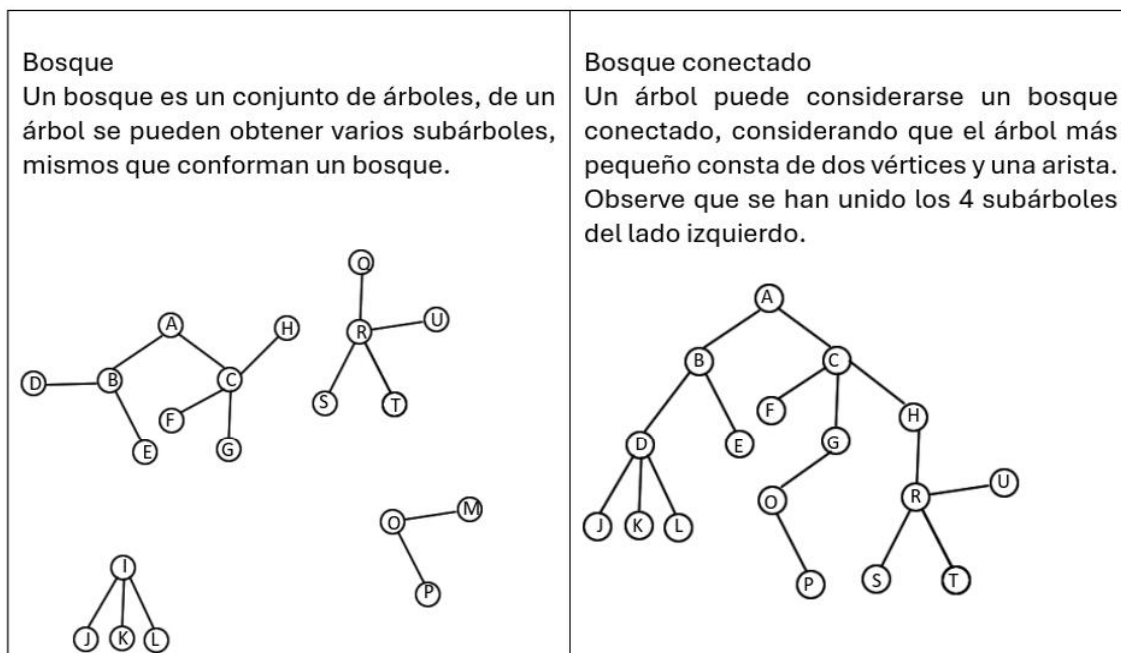


Figura 6.4. Ejemplo de un bosque

6.2 Árboles con peso

Los árboles con peso son aquellos en los cuales a los nodos se les ha asignado un valor. Una vez asignado un valor a los nodos es posible realizar algunas operaciones.

Por ejemplo, para representar caracteres en la computadora, es muy común utilizar el código ASCII, este código consiste en asignar a cada letra, número o signo especial un número o código que corresponde a su representación. En el código ASCII todos los caracteres se representan con una cadena de 8 caracteres. Sin embargo, es común que la mayoría de los caracteres no se utilicen con la misma frecuencia. Las cadenas más pequeñas pueden representar a los caracteres que se presentan con mayor frecuencia como son las vocales y algunas consonantes.

Para codificar la información, los bits se colocan en un árbol binario completo donde las cadenas de bits de los caracteres más frecuentes están más cerca de la raíz y los que casi no se usan están más alejados de la raíz. La información del símbolo almacenado se encuentra en los nodos hoja, mientras que las ramas etiquetadas con 0 y 1 del árbol binario muestran el carácter cifrado en la trayectoria de la raíz a la hoja. Esta técnica fue desarrollada por David A. Huffman y se conoce como “código de Huffman”, mediante esta estrategia, se va recorriendo el árbol desde la raíz hasta llegar un nodo hoja, y reconociendo la trayectoria como un carácter, al terminar de leer o codificar un carácter, se continua con el siguiente carácter a leer empezando nuevamente en la raíz y hasta llegar al nodo hoja, y así sucesivamente hasta terminar de descifrar el mensaje.

El algoritmo de codificación de Huffman, es un algoritmo esencial en la compresión de datos, pues permite reducir el número de bits que se requieren para representar la información. Existen muchos programas que permiten compactar la información de manera que el archivo original pese menos. El código de Huffman es muy utilizado para comprimir cadenas de bits que representan texto, así como compresión de archivos de imágenes y audio.

El código de Huffman

Mediante el código de Huffman se construye un árbol binario completo, en donde cada nodo padre del árbol tiene dos hijos, el hijo izquierdo y el hijo derecho. Todas las ramas del árbol están etiquetadas con 0 y 1, que corresponden al hijo izquierdo e hijo derecho. Los nodos hoja del árbol tienen asignado un símbolo. Para descifrar o leer un mensaje con el código de Huffman, se recorren los nodos del árbol, desde la raíz hasta la hoja para descifrar un símbolo, continuando con el siguiente símbolo hasta terminar el mensaje. Los pasos del algoritmo se mencionan a continuación.

1. Iniciar en la raíz del árbol.
2. Si el bit es 0 se avanza por la rama izquierda.
3. Si el bit es 1 se avanza por la rama derecha.
4. Continuar recorriendo todas las ramas del árbol hasta llegar a la hoja. Cada hoja tiene asignado un símbolo, se registra y se continua con el siguiente símbolo hasta terminar de descifrar el mensaje.

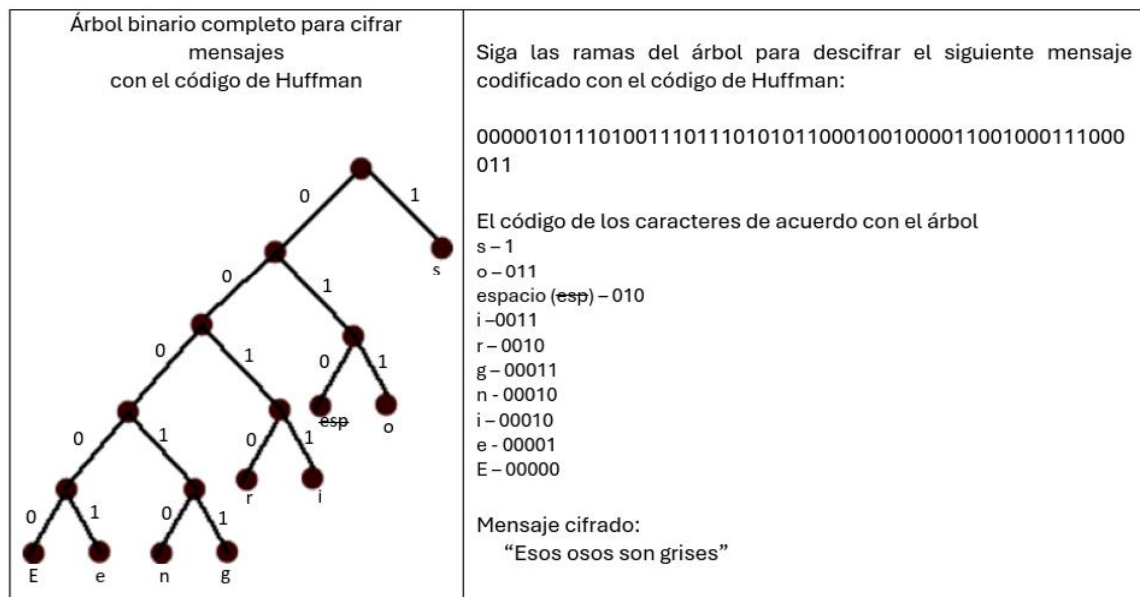


Figura 6.5. Ejemplo de descifrado de un mensaje con el código Huffman

Los caracteres con cadenas más pequeñas son producto de la frecuencia de uso. Cuando se desea compactar la información de un documento, lo recomendable es primero revisarlo para determinar el número de veces que se presenta cada uno de los caracteres que integran dicho documento, posteriormente se elabora el árbol óptimo que permita codificar con menos bits los caracteres que se repiten más y con un número de bits más grande los que se repiten menos. Estrategias similares utiliza el software que compacta información.

Construcción del árbol binario óptimo basado en el código de Huffman

Se desea construir un árbol binario basado en el código de Huffman para cifrar el mensaje: "Esos osos son grises". Para cifrar y compactar el mensaje, se tomará en cuenta la frecuencia con que se repiten los caracteres en el mensaje.

A continuación se mencionan los pasos para la construcción del Árbol óptimo utilizando el Algoritmo de Huffman:

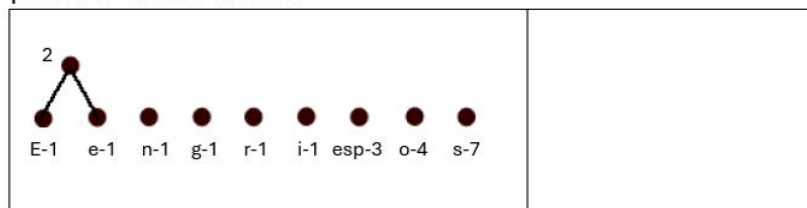
1. Especificar los símbolos utilizados en el mensaje y la frecuencia o el peso de uso de cada uno de los símbolos.

Símbolo	Peso o frecuencia
E	1
s	7
o	4
Espacio	3
n	1
g	1
r	1
i	1
e	1

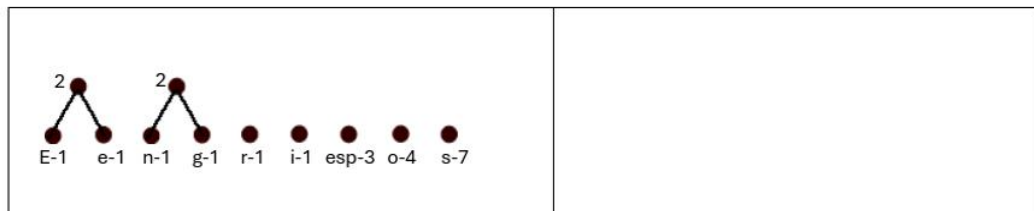
2. Ordenar los pesos de menos a mayor

Símbolo	Peso o frecuencia
n	1
g	1
r	1
i	1
e	1
E	1
Espacio	3
o	4
s	7

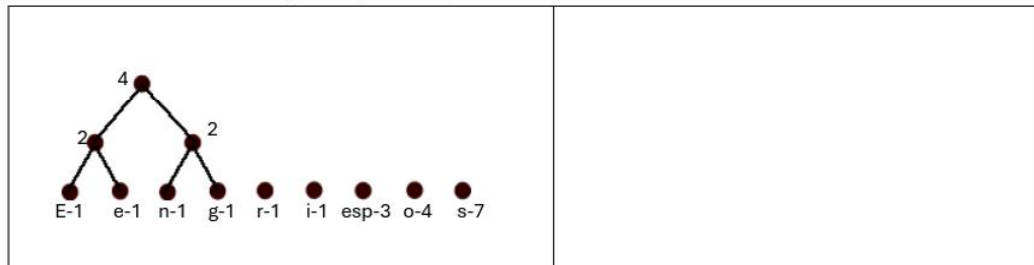
3. Una vez ordenados los pesos de menor a mayor, se inicia la construcción del árbol, unir los dos pesos de menor frecuencia como nodos izquierdo y derecho del nuevo árbol, y como raíz se agrega un nodo padre cuyo valor corresponde a la suma de los pesos de dichos nodos.



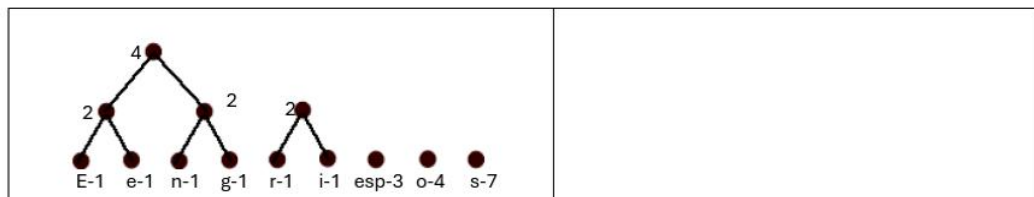
4. Continuar agregando nodos al árbol en construcción. Se tienen dos opciones para agregar nodos: tomar de la lista de frecuencias el siguiente nodo de menor peso y el nodo padre recién creado y unirlos como hijos izquierdo y derecho, a estos nodos se les agrega un nuevo nodo padre; o la segunda opción es tomar los siguientes dos nodos candidatos de la tabla de frecuencias y unir como nodos izquierdo y derecho del árbol, agregando un nodo padre cuyo valor es la suma de las frecuencias de sus hijos izquierdo y derecho. Se elige aquella opción cuya suma de los pesos de sus frecuencias sea menor. En el caso del ejemplo, al unir el nodo padre con el nuevo nodo tomado de la tabla de frecuencias la suma es: $2+1=3$ y si se unen los siguientes dos nodos de la lista de frecuencias la suma que se obtiene es $1+1=2$, por lo que se procede a formar un nuevo subárbol tomando los dos siguientes elementos de la tabla de frecuencias.



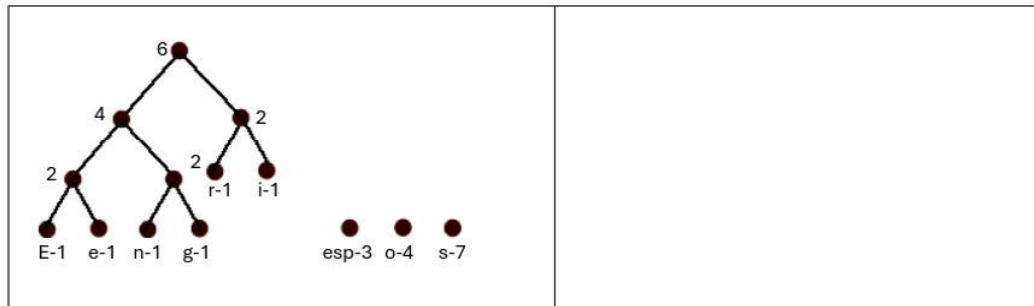
5. Si los dos nodos que se unieron, se tomaron de la tabla de frecuencias, entonces se unen los dos subárboles y se agrega un nuevo nodo padre que corresponde a la suma del valor de las raíces de cada subárbol que se unió. Si se seleccionó un solo nodo de la tabla de frecuencias y se unió con el subárbol existente, de igual manera se debe agregar un nuevo nodo padre para unir los nodos y el valor del nodo padre es la suma de los valores de sus hijos izquierdo y derecho.



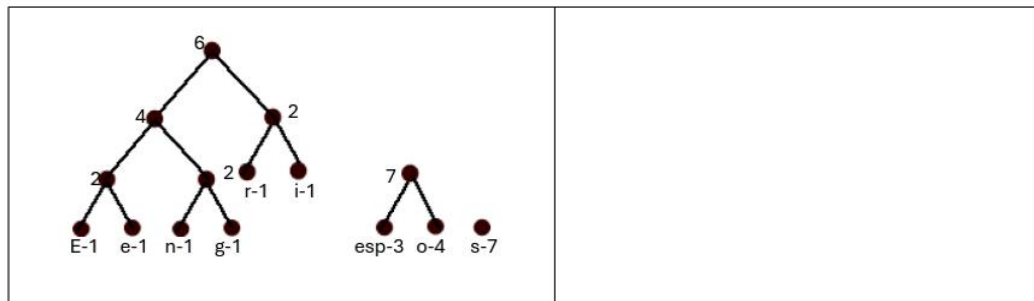
6. Continuar el mismo procedimiento buscando unir la raíz del árbol actual con el siguiente elemento de la tabla de frecuencias cuyo resultado es $4+1=5$, o unir los siguientes dos elementos de la tabla de frecuencias, que en este caso es $1+1=2$, por lo que se unen los siguientes dos elementos de la tabla de frecuencias formando un nuevo subárbol.



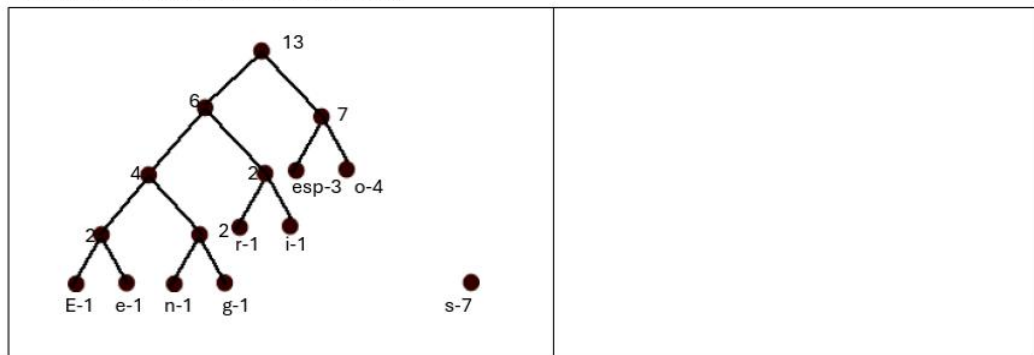
7. Si los dos nodos que se unieron, se tomaron de la tabla de frecuencias, entonces se unen los dos subárboles y se agrega un nuevo nodo padre que corresponde a la suma del valor de las raíces de cada subárbol que se unió. Si se seleccionó un solo nodo de la tabla de frecuencias y se unió con el subárbol existente, de igual manera se debe agregar un nuevo nodo padre para unir los nodos y el valor del nodo padre es la suma de los valores de sus hijos izquierdo y derecho. En este caso se unen los dos subárboles.



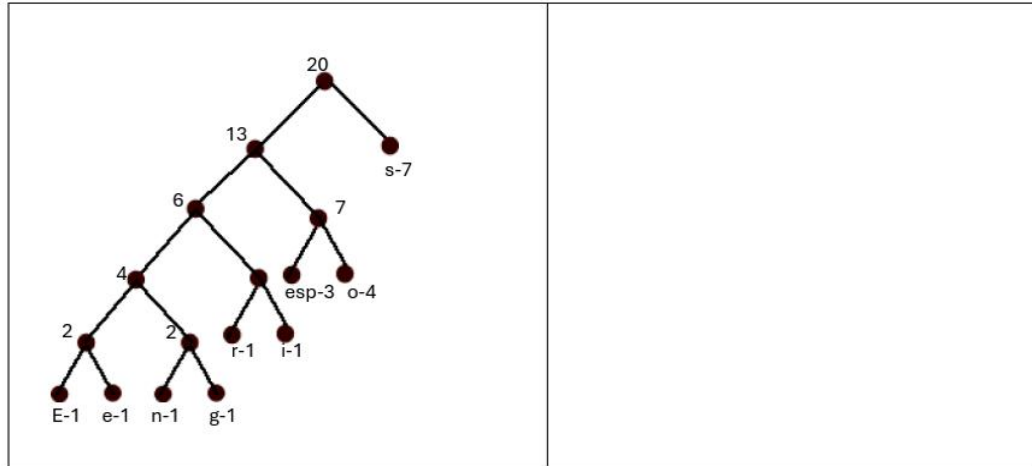
8. Continuar con el mismo procedimiento, al sumar los pesos la raíz del árbol en construcción con un elemento de la tabla de frecuencias la suma es $6+3=9$, y al sumar dos nodos de la tabla de frecuencias el resultado es $3+4=7$, por lo que se elige la segunda opción.



9. Unir los dos subárboles actuales



10. Unir al árbol el nodo restante



6.2.1 Recorrido de un árbol

La operación de recorrer la estructura de datos llamada árbol consiste en visitar cada uno de los vértices a partir del nodo raíz. El recorrido inicia en el nodo raíz y se visitan todos los nodos de acuerdo con un orden especificado. Existen dos formas de recorrer el árbol: búsqueda a lo ancho y búsqueda en profundidad.

Búsqueda a lo ancho

La búsqueda comienza en la raíz y después se examinan todos sus hijos, comenzando de izquierda a derecha. Si la información que se busca no se encuentra en los descendientes directos de la raíz, entonces se procede a buscar en el siguiente nivel también de izquierda a derecha, y así sucesivamente hasta encontrar la información o recorrer todo el árbol sin encontrar la información.

Búsqueda en profundidad

Se comienza la búsqueda en el nodo raíz, después se busca en el hijo de la izquierda y si este nodo tiene hijos se continua la búsqueda con su hijo izquierdo y así sucesivamente hasta llegar a la parte más alejada de la raíz del árbol. Si este ya no tiene hijo izquierdo, se continua la búsqueda con el hijo de la derecha (o el que se encuentra a la derecha en caso de no ser un árbol binario) hasta llegar a la hoja. Si no se ha encontrado la información, se recorre el camino andado hasta el nodo inmediato anterior que tenga hijos y cuyos hijos no hayan sido visitados, dando preferencia al que se encuentra más a la izquierda. Cuando se llega nuevamente a la hoja, se regresa hasta el nodo inmediato anterior que tenga hijos sin visitar y así sucesivamente. En la búsqueda en profundidad se regresa hasta el nodo raíz (lo cual significa que se visitó toda la rama dependiente del hijo izquierdo de la raíz). Si la raíz tiene hijos que aún no han sido inspeccionados se selecciona el nodo más cercano al hijo izquierdo y todos sus descendientes, dando preferencia a los de la izquierda. Este procedimiento se lleva a cabo hasta encontrar la información o bien recorrer todo el árbol.

Recorrido de un árbol

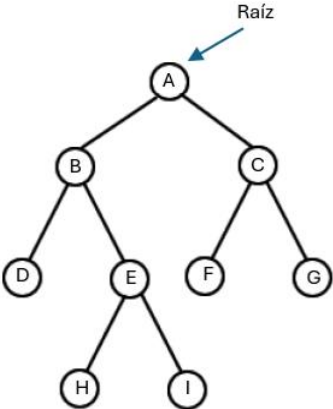
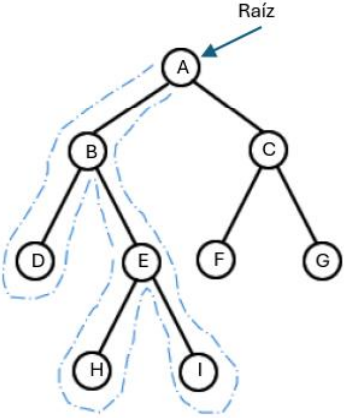
Existen diferentes procedimientos para almacenar la información en un árbol, la manera en que se almacena la información impacta directamente la forma en que se recupera la misma. Existen 3 formas de recorrer la información de un árbol. Estos recorridos son procedimientos recursivos, es decir, el procedimiento se sigue en forma repetida hasta llegar a una condición de paro, que en el caso de los árboles el procedimiento se detiene al llegar a las hojas. Los recorridos de un árbol son:

1. Recorrido en pre-orden. (padre, hijo izquierdo hijo, derecho). En este recorrido, primero se visita el padre, posteriormente el hijo izquierdo y finalmente el hijo derecho. El recorrido comienza en la raíz, posteriormente se visita al hijo izquierdo, si este nodo tiene hijos, se sigue siempre por el hijo de la izquierda hasta llegar a la hoja. Una vez que se llega a la hoja, es decir, se han visitado todos los nodos de la izquierda, si este nodo tiene hermanos, entonces se visita el hermano más cercano, es decir, el siguiente hermano más a la izquierda. El procedimiento recursivo comienza regresando hacia el nodo padre y buscando el siguiente hermano que no haya sido visitado hasta llegar a la raíz, lo que indica que ya se ha recorrido todo el subárbol izquierdo de la raíz. Se continúa con el mismo procedimiento ahora con el subárbol de la derecha.
2. Recorrido en in-orden. (hijo izquierdo, padre, hijo derecho). En este recorrido primero se visita al hijo izquierdo, posteriormente se visita al padre, y por último al hijo derecho. En la trayectoria, el primer nodo que se visita es la hoja que se encuentra más hacia la izquierda del árbol, posteriormente se regresa un nivel y visita al padre y posteriormente se visita al hijo derecho. Una vez que se visitó el hijo derecho, entonces se continua visitando el siguiente padre que no haya sido visitado aún, se continua con su rama derecha y así sucesivamente hasta recorrer todo el árbol.
3. Recorrido en pos-orden. (hijo izquierdo, hijo derecho, padre). Para iniciar el recorrido se visita al hijo izquierdo, y si éste tiene hijos se visita a su hijo izquierdo, y así sucesivamente hasta llegar al nodo hoja. Al llegar al nodo hoja, se regresa al nodo padre de la hoja y se continua ahora buscando a su hijo derecho, se visita si es que existe y posteriormente se visita al padre del nivel superior. Una vez recorrido este subárbol, se comienza el proceso recursivo, buscando el siguiente hijo derecho de los nodos más cercanos que no hayan sido visitados, se continua con el nodo padre, y así sucesivamente hasta recorrer todos los nodos. En este recorrido se visitan primero los hijos y posteriormente su nodo padre.

Para realizar los diferentes recorridos del árbol, se lleva un registro de los nodos en el orden en que han sido visitados, ningún nodo aparece en el registro más de una vez y ningún nodo queda pendiente de visitar. La operación de visitar un nodo significa que se tiene acceso a la información que ahí se encuentre, en este caso se identifica con la letra de cada nodo.

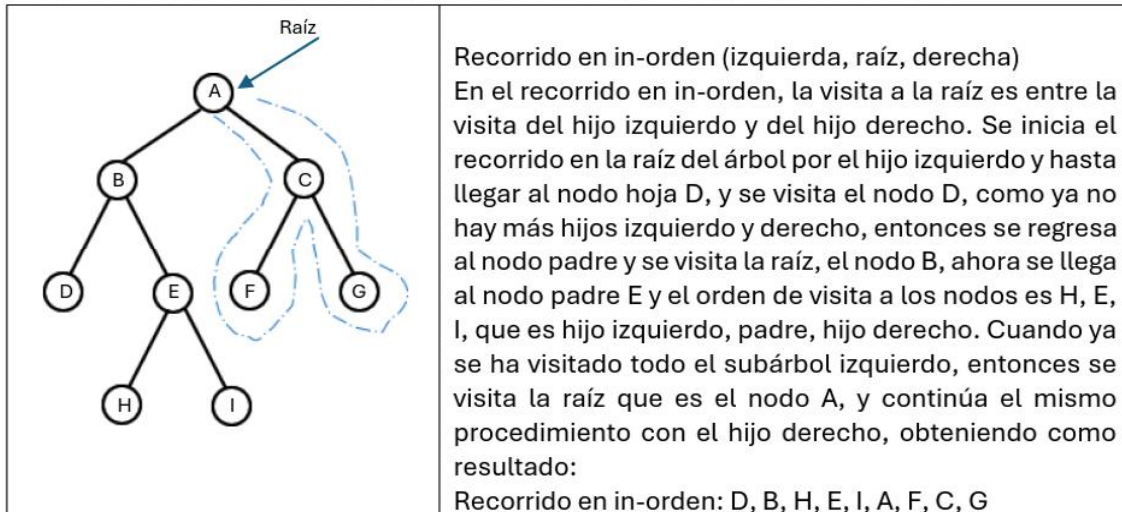
Recorrido en pre-orden

En el recorrido en pre-orden el primer elemento que se visita es la raíz, es por ello que el nodo A es el primero que aparece en la lista, posteriormente se intenta localizar la raíz del subárbol izquierdo y posteriormente del subárbol derecho iniciando el procedimiento de búsqueda en pre-orden en los subárboles correspondientes.

	Recorrido en pre-orden (raíz, izquierda, derecha): El recorrido inicia en la raíz, y es el primer nodo que se visita, posteriormente, intenta encontrar el hijo izquierdo del nodo padre actual, que en este caso es la raíz y viaja hacia el nodo B y visita el nodo B, que en este momento es el nodo padre actual, se busca el hijo izquierdo de B, que en este caso es D y también se visita. Cuando se llega a un nodo hoja, como en este caso al nodo D, que ya no tiene más hijos izquierdo y derecho, entonces se regresa al nodo padre para continuar el recorrido buscando al hijo derecho de B. El hijo derecho de B, es el nodo E, se visita y se busca el hijo izquierdo de E, en este caso el nodo H, se visita y como es un nodo hoja, continua ahora buscando al hijo derecho de E y todos sus árboles correspondientes.
	En el recorrido pre-orden, cada nodo al que se llega durante la trayectoria, se visita, y posteriormente se sigue por la rama izquierda. Cuando se llega a un nodo hoja y se visita, al no tener más hijos izquierdo y derecho y debido a que el nodo padre ya fue visitado, entonces empieza el retorno buscando al primer hijo derecho que se encuentre de las raíces que ya han sido visitadas. Este orden de búsqueda asegura que todos los nodos del subárbol izquierdo de la raíz hayan sido visitados, así como el nodo raíz, que fue el primero en ser visitado. El procedimiento continúa realizando las mismas acciones en el subárbol derecho, obteniendo como resultado: Recorrido en pre-orden: A, B, D, E, H, I, C, F, G

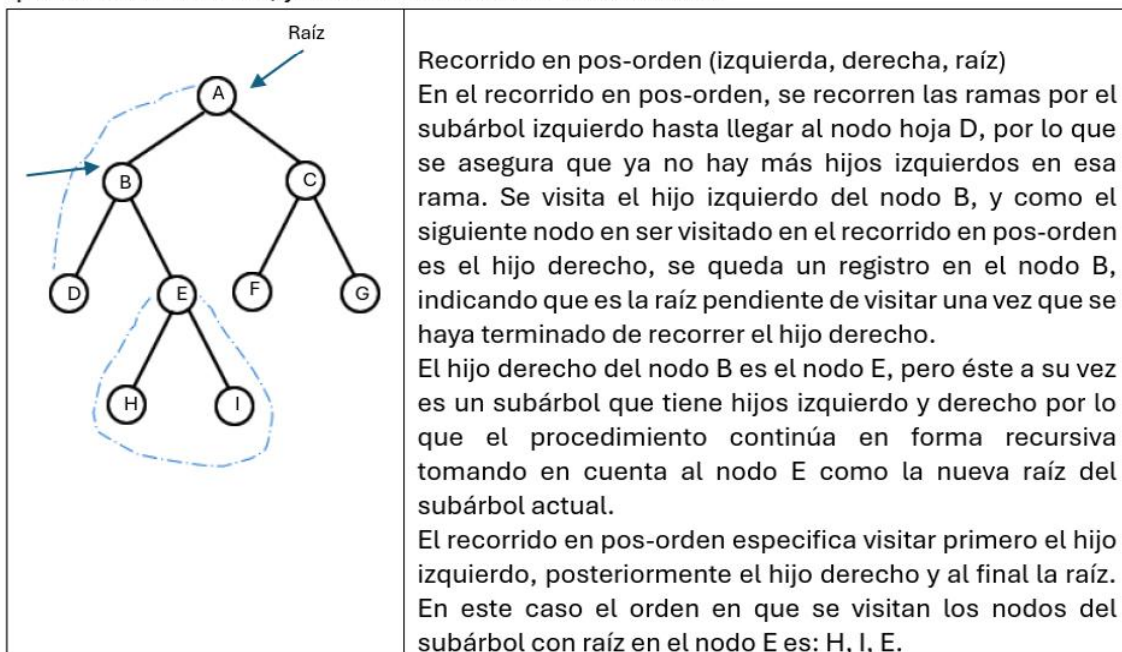
Recorrido en in-orden

En la lista que muestra la trayectoria de los nodos recorridos se observa que en el recorrido en in-orden, el nodo A es visitado entre los nodos del subárbol izquierdo y los nodos del subárbol derecho del nodo A. Asimismo, al visitar el subárbol con raíz en B, el nodo B queda registrado entre el nodo del hijo izquierdo que es D y los nodos del subárbol derecho de B.



Recorrido en pos-orden

El recorrido en pos-orden indica que el nodo raíz es el último nodo en visitar, una vez que hayan sido visitados los subárboles izquierdo y derecho de A. Como se puede observar, en el subárbol derecho de la raíz A, el orden en que se listan los nodos es primero el hijo izquierdo que es el nodo F, posteriormente el hijo derecho que es el nodo G y al final la raíz del subárbol que en este caso es C, y el último nodo en ser visitado es A.



	Como se ha terminado de recorrer el hijo derecho de B, y ya se visitó el hijo izquierdo, entonces se procede a visitar el nodo B, que es la raíz del subárbol izquierdo del nodo A. Posteriormente se procede a visitar el hijo derecho del nodo A que es la raíz siguiendo el mismo orden que indica el recorrido en pos-orden, y finalmente se visita el nodo A. Recorrido en pos-orden: D, H, I, E, B, F, G, C, A
--	---

6.3 Redes

Una red (o red de transporte) es un conjunto de nodos interconectados a través de aristas que tienen la función de compartir la información entre los nodos. Las redes, al igual que los árboles son grafos que se utilizan para modelar una gran diversidad de problemas. Por ejemplo, las redes de computadoras, donde las computadoras se comunican a través de dispositivos alámbricos o inalámbricos que permiten enviar y recibir información mediante paquetes de datos, permitiendo compartir recursos y actuando como una unidad organizada. Diseñar una red de computadoras permite administrar una comunicación interna para compartir la ejecución de programas, el acceso a internet, administración de dispositivos periféricos como impresoras y escáneres.

Existen muchos procesos que pueden modelarse utilizando las redes, una de las principales características de las redes es que cuentan con procesos de emisión y recepción de mensajes entre los nodos por lo que se pueden utilizar para problemas de investigación de operaciones como optimización de rutas. En este tema se tratan problemas de maximizar el flujo a través de una red. Las redes pueden ser una tubería por la que se transporta un líquido, las redes de computadoras que comparten información, la optimización de rutas para recorrer un conjunto de ciudades en un país, etc.

En el diseño de problemas con redes se pretende maximizar el flujo máximo, que es un problema que involucra tanto a la teoría de grafos como a la investigación de operaciones. Una red o también se denomina red de transporte es un grafo simple, dirigido y ponderado que cumple las siguientes condiciones:

1. Un vértice de la red se designa como origen o fuente y no tiene aristas entrantes.
2. Un vértice se designa como destino y no tiene aristas salientes.
3. Existe un peso o valor no negativo de la arista dirigida que conecta el nodo i hacia el nodo j , que se denomina capacidad de (i,j) .

A cada arista en la red, se le asigna un flujo que no excede la capacidad del peso de la arista. Además, para cada vértice v , que no es el vértice origen ni el vértice destino de la red, se tiene que el flujo que entra al vértice v , es igual al flujo que sale del vértice v , sin sobrepasar la capacidad de cada vértice.

Se llama conservación del flujo a la condición de no utilizar ni suministrar el flujo en las llamadas estaciones de bombeo que son los nodos que no son fuente ni destino.

6.3.1 Teorema de flujo máximo

En una red de transporte, un flujo máximo en el grafo G , es el flujo que tiene un valor máximo para transportar en las aristas, dadas las capacidades permitidas de los vértices. En un grafo, podría haber varias trayectorias con el flujo máximo, mediante el algoritmo de flujo máximo, se encuentra este valor; en este algoritmo, se comienza con un flujo inicial y se incrementa en forma iterativa, hasta que ya no sea posible mejorarlo.

6.3.2 Teorema de flujo mínimo

En una red de transporte, un corte de flujo mínimo en el cual quedan dos partes disjuntas del conjunto de vértices V_1 y V_2 que situados en la red, dejan la fuente en una de ellas y el destino en el otro conjunto.

6.3.3 Pareos y redes de Petri

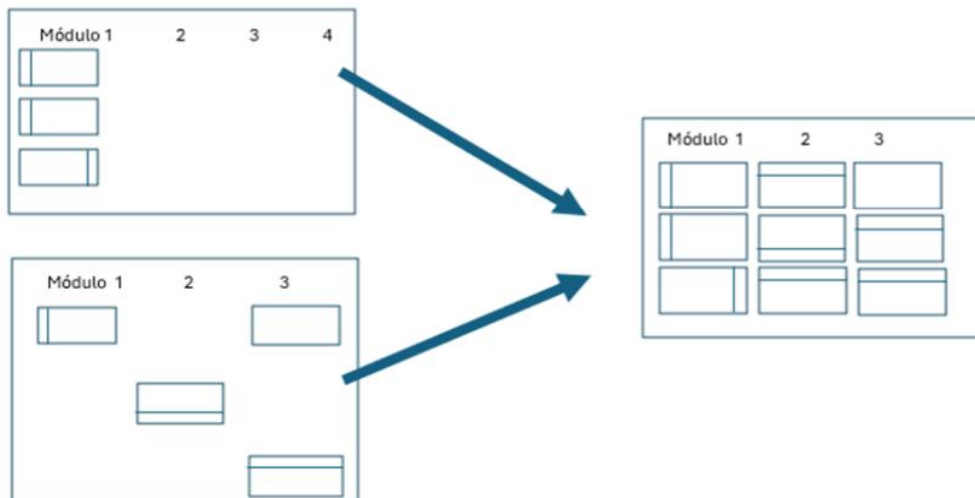
Las redes de Petri son grafos dirigidos cuyos elementos principales son los lugares y las transiciones. Los lugares representados por los vértices del grafo y las transiciones representadas por las aristas.

Bibliografía para la materia de Matemáticas discretas

1. **Tocci Ronald J.**; "Sistemas digitales, principios y aplicaciones"; Pearson Prentice Hall; Décima edición; México 2007; **ISBN: 978-970-26-0970-4** .
2. **Johnsonbaugh Richard**; "Matemáticas discretas"; Pearson Educación; Sexta edición; México, 2005; **ISBN: 970-26-0637-3**; páginas 696.
3. **Jiménez Murillo, José A.**; "Matemáticas para la computación"; Ed. Alfaomega; 1ª edición; México, 2009; **ISBN: 978-970-15-1401-6**; páginas 496
4. **Suppes Patrick, Hill Shirley**; "Primer curso de lógica matemática"; Ed. Reverté; Colombia, 1988; **ISBN: 84-291-5150-8**.
5. **Kenneth H. Rosen**; "Matemática Discreta y sus aplicaciones"; Mc Graw Hill; 5ª edición; España, 2004; **ISBN: 84-481-4073-7**.
6. **Villalpando Becerra, J. F.**; "Matemáticas discretas: aplicaciones y ejercicios"; Grupo Editorial Patria, 2014; páginas 369; **ISBN: 9786074389258**; Disponible en: <https://elibro.net/es/ereader/tecnmcdmadero/39454?page=1>
7. **Pérez Águila, R.**; "Una introducción a las matemáticas discretas y teoría de grafos"; Ed. Santa Fe: El Cid Editor, 2014; páginas 319; **ISBN: 978-1-4135-2434-5**; Disponible en: <https://elibro.net/es/ereader/tecnmcdmadero/36562?page=10>
8. **García Muñoz, M. A.**; "Matemática discreta para la computación: nociones teóricas y problemas resueltos"; Ed. Jaén, 2017; páginas 753; **ISBN: 978-84-9159-033-0**; Disponible en: <https://elibro.net/es/ereader/tecnmcdmadero/59076?page=4>

Apuntes de la asignatura:

Ingeniería de software



1. Datos Generales de la asignatura

Nombre de la asignatura:	Ingeniería de Software
Clave de la asignatura:	SCD-1011
SATCA¹:	2 -3-5
Carrera:	Ingeniería en Sistemas Computacionales

2. Presentación

Caracterización de la asignatura

Esta asignatura aporta al perfil del Ingeniero en Sistemas Computacionales las siguientes habilidades:

- Implementa aplicaciones computacionales para solucionar problemas de diversos contextos, integrando diferentes tecnologías, plataformas o dispositivos
- Diseña e implementa interfaces para la automatización de sistemas de hardware y desarrollo del software asociado.
- Coordina y participa en equipos multidisciplinarios para la aplicación de soluciones innovadoras en diferentes contextos.
- Desarrolla y administra software para apoyar la productividad y competitividad de las organizaciones cumpliendo con estándares de calidad.

La importancia de esta asignatura, es que permite al estudiante aplicar las fases de la metodología para el desarrollo de un sistema en un contexto multidisciplinario; aplicando el conocimiento científico, a través de los métodos, técnicas y normas adecuados, para el desarrollo de software.

La disciplina de Ingeniería de Software se relaciona con materias precedentes como: Fundamentos de Programación, Programación Orientada a Objetos, Estructura de Datos, Fundamentos de Ingeniería de Software, Tópicos Avanzados de Programación, Sistemas Operativos, Arquitectura de Computadoras y Taller de Bases de Datos; y con las materias posteriores: Gestión de Proyectos de Software y Programación Web.

Requiere de competencias previas como: Manejo de un lenguaje de modelado, dominio en el uso de herramientas CASE, uso de algún Sistema Manejador de Bases de Datos, dominio de algún lenguaje de programación orientado a objetos, identificación de las etapas del ciclo de desarrollo de sistemas y de las diferentes plataformas operativas

Intención didáctica

La asignatura debe ser teórico – práctico, y capaz de desarrollar en el estudiante la habilidad para la aplicación de las diferentes técnicas en el desarrollo de software, considerando siempre los principios de la ingeniería de software, para lo cual se organiza el temario en cuatro temas.

¹ Sistema de Asignación y Transferencia de Créditos Académicos

En el tema uno, el Modelo de Análisis, es generado a través de la obtención de requisitos, selección y uso de herramientas CASE, se realiza el estudio de factibilidad mediante la información obtenida estableciendo así el costo- beneficio del sistema; es de suma importancia profundizar en el modelo de requisitos, casos de uso, documentación de estos últimos y derivar en el modelo de dominio del sistema a desarrollar.

En el tema dos, Diseño, se integra la información obtenida en el análisis, se elaboran los modelos de: clases, de datos, diagramas de secuencia o colaboración y diseño de interfaz, estas últimas orientadas a los casos de uso.

En el tema tres, Desarrollo, se construye el sistema tomando como base la selección de los lenguajes de programación, manejadores de bases de datos y desarrollo rápido de las aplicaciones, estableciendo la validación y consistencia del sistema.

En el tema cuatro, Pruebas e Implantación, se realiza las diferentes pruebas para verificar la eficiencia del sistema generando las pruebas del sistema, pruebas de componentes, diseño de caso de prueba, documentación de las pruebas; una vez probado se implanta y se procede a la capacitación a usuarios dando un tiempo para la estabilidad del mismo. Una vez que se estabiliza se procede a liberar y entregar la documentación del sistema del cliente.

El papel del docente es ejemplificar cada etapa de la Ingeniería de software y propiciar el trabajo en equipo para atender proyectos del contexto que induzcan al estudiante a la aplicación de la metodología en un ambiente real.

3. Participantes en el diseño y seguimiento curricular del programa

Lugar y fecha de elaboración o revisión	Participantes	Evento
Instituto Tecnológico de Saltillo del 5 al 9 de octubre de 2009.	Representantes de los Institutos Tecnológicos de: Alvarado, Arandas, Campeche, Celaya, Centla, Cerro Azul, Champotón, Ciudad Acuña, Ciudad Cuauhtémoc, Ciudad Juárez, Ciudad Madero, Ciudad Valles, Coatzacoalcos, Cocola, Colima, Comitán, Durango, El Istmo, Huetamo, La Laguna, La Paz, Lázaro Cárdenas, Lerdo, Libres, Linares, Macuspana, Matamoros, Mérida, Mexicali, Morelia, Nuevo Laredo,	Reunión Nacional de Diseño e Innovación Curricular para el Desarrollo y Formación de Competencias Profesionales de las Carreras de Ingeniería en Sistemas Computacionales, Ingeniería Informática e Ingeniería en Geociencias.

	Nuevo León, Occidente del Estado de Hidalgo, Ocotlán, Orizaba, Oriente del Estado de Hidalgo, Parral, Piedras Negras, Pinotepa, Saltillo, San Luis Potosí, Sur de Guanajuato, Sur del Estado de Yucatán, Tapachula, Tepexi de Rodríguez, Teziutlán, Tijuana, Toluca, Tuxtepec, Veracruz, Villahermosa, Xalapa, Zacatecas y Zacatepec.	
Instituto Tecnológico Superior de Poza Rica del 22 al 26 de febrero de 2010.	Representantes de los Institutos Tecnológicos de: Alvarado, Arandas, Campeche, Celaya, Centla, Cerro Azul, Champotón, Ciudad Acuña, Ciudad Cuauhtémoc, Ciudad Juárez, Ciudad Madero, Ciudad Valles, Coatzacoalcos, Cocula, Colima, Comitán, Durango, El Istmo, Huetamo, La Laguna, La Paz, Lázaro Cárdenas, Lerdo, Libres, Macuspana, Matamoros, Mérida, Mexicali, Morelia, Nuevo Laredo, Nuevo León, Occidente del Estado de Hidalgo, Orizaba, Oriente del Estado de Hidalgo, Parral, Piedras Negras, Pinotepa, Saltillo, San Luis Potosí, Sur de Guanajuato, Sur del Estado de Yucatán, Tapachula, Tepexi de Rodríguez, Teziutlán, Tijuana, Toluca, Tuxtepec, Veracruz, Villahermosa, Xalapa, Zacatecas y Zacatepec.	Reunión Nacional de Consolidación de los Programas en Competencias Profesionales de las Carreras de Ingeniería en Sistemas Computacionales, Ingeniería Informática e Ingeniería Petrolera del SNEST.
Instituto Tecnológico de Querétaro del 22 al 25 de octubre de 2012.	Representantes de los Institutos Tecnológicos de: Acayucan, Altamira, Cajeme, Campeche, Cananea, Cd. Acuña, Cd. Cuauhtémoc, Cd. Juárez, Cd. Madero, Cd. Valles, Celaya, Centla, Cerro Azul, Chetumal, Chihuahua II,	Reunión Nacional de Seguimiento Curricular de los Programas en Competencias Profesionales de las Carreras de Ingeniería en Sistemas Computacionales, Ingeniería Informática e Ingeniería en

	Chilpancingo, Coalcomán, Coatzacoalcos, Cocula, Colima, Comalcalco, Delicias, Durango, Ébano, Escárcega, Huixquilucan, La Paz, León, Lerdo, Los Ríos, Macuspana, Mante, Milpa Alta, Minatitlán, Morelia, Nuevo Laredo, Nuevo León, Oaxaca, Oriente del Estado de México, Oriente del Estado de Hidalgo, Pachuca, Piedras Negras, Progreso, Puerto Vallarta, Purhepecha, Tacámbaro, Tehuacán, Tepexi de Rodríguez, Tepic, Teposcolula, Teziutlán, Tierra Blanca, Tijuana, Tlaxiaco, Toluca, Tuxtepec, Uruapan, Valladolid, Veracruz, Villahermosa, Zacatecas, Zacatecas Norte, Zacatepec, Zapopan, Zitácuaro y Zongolica.	Tecnologías de la Información y Comunicaciones.
Instituto Tecnológico de Toluca, del 10 al 13 de febrero de 2014.	Representantes de los Institutos Tecnológicos de: Cerro Azul, Colima, Lerdo, Toluca y Veracruz.	Reunión de Seguimiento Curricular de los Programas Educativos de Ingenierías, Licenciaturas y Asignaturas Comunes del SNIT.

4. Competencia(s) a desarrollar

Competencia(s) específica(s) de la asignatura
Desarrolla soluciones de software, considerando la metodología y herramientas para la elaboración de un proyecto aplicativo en diferentes escenarios.

5. Competencias previas

- Diseña y desarrolla programas para la solución de problemas computacionales utilizando el paradigma orientado a objetos.
- Desarrolla soluciones de software para resolver problemas en diversos contextos utilizando programación concurrente, acceso a datos, que soporten interfaz gráfica de usuario y consideren dispositivos móviles.
- Crea y gestiona bases de datos para resolver problemas del contexto considerando la concurrencia e interoperabilidad de los datos.
- Realiza el análisis de un proyecto de software, a partir de la identificación del modelo de negocios de la organización que permitan alcanzar estándares y métricas de calidad.

6. Temario

No.	Temas	Subtemas
1.	Análisis.	1.1 Revisión de especificación de requisitos. 1.1.1 Norma IEEE830 1.1.2 Trazabilidad de requisitos 1.2 Descripción de procesos actuales 1.3 Diagramas UML 1.4 Estudio de Factibilidad 1.5 Análisis Costo-Beneficio
2.	Diseño.	2.1 Diseño de procesos propuestos 2.1.1 Herramientas CASE para diseño 2.2 Diseño arquitectónico 2.3 Diseño de datos 2.4 Diseño de interfaz de usuario
3.	Desarrollo.	3.1 Lenguajes para el desarrollo de software 3.2 Manejadores de bases de datos 3.3 Construcción del sistema
4	Pruebas e Implantación.	4.1 Diseño de caso de prueba 4.2 Pruebas de componentes 4.3 Pruebas del sistema 4.4 Documentación de resultados de las pruebas. 4.5 Entrega del sistema y Capacitación a usuarios 4.6 Entrega de documentación técnica y de usuario del sistema.

7. Actividades de aprendizaje de los temas

1. Modelo de Análisis	
Competencias	Actividades de aprendizaje
Específica(s): - Abstrae información del usuario final para elaborar el análisis de requerimientos del software a desarrollar. Genéricas: - Trabajo en equipo - Capacidad de analizar información provenientes de diversas fuentes - Habilidad de comunicación oral y escrita. - Propuesta de solución de problemas - Toma de decisiones. - Habilidades interpersonales - Capacidad de aplicar los conocimientos en la práctica.	- Seleccionar una problemática real del entorno para ofrecer una posible solución. - Realizar una entrevista con el cliente para determinar los requerimientos del sistema. - Elaborar una carta compromiso Cliente-Desarrollador para delimitar los alcances del sistema. - Aplicar la notación UML para representar el modelo de requisitos, casos de uso, documentación de casos de uso y modelo del dominio. - Realizar un estudio de factibilidad técnica y operativa para justificar el costo beneficio del sistema. - Definir la metodología de desarrollo de software.
2. Modelo de diseño	
Competencias	Actividades de aprendizaje
Específica(s): - Aplica modelos, técnicas y herramientas para la etapa de diseño del software. Genéricas: - Trabajo en equipo - Capacidad de analizar información provenientes de diversas fuentes - Habilidad de comunicación oral y escrita. - Propuesta de solución de problemas - Toma de decisiones.	A partir del modelo de análisis del proyecto propuesto se realizarán las siguientes actividades utilizando la herramienta CASE correspondiente: - Modelo de datos, verificando que el diagrama de datos refleje una base de datos normalizada. - Diagramas de secuencia, cuidando que refleje la comunicación efectiva entre las clases dentro de un proceso. - Interfaces del sistema, cuidando que cada caso de uso, tenga asociada la interfaz correspondiente.

- Habilidades interpersonales - Capacidad de aplicar los conocimientos en la práctica.	
3. Desarrollo	
Competencias	Actividades de aprendizaje
Específica(s): - Construye un software derivado de la problemática planteada en el análisis y diseño de un sistema. Genéricas: - Capacidad para usar nuevas tecnologías. - Trabajo en equipo - Habilidad de comunicación oral y escrita. - Propuesta de solución de problemas - Toma de decisiones. - Habilidades interpersonales - Capacidad de aplicar los conocimientos en la práctica.	- Seleccionar el lenguaje de programación y el sistema gestor de base de datos, pertinente para la construcción del sistema. - Codificar los procesos identificados en el sistema propuesto. - Elaborar la documentación técnica de la construcción del sistema.
4. Pruebas e implementación	
Competencias	Actividades de aprendizaje
Específica(s): - Evalúa el software construido a partir de las pruebas e implementación realizada. Genéricas: - Trabajo en equipo - Habilidad de comunicación oral y escrita. - Propuesta de solución de problemas - Toma de decisiones. - Habilidades interpersonales - Capacidad de aplicar los conocimientos en la práctica.	- Elaborar un plan de pruebas aplicable al software realizado. - Contrastar los resultados del software con el modelo de requisitos planteado en el análisis. - Establecer una visita a la empresa para implantar el sistema. - Hacer un programa de capacitación a usuarios del sistema. - Realizar adecuaciones resultado de la capacitación y revisión por el usuario final. - Liberar el sistema y hacer entrega de los documentos del mismo.

8. Práctica(s)

1. Elaboración y revisión del anteproyecto del sistema de desarrollo.
2. Con la ayuda de una herramienta CASE elabora el análisis del modelo de negocio seleccionado, considerando el modelo de requisitos, casos de uso, documentación de casos de uso y modelo de dominio.
3. Elaborar un estudio de factibilidad y el costo-beneficio aplicado a la organización.
4. Establecer un diseño preliminar de las interfaces de usuario de acuerdo a los requisitos.
5. Elaborar un diseño de bases de datos emanado del modelo entidad-relación.
6. A partir del diccionario de datos y el diagrama E-R crear una base de datos.
7. Usando un lenguaje de programación establecer la conexión a una base de datos.
8. Desarrollar los procesos identificados, asegurando las operaciones básicas de todo sistema: registro, actualización, consulta y estadística.
9. Probar el sistema con las técnicas existentes y validar que el modelo de requisitos esté atendido.
10. Implementar el sistema, capacitar a los usuarios y verificar la estabilidad del sistema para su liberación.

9. Proyecto de asignatura

El objetivo del proyecto que planteé el docente que imparta esta asignatura, es demostrar el desarrollo y alcance de la(s) competencia(s) de la asignatura, considerando las siguientes fases:

- **Fundamentación:** marco referencial (teórico, conceptual, contextual, legal) en el cual se fundamenta el proyecto de acuerdo con un diagnóstico realizado, mismo que permite a los estudiantes lograr la comprensión de la realidad o situación objeto de estudio para definir un proceso de intervención o hacer el diseño de un modelo.
- **Planeación:** con base en el diagnóstico en esta fase se realiza el diseño del proyecto por parte de los estudiantes con asesoría del docente; implica planificar un proceso: de intervención empresarial, social o comunitario, el diseño de un modelo, entre otros, según el tipo de proyecto, las actividades a realizar los recursos requeridos y el cronograma de trabajo.
- **Ejecución:** consiste en el desarrollo de la planeación del proyecto realizada por parte de los estudiantes con asesoría del docente, es decir en la intervención (social, empresarial), o construcción del modelo propuesto según el tipo de proyecto, es la fase de mayor duración que implica el desempeño de las competencias genéricas y específicas a desarrollar.
- **Evaluación:** es la fase final que aplica un juicio de valor en el contexto laboral-profesión, social e investigativo, ésta se debe realizar a través del reconocimiento de logros y aspectos a mejorar se estará promoviendo el concepto de “evaluación para la mejora continua”, la metacognición, el desarrollo del pensamiento crítico y reflexivo en los estudiantes.

10. Evaluación por competencias

Para evaluar las actividades de aprendizaje se recomienda solicitar: resúmenes, reportes de prácticas, estudios de casos, exposiciones en clase, reportes de visitas, desarrollo de proyecto, portafolio de evidencias.

Para verificar el nivel del logro de las competencias del estudiante se recomienda utilizar: listas de cotejo, matrices de valoración, rúbricas, guías de observación, coevaluación y autoevaluación.

11. Fuentes de información

Impresas:

1. Booch G. (2006). *El lenguaje Unificado de Modelado, UML 2.0*, Guía de Usuario. 2ª. Edición. España: Pearson ADDISON-WESLEY.
2. Braude, E. (2003). *Ingeniería de Software una perspectiva orientada a objetos*. México: ALFAOMEGA.
3. Fontela, C. (2011) *UML Modelado de Software para Profesionales*. 1ª. Edición, Buenos Aires: Alfa-Omega Grupo Editor.
4. Fowler M. (2000). *UML Gota a Gota*. 1ª. Edición. México: PEARSON.
5. Hansen G.W., Hansen J.V. 2002 *Diseño y administración de Bases de datos*: Prentice Hall.
6. Kendall E. K. (2011). *Análisis y Diseño de sistemas*. 8ª. Edición. México: PEARSON..
7. Pressman, R.S. (2008). *Ingeniería del Software un enfoque práctico*. 6ª. Edición México: MC GRAW HILL.
8. Senn J.A. (1996). *Análisis y Diseño de Sistemas*. 2ª Edición. México: MC GRAW HILL.
9. Silberschatz, A.; Korth, H. F., Sudarshan, S. 2002 *Fundamentos de Bases de Datos*. 5ª ed. Ed. McGraw Hill.
10. Sommerville, Ian (2001). *Ingeniería de Software*. México: PRENTICE HALL.

Electrónicas:

11. Unam. (Sin fecha). Panorama de los diferentes modelos de ciclo de vida de los sistemas. Consultado en Octubre 2012. Disponible en: http://uxmcc1.iimas.unam.mx/~cursos/Objetos/clases3_4.html.

Ingeniería de software

TABLA DE CONTENIDO

1.1 Revisión de especificación de requisitos.....	142
1.1.1 Norma IEEE 830.....	145
1.1.2 Trazabilidad de requisitos	153
1.2 Descripción de procesos actuales.....	153
1.3 Diagramas UML	154
1.4 Estudio de Factibilidad	163
1.5 Análisis costo-beneficio	166
2.1 Diseño de procesos propuestos	171
2.1.1 Herramientas CASE para el diseño	172
2.2 Diseño arquitectónico	174
2.3 Diseño de datos	179
2.4 Diseño de interfaz de usuario.....	183
3.1 Lenguajes para el desarrollo de software	190
3.2 Manejadores de bases de datos.....	192
3.3 Construcción del sistema	203
4.1 Diseño de caso de prueba	213
4.2 Pruebas de componentes	214
4.3 Pruebas del sistema.....	215
4.4 Documentación de resultados de las pruebas	230
4.5 Entrega del sistema y capacitación de usuarios	230
4.6 Entrega de documentación técnica y de usuario del sistema	230

Tema 1: Análisis

El núcleo de las actividades llevadas a cabo por muchas empresas está soportado por sistemas de información almacenados en computadoras. Las capacidades y características de sistemas de información bien diseñados e integrados a las políticas y necesidades de la compañía proporcionan una ventaja competitiva para las empresas que desean incursionar en nuevos mercados, trazar líneas de acción para dar respuesta a la competencia o para mejorar la forma en que se llevan a cabo las actividades cotidianas del negocio.

Cuando se desarrolla un nuevo sistema, éste debe integrarse a las actividades de la empresa: “Para diseñar y usar sistemas de información de manera eficaz, primeramente, es necesario entender el entorno, la estructura, la función y las políticas de las instituciones, así como el papel de la administración y la toma de decisiones en ésta” (Análisis y diseño de sistemas de información, 2024). También se debe conocer la tecnología computacional con la que cuenta la empresa.

El tema 1 de Análisis de sistemas enfatiza la importancia de que el ingeniero de software se involucre en el entorno y que exista empatía para comprender las necesidades y problemas que se presentan en la empresa. Sólo bajo estas condiciones, el ingeniero de software cuenta con los elementos necesarios para definir los requisitos de un sistema de información.

Competencia específica: Abstrae información del usuario final para elaborar el análisis de requerimientos del software a desarrollar.

Introducción

La calidad en el software está relacionada con la percepción del cliente de cómo trabaja su sistema. Uno de los métodos que para monitorizar y controlar la calidad durante el proceso de desarrollo del software es la inspección, revisar el cumplimiento de los requerimientos del cliente, los cuales han sido especificados desde el inicio del proyecto. La estrecha comunicación con el cliente ayuda a los ingenieros de software a enfocarse en las partes importantes del sistema en desarrollo desde el punto de vista del usuario.

El equipo de desarrollo de software debe conocer lo que hace el cliente en su día a día, cuáles son sus intereses, sus objetivos y su ambiente de trabajo. Esto le permite al desarrollador tomar mejores decisiones durante el proceso de desarrollo de software.

Con base en la información proporcionada por el cliente, el ingeniero de software debe identificar los casos de uso o historias de usuario que serán implementados por el sistema. Un caso de uso es una actividad o acción llevada a cabo por el sistema de software. Los casos de uso deben ser priorizados de acuerdo con lo que el usuario considera más valioso para su organización. Los casos de uso o historias de usuario se colocan en una lista organizados por prioridad, el orden de los casos de uso refleja lo que un usuario (o grupo de usuarios) piensa que es lo más importante para su sistema.

1.1 Revisión de especificación de requisitos

Una de las primeras tareas a realizar por el ingeniero de software es indagar o investigar los requisitos del cliente. La especificación de requisitos de software (Software Requirements Specification, SRS por sus siglas en inglés, o ERS por sus siglas en español) es un formato que muestra una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso que describen todas las interacciones que tendrán los usuarios con el software. La ERS también contiene requisitos no funcionales, comúnmente también definidos como restricciones del sistema, que son propiedades o características que debe de cumplir el software en desarrollo.

Antes de iniciar el desarrollo del sistema, el ingeniero de software debe confirmar las funcionalidades y características requeridas para el sistema. Es decir, el ingeniero de software debe tener la certeza de que la especificación de requisitos de software para el desarrollo de un sistema cumple con las necesidades y expectativas del cliente.

Ingeniería de requisitos

Es una estrategia de la Ingeniería de software para comprender lo que el cliente quiere, analizando sus necesidades, confirmando su viabilidad, negociando una solución razonable, especificando la solución sin ambigüedad, validando y gestionando los requisitos para que se transformen en un sistema operacional.

En la etapa de ingeniería de requisitos se pretende reunir la información necesaria para comprender la naturaleza de los programas a construirse. En esta etapa del desarrollo de software es en donde se recopilan los requisitos que debe cumplir el sistema a desarrollar. La fase de recopilación de requisitos es una etapa particularmente crítica en el proceso de desarrollo de software, ya que los errores en esta etapa conducen inevitablemente a problemas en las etapas siguientes de diseño e implementación.

La ingeniería de requisitos es el conjunto de actividades y tareas del proceso de desarrollo de sistemas que tiene como objetivos:

- Definir con la mejor calidad posible, las características de un sistema de software que satisfaga las necesidades del negocio de clientes y usuarios, además de integrarse con éxito en el entorno de trabajo.
- Gestionar las peticiones de cambio que se presenten en la especificación de requisitos, manteniendo la trazabilidad entre los requisitos y los productos en desarrollo.

El producto final de la Ingeniería de requisitos es un documento que contiene los requisitos del sistema obtenido por el ingeniero. El documento de especificación de requerimientos de software describe las funciones y propiedades de un sistema de software y las restricciones que gobiernan su desarrollo.

Pasos del proceso de la Ingeniería de requisitos:

- a) Identificación de requisitos
- b) Análisis y negociación de requisitos
- c) Especificación de requisitos
- d) Modelado del sistema
- e) Validación de requisitos
- f) Gestión de requisitos

a) Identificación de requisitos

En esta etapa, existe mucha cercanía entre el usuario y el Ingeniero de software, pues ambos comparten información indispensable para el desarrollo del software. El ingeniero de software indaga con el cliente, con los usuarios y con los involucrados cuáles serán los objetivos y requisitos del sistema. También se debe investigar cómo se ajustará el nuevo sistema a las necesidades de la empresa, y cómo dicho sistema se pretende utilizar día con día en las operaciones del negocio.

Para llevar a cabo esta investigación se utilizan herramientas como entrevistas, cuestionarios, observación directa, revisión de datos históricos, reuniones JAD (Joint Application Design), lluvia de ideas, escenarios, historias de usuario y si existe un documento de declaración de requisitos también se utiliza.

La identificación de requisitos es una de las etapas que requiere más cuidado al realizarlas debido a que en muchas ocasiones los requisitos no son fáciles de identificar, aún para los mismos clientes. Existen algunos problemas que se asocian con la identificación de los requisitos:

- Problemas de alcance. En ocasiones, los límites del sistema no están bien definidos, no se tiene la seguridad de qué funcionalidades debe incluir el sistema o los detalles técnicos innecesarios confunden.
- Problemas de comprensión. El cliente o el usuario no está seguro de lo que necesita, o ignora las características y funcionalidades que un sistema puede realizar.
- Problemas de volatilidad. Los requisitos cambian con el tiempo.

b) Análisis y negociación de requisitos

Cuando se inicia la actividad de análisis y negociación de requisitos, es necesario plantearse las siguientes preguntas:

- ✓ ¿Cada uno de los requisitos planteados por el cliente es consistente con los objetivos generales del sistema?
- ✓ ¿Cada uno de los requisitos están bien definidos, bien delimitados y sin ambigüedad?
- ✓ ¿Cada uno de los requisitos tiene referencia?, es decir, ¿Cada uno de los requisitos ha sido solicitado de forma explícita por el cliente?
- ✓ ¿Existe algún requisito o algunos requisitos que son incompatibles con otros requisitos? Es decir, ¿Algún requisito se contradice con otro requisito requerido?
- ✓ ¿Se pueden probar los requisitos una vez implementados para asegurar que cumplen su propósito?

Estos cuestionamientos ayudan al Ingeniero de software a limitar el alcance del problema. Las preguntas deben realizarse por cada uno de los requisitos del sistema, es común que los clientes y usuarios soliciten más de lo que puede realizar el sistema, por ello debe acotarse a las características factibles. También es común que los clientes propongan requisitos que son contradictorios entre sí, argumentando que son requisitos indispensables.

c) Especificación de requisitos

La especificación de requisitos es el producto final sobre los requisitos del sistema obtenido por el ingeniero. Describe las funciones y características de un sistema de computación y las

restricciones que los gobiernan. Los requisitos del sistema se integran en un formato de ERS, el cual contiene la información necesaria para la construcción del sistema.

Existen varios lineamientos que pueden ser utilizados para documentar los requisitos del sistema, uno de los más comunes y ampliamente utilizados es el estándar IEEE 830. La norma o estándar IEEE 830 ayuda a concentrar y organizar la información importante del sistema a construir.

d) Modelado del sistema

El modelado del sistema muestra en forma gráfica las características del sistema a desarrollar. Estas imágenes gráficas mejoran la comprensión de las características requeridas del sistema, constituyen un mecanismo que complementa la especificación del sistema. Para el modelado del sistema, una de las estrategias más utilizadas es el modelado UML.

e) Validación de requisitos

Examina el documento de especificación de requisitos para asegurar que todos los requisitos y restricciones del sistema se han definido sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, proyecto y producto de software.

f) Gestión de requisitos

La gestión de los requisitos es un conjunto de actividades que ayudan al equipo de trabajo a identificar, controlar y dar seguimiento a los requisitos y los cambios que se puedan suscitar en cualquier momento. Los cambios durante el proceso de desarrollo de software son normales, pero el ingeniero de software debe tener cuidado de utilizar un mecanismo para controlar dichos cambios.

La especificación de requisitos

La especificación de requisitos del software (ERS, o Software Requirements Specification, SRS), es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso, los cuales describen todas las interacciones que tendrán los usuarios con el software. Los casos de uso también se conocen como requisitos funcionales o historias de usuario.

1.1.1 Norma IEEE 830

El estándar IEEE 830 es un documento que describe el contenido y la calidad que debe tener una buena especificación de requisitos del software. Además, propone una forma de organización de la información de manera que sea comprensible tanto a los usuarios como a los desarrolladores del sistema.

El estándar IEEE 830 debe:

- Ayudar a los clientes a describir claramente lo que desea obtener de un software.
- Ayudar a los desarrolladores a entender qué quiere o necesita el cliente.
- Servir de base para desarrollar estándares de ERS particulares a cada organización.

Para los clientes, proveedores y otros participantes en el proyecto de desarrollo de software, una buena especificación de requisitos de software debe proporcionar algunos de los siguientes beneficios específicos:

- Establecer una base para acuerdos entre los clientes y los proveedores sobre lo que el producto a desarrollar debe realizar. La descripción completa de las funciones a ser realizadas por el software especificado en el documento SRS ayudará a los usuarios potenciales a determinar si el software especificado cubre sus necesidades o cómo el software debe ser modificado para satisfacer sus necesidades.
- Reducir el esfuerzo de desarrollo. La preparación de una buena ERS fuerza a los grupos involucrados a considerar rigurosamente todos los requerimientos del software antes de que el diseño inicie, por lo que reduce tiempo posterior de rediseño debido a malentendido o interpretación errónea de los requisitos durante la fase de recopilación de la información. La identificación temprana de este tipo de situaciones reduce el tiempo de desarrollo.
- Proporcionar una base para la estimación de costos y la calendarización de la programación de tareas. La descripción de un producto a ser desarrollado es una base para estimar los costos de un proyecto y asimismo puede ser utilizada para la programación de actividades.
- Proporcionar una base para la validación y verificación del software. Las organizaciones pueden desarrollar sus planes de validación y verificación de forma más eficiente a partir de una buena ERS.
- Facilitar la transferencia del sistema. El documento de ERS facilita la transferencia de un producto de software a nuevos usuarios o nuevas máquinas. Los clientes encuentran más fácil transferir el software a otras partes de su organización y los desarrolladores encuentran más fácil transferirlo a otros clientes.
- Servir como base para el mejoramiento. Una vez que el software ha sido terminado de desarrollar (o aún durante su desarrollo), el documento de ERS puede ser alterado o mejorado para ser utilizado en posteriores desarrollos.

De acuerdo con la Norma IEEE 830, un documento de especificación de requerimientos se organiza en 4 capítulos: Introducción, Descripción general, Requerimientos específicos y Apéndices.

Documento de especificación de requisitos de software conforme al estándar IEEE-830

Este documento de IEEE fue desarrollado por miembros voluntarios del Comité de Coordinación de Estándares y es un consenso de opiniones de una amplia experiencia en la materia recopilada por un grupo de expertos participantes quienes han expresado un interés en participar en la elaboración del documento.

El documento de especificación de requisitos del software se organiza en 4 capítulos:

1. Introducción

2. Descripción general

3. Requerimientos específicos

4. Apéndices

La utilización de un estándar durante el proceso de desarrollo de un software es muy recomendable, pues funciona como una guía de trabajo. Además del estándar IEEE 830 existen otras formas de producir, probar y medir el software. Es importante enfatizar que el estándar IEEE 830 representa un consenso de los intereses que conciernen a los participantes en su elaboración. Cualquier interpretación en la Norma debe analizarse y en todo caso adaptarse a los intereses particulares del proyecto en desarrollo.

- 1. Introducción
 - 1.1. Propósito
 - 1.2. Ámbito del sistema
 - 1.3. Definiciones, acrónimos y abreviaturas
 - 1.4. Referencias
 - 1.5. Visión general del documento

La introducción brinda información general sobre la Especificación de Requerimientos de Software.

Propósito. En esta sección se define de forma clara, precisa y concisa cuál es el propósito de este documento, así como a quién se dirige.

Ámbito del sistema. En esta sección se define el alcance del sistema, describe las características del producto de software a desarrollar, lo que hace y lo que no hace, así como los beneficios, objetivos y metas.

Definiciones, acrónimos y abreviaturas. En esta sección se definen todos los términos, acrónimos y abreviaciones que se requieren para interpretar apropiadamente el documento de especificación de requisitos de software.

Referencias. Esta sección debe incluir una lista completa de todos los documentos referenciados en cualquier parte de la ERS, identificando cada documento por su título, fecha, organización que publica; así también se deben especificar las fuentes de las que se obtuvo información. La información puede incluirse por medio de una referencia a un apéndice o en otro documento aparte.

Visión general del documento. Describe la forma en que se organiza la información en el documento de ERS, así como una breve descripción de su contenido.

- 2. Descripción general
 - 2.1. Perspectiva del producto
 - 2.2. Funciones del producto
 - 2.3. Características de los usuarios
 - 2.4. Restricciones
 - 2.5. Suposiciones y dependencias
 - 2.6. Requisitos futuros

En la descripción general, se describen en forma sucinta aspectos generales que afectan el producto y sus requisitos. En esta sección no se determinan requisitos específicos, sino que proporciona antecedentes para dichos requisitos.

Perspectiva del producto. Describe la relación del sistema respecto a otros sistemas, debe mostrar cuál es el rol que juega el sistema dentro de la organización y cómo se integra a los demás sistemas de la empresa (si el sistema funciona en forma independiente, o es un módulo de un sistema más grande o funciona interactuando con otro sistema). Si se requiere la interacción con otro sistema o es parte de otro sistema, entonces se deben describir los requisitos que se deben cumplir para llevar a cabo dicha interacción y los requisitos de la interfaz de comunicación entre sistemas. En esta sección, se puede utilizar un diagrama de bloques para mostrar los principales componentes del sistema, así como sus interacciones.

Funciones del producto. Proporciona un resumen de las principales funcionalidades que el software podrá realizar. Este resumen de funciones del sistema puede ser tomada de la

sección de especificación de alto nivel, en caso de que exista. Para dar claridad a la lectura, las funciones deben ser organizadas de manera que sea comprensible para el cliente y cualquier persona involucrada con el proyecto. Los métodos gráficos y textuales pueden ser utilizados para mostrar las diferentes funciones y sus relaciones. Tal diagrama no pretende mostrar el diseño del producto, simplemente muestra la relación lógica entre los elementos.

Características de los usuarios. Muestra las características generales de los usuarios prospectos del producto incluyendo su nivel educativo, experiencia, experiencia técnica, edad si es necesario, etc.

Restricciones. Se describen las limitaciones que se imponen en el desarrollo del producto. Muestra características de cualquier elemento que limitará las opciones de los desarrolladores, incluyendo políticas de la organización, políticas regulatorias, limitaciones de hardware, interfaces con otros sistemas, operaciones en paralelo, funciones de auditoría, funciones de control, requisitos lingüísticos, protocolos de comunicación, requisitos de confiabilidad, puntos críticos de una aplicación, consideraciones de seguridad.

Suposiciones y dependencias. En esta sección se listan cada uno de los factores que afectan los requisitos especificados en el documento de SRS. Describe situaciones que se presuponen ciertas en el momento de elaborar el documento de SRS y que, si cambian, implica cambios en los requerimientos, por ejemplo, la utilización de una tecnología específica, la vigencia de normas, las características de otros sistemas con los que debe interactuar el sistema en desarrollo, etc.

Requisitos futuros. Esta sección contiene sugerencias y recomendaciones que no son incluidas en esta versión del software a desarrollar, pero que son mejoras futuras al sistema y que podrán analizarse e implementarse en un futuro.

- 3. Requisitos específicos
 - 3.1. Interfaces externas
 - 3.2. Funciones
 - 3.3. Requisitos de rendimiento
 - 3.4. Restricciones de diseño
 - 3.5. Atributos del sistema
 - 3.6. Otros requisitos

Como su nombre lo indica, la sección de requisitos específicos describe en forma detallada y precisa los requisitos que el software debe cumplir. Los requisitos contenidos en esta sección deben describirse a un nivel de detalle suficiente para permitir que:

- Los diseñadores puedan esbozar un sistema que cumpla con los requisitos establecidos de manera eficiente.
- Los desarrolladores puedan implementar una solución con base en las características del diseño.
- El equipo de pruebas pueda planificar y diseñar un conjunto de casos de prueba que demuestren si el sistema cumple con los requisitos especificados.
- El cliente pueda determinar si el sistema satisface sus necesidades.

Los requisitos especificados en esta sección describen el comportamiento externo del sistema, por lo que son perceptibles para cualquiera de los involucrados en el proyecto. Esta sección es la más extensa, debido al nivel de detalle requerido, también es la más importante, pues si se logra especificar cada uno de los requisitos con el suficiente nivel de detalle requerido, se facilitará el proceso de desarrollo en las siguientes fases de análisis, diseño, desarrollo. Para garantizar el éxito del cumplimiento de los objetivos del producto se debe tener cuidado de aplicar los siguientes principios:

- El documento debe ser legible por cualquiera de las personas involucradas en el proyecto, sin importar sus intereses o formación profesional.
- Debe existir una referencia a todo documento relevante que influya en los requisitos del sistema.
- Cada uno de los requisitos debe ser identificado mediante algún código o sistema de numeración apropiado.

Interfaces externas. Describe las interfaces de usuario, así como las interfaces de comunicación con otros sistemas de hardware o software. Las interfaces externas incluyen una descripción detallada de todas las entradas al sistema y salidas proporcionadas por el sistema. Cada una de las interfaces externas debe incluir tanto el formato como el contenido como sigue: nombre de la interfaz de entrada o salida, propósito, fuente de entrada de datos y destino de la salida de información, rangos válidos de valores, precisión y tolerancia, unidades de medición, relación con otras entradas y salidas, formatos de pantalla, y organización del sistema,

Funciones. Esta sección también se conoce como sección de requisitos funcionales y se describe en forma detallada cada una de las acciones o funciones que desarrolla el sistema, define cómo se aceptan y procesan las entradas y cómo se procesan y generan las salidas.

Generalmente es un listado con enunciados que inician con: “El sistema deberá...”. La descripción de los requisitos funcionales incluye:

- La validación de datos de entrada
- Definir la secuencia de las operaciones
- Respuestas a situaciones anormales como
 - Desbordamiento de datos (overflow)
 - Problemas de comunicación entre módulos
 - Manejo y recuperación de errores
- Efectos de envío de parámetros
- Relación entre las entradas y salidas de datos, incluyendo la secuencia entrada/salida y las fórmulas para conversión de entradas en salidas.

Podría ser apropiado particionar los requisitos funcionales en subprocesos (esto no implica que el diseño de software sea particionado de la misma manera). Cuando la lista de requisitos funcionales es muy larga, se recomienda particionar u organizar los requisitos. Los requisitos pueden ser organizados tomando en cuenta las siguientes consideraciones, aunque se pudiera elegir cualquier otra forma de organización que resulte más apropiada:

- Por tipos de usuarios. Los diferentes usuarios del sistema tienen acceso a un conjunto específico de requisitos, de acuerdo con su perfil. Para cada tipo de usuario se pueden agrupar los requisitos funcionales que afectan o tengan mayor relación con sus actividades de trabajo.
- Por objetos. En este caso, por objeto se refiere a entidades del mundo real que se manejan en el sistema, no se refiere a que el diseño del sistema siga el paradigma Orientado a Objetos. Para cada objeto, se definen sus atributos y funciones, y si es conveniente se agrupan en clases.
- Por objetivos. Los objetivos se refieren a un servicio que el sistema deberá ofrecer y que requiere una entrada para procesar la salida. Para cada objetivo del sistema se detallarán las funciones que permiten que logre.
- Por estímulos. En este caso, se deben especificar los estímulos que recibe el sistema y el conjunto de funciones relacionadas con dicho estímulo.
- Por jerarquía funcional. Otra alternativa adicional que resulta útil al momento de organizar las funciones es la especificación de una jerarquía de funciones que comparten entradas, salidas y datos internos. Se describen las funciones y las subfunciones de acuerdo con su entrada, proceso y salida. Sin embargo, esto no significa que el diseño del sistema se deba realizar con el paradigma de diseño estructurado.

Requisitos de desempeño. Los requisitos de desempeño o de rendimiento están relacionados con la carga máxima que se espera que el sistema soporte. Los requisitos de rendimiento especifican la cantidad de usuarios que usan el sistema simultáneamente, cantidad de operaciones o transacciones que se realizan por unidad de tiempo, cantidad y tipo de datos que se pueden almacenar, capacidad del sistema para permitir el acceso a la información, de acuerdo con los perfiles de usuario válidos.

Restricciones de diseño. En esta sección se especifican las condiciones que deben cumplirse durante la etapa de diseño del sistema, todo aquello que limite las decisiones relativas al diseño de la aplicación, por ejemplo, la aplicación de estándares, limitaciones de hardware, limitaciones del entorno de operación donde se instalará el sistema.

En esta sección el cumplimiento de estándares toma especial relevancia. Se especifican los requerimientos derivados de las regulaciones o estándares existentes. Estos requisitos pueden incluir: formatos específicos para los reportes, denominación de variables, procedimientos contables específicos, seguimiento de auditoría.

Como parte de las restricciones de diseño se podría necesitar un software para el seguimiento de las actividades de procesamiento, y se debe especificar, su nombre, el código mnemotécnico, el número de la especificación y número de versión del software.

Atributos del sistema. Todo sistema debe contar con atributos de calidad como fiabilidad, mantenibilidad, portabilidad y seguridad. A estos atributos también se les conoce como requisitos no funcionales. En esta sección se especifican qué tipos de usuario están autorizados o no para realizar tareas específicas, o cómo se implementarán mecanismos de seguridad como por ejemplo control de acceso al sistema como la utilización de login y password.

Otros requisitos. En esta sección se especifican todos los requisitos que no se han incluido en las categorías mencionadas, pero que son importantes para el proceso de desarrollo de software. Generalmente se asocian a situaciones particulares del contexto, son características propias del sistema a desarrollar o de la empresa para la cual se desarrolla el software.

4. Apéndices

El capítulo 4 contiene información adicional y relevante que no forma parte de las secciones anteriores. Es información de soporte que hace más fácil de leer, entender y usar el documento de especificación de requisitos (SRS). Puede incluir:

- Tabla de contenido
- Índices
- Apéndices
- Formatos de entrada/salida
- Análisis de costos
- Información de licencias
- Información relacionada con lenguajes de programación, etc.

1.1.2 Trazabilidad de requisitos

Una vez que han sido especificados, revisados y plasmados en un documento los requisitos del proyecto, se debe realizar un plan de trabajo que debe contener un cronograma de actividades donde se muestren las fechas de inicio y terminación del proyecto, así como un conjunto de actividades bien definidas que al realizarlas vayan encaminando el proyecto hacia su culminación. Es necesario dar seguimiento al proceso de desarrollo de software, para asegurar el avance en la construcción del sistema.

El seguimiento de un proyecto (o trazabilidad de los requisitos) es un mecanismo que permite separar y especificar correctamente los requisitos para controlar su evolución y soportar los cambios que se presenten en el camino. La trazabilidad de los requisitos asegura la continua concordancia entre los requisitos especificados en el documento de SRS y los artefactos producidos durante el proceso de desarrollo de software. Los requisitos del sistema son susceptibles de cambios, no sólo después de la entrega del producto sino también durante el proceso de desarrollo. Para que la trazabilidad de los requisitos sea efectiva, es imprescindible que las relaciones entre los requisitos y demás especificaciones elaboradas en la etapa de Ingeniería de requisitos estén adecuadamente definidas. La trazabilidad de requisitos es un mecanismo que permite mantener el control de los requisitos, es la habilidad para describir y seguir la vida de un requisito en ambos sentidos: desde el estado actual del sistema en desarrollo hacia sus orígenes, o desde la solicitud del requisito hacia su implementación.

1.2 Descripción de procesos actuales

Los sistemas de software vienen a complementar las tareas y actividades que se llevan a cabo en una empresa, no es un ente aislado que se incrusta de manera independiente en las actividades de la empresa. Un nuevo sistema de software en la empresa permite automatizar algunas partes de los procesos, mejorando la eficiencia en las actividades cotidianas y la productividad de los empleados.

Un sistema que no aporta a una empresa y por el contrario representa un problema significa que es necesario replantear su existencia o tal vez adaptarlo a nuevas necesidades que surjan en la misma. Defender un proyecto, participar en su desarrollo y entregar un producto que se adapte a las necesidades de la empresa debe ser la meta de un equipo de desarrollo de software.

El ingeniero de software debe tener una mente abierta y empatía al interactuar con el cliente, ser consciente que el sistema que se va a producir es para que el cliente mejore la forma en que se llevan a cabo las actividades en la empresa. En el proceso de desarrollo de software se requieren recursos humanos capaces de identificar áreas de negocios que puedan ser automatizadas o que puedan mejorar su procesamiento utilizando modernos sistemas de software que cubran las necesidades de los clientes.

1.3 Diagramas UML

Como se mencionó en el subtema 1.1, en los primeros tres pasos del proceso de Ingeniería de requisitos, los requisitos deben ser identificados, analizados y especificados. Y una vez que los requisitos del sistema han sido especificados, el siguiente paso es obtener un modelo de los requisitos del sistema.

Un modelo conceptual es la representación de un objeto del mundo real, es un modelo formal que a menudo se plasma en un modelo visual. Para realizar un modelo se emplea una notación que permita comunicar fácilmente las ideas del autor del modelo a otras personas distintas. Las notaciones que permiten representar estos modelos emplean un lenguaje y un conjunto de reglas que establecen cómo se utilizan los elementos del lenguaje al momento de construir los modelos.

Existen muchas notaciones y especificaciones de diseño para realizar modelos en el desarrollo de software, pero una de las notaciones más utilizadas de manera universal es el lenguaje UML (Unified Modeling Language, Lenguaje de modelado unificado).

El lenguaje de modelado unificado (UML), es un lenguaje para el modelado de sistemas de software. UML es un lenguaje gráfico para visualizar, analizar, diseñar, especificar, construir y documentar un sistema de software. Es uno de los más conocidos en la actualidad, y es un lenguaje estandarizado para describir un modelo del sistema. La utilización de UML en el desarrollo de software permite describir diferentes aspectos del sistema a desarrollar, al aplicarlo en este proceso permite obtener diferentes diagramas y dar soporte a una metodología de desarrollo de software.

A través de un modelo, se pueden recopilar enunciados de interés para un sistema en particular, realizando una abstracción de todos los detalles del sistema que pudieran ser descritos desde un punto de vista en particular y para un propósito específico.

El modelado UML permite representar diferentes tipos de diagramas, por ejemplo:

- Diagramas de casos de uso
- Diagrama de clases
- Diagrama de secuencia
- Diagrama de estados

Cuando se tienen definidos los requisitos del sistema y se identifican los usuarios que interactúan con el sistema, se puede llevar a cabo el diagrama de casos de uso. Los casos de uso y los actores son los componentes más importantes para diseñar un diagrama de casos de uso. Algunos conceptos importantes para comprender mejor el modelado UML:

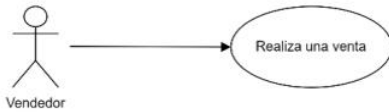
- Casos de uso. Un caso de uso expresa los requisitos del sistema, permite lograr una mejor comprensión del problema. Son escenarios en los cuales el usuario interactúa con el sistema que se está definiendo para lograr la comprensión de cierto objetivo. A partir de las entrevistas con los clientes y usuarios involucrados, se intenta descubrir las diversas situaciones que se pueden presentar durante la utilización del sistema.
- Actor. Es algo o alguien que debe interactuar con el sistema en desarrollo. No tiene que ser una persona, pueden ser otros sistemas o hardware externo. Un actor representa una entidad externa al sistema, la cual interactúa con el software en desarrollo. Los actores interactúan con el sistema mediante mensajes. Los actores se conectan con los casos de uso mediante asociaciones que permiten la comunicación.
- Diagrama de casos de uso. Muestra los actores y casos de uso, así como las relaciones que existen entre ellos (no todos los actores se relacionan con todos los casos de uso). También existen relaciones entre los casos de uso, las dos relaciones principales son de uso (include) y de extensión (extend).

Diseño de un diagrama de casos de uso

Los diagramas de casos de uso muestra a los actores y los casos de uso, así como las relaciones que existen entre ellos, en un diagrama de casos de uso no todos los actores se relacionan con todos los casos de uso.

Los actores se representan con una figura de humano y los casos de uso por un óvalo. Las relaciones entre un actor y un caso de uso se representan por una flecha. Los actores, casos de uso y las relaciones entre los mismos se muestran en la figura 1.1.

Diagrama de caso de uso del requisito: Realizar una venta



En el diagrama, el actor se representa con una figura humana y el caso de uso se representa con un óvalo. Se utiliza una flecha para representar qué actor es el que lleva a cabo el caso de uso.

En este diagrama de casos de uso, el vendedor (actor) se encarga de realizar una venta.

Figura 1.1 Diagrama de caso de uso

En un diagrama de casos de uso existen diferentes tipos de relación entre los casos de uso, entre los que destacan las relaciones de uso y de extensión (include y extend respectivamente). Las relaciones <<include>> y <<extend>> muestran gráficamente una relación de dependencia entre dos casos de uso, son relaciones que se utilizan para enlazar dos casos de uso. En la figura 1.2 se muestra un ejemplo de uso de las relaciones <<include>> y <<extend>>.

Una relación <<include>> muestra un comportamiento común a uno o más casos de uso. Indica que un caso de uso necesita el apoyo o el servicio de otro caso de uso para poder cumplir una tarea (la flecha de relación se etiqueta con <<include>>, use o sin etiqueta).

Una relación <<extend>> muestra un comportamiento que es opcional, es decir, indica que un caso de uso puede requerir o no la colaboración de otro caso de uso para cumplir su cometido.

Caso de uso 1

Un caso de uso asociado a otro caso de uso mediante una relación <<include>>.

Este tipo de relación muestra un caso de uso que requiere la colaboración de otro caso de uso para poder llevar a cabo su función.

a) Asociación <<include>> entre dos casos de uso



Cuando el actor vendedor realiza una venta, también debe realizar el cobro de la venta.

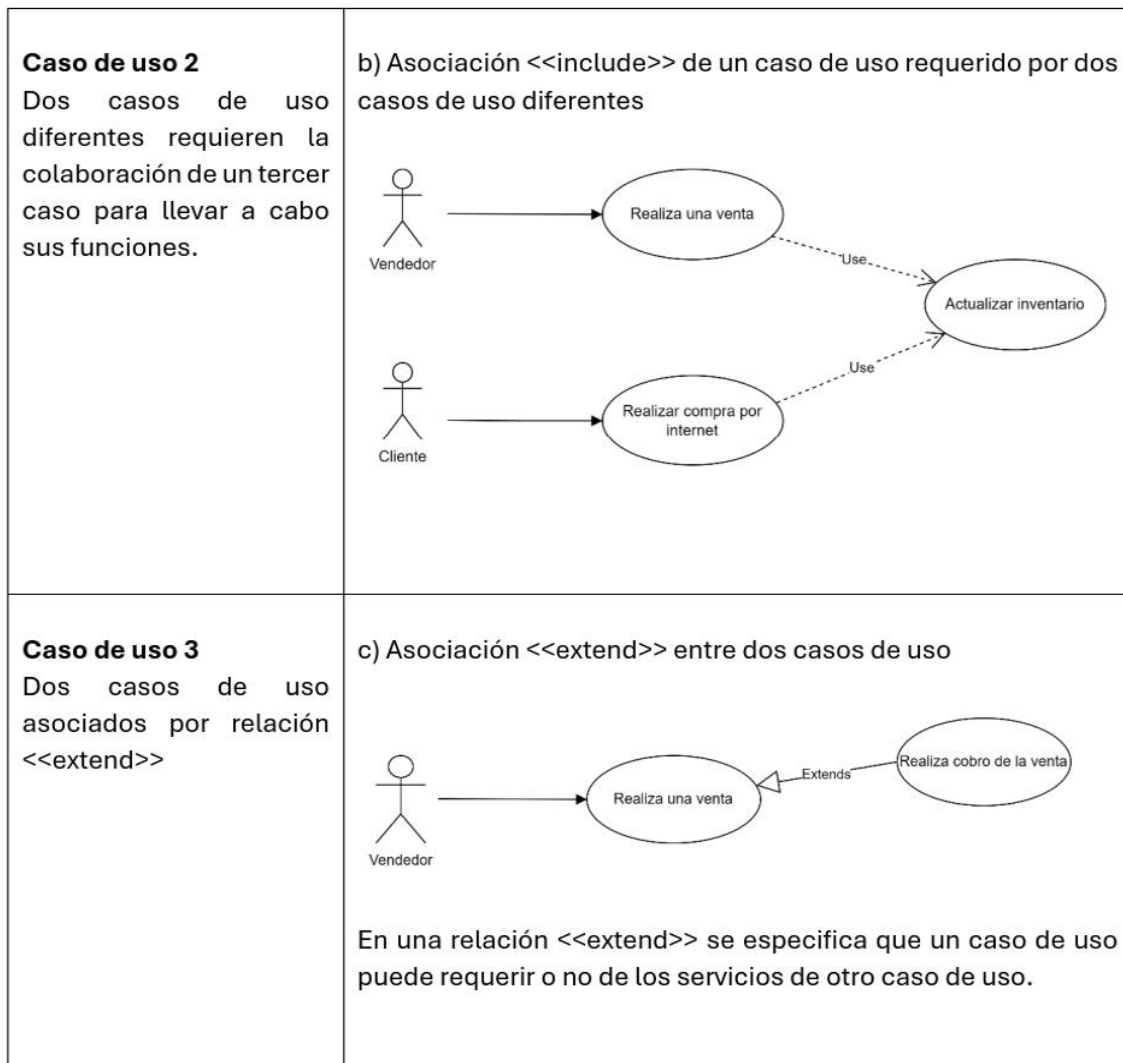


Figura 1.2 Ejemplo de relaciones <<include>> y <<extend>>

Diagrama de casos de uso

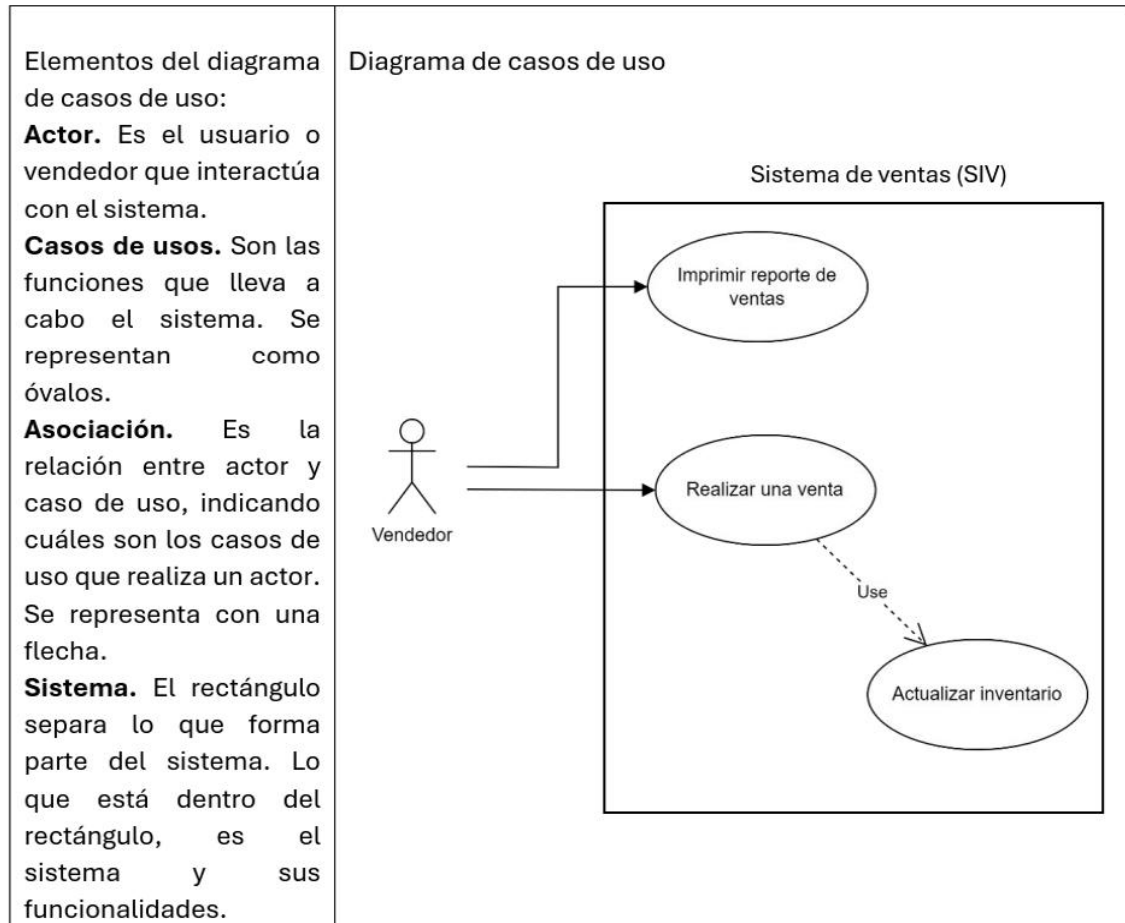


Figura 1.3 Ejemplo de un diagrama de casos de uso

Diagrama de clases

Un paso muy importante en el análisis del sistema es la identificación de las clases que formarán parte del sistema, así como los atributos y responsabilidades de cada clase. Se deben identificar las clases del dominio de la aplicación, por ejemplo, si el dominio de la aplicación es un banco, las posibles clases que podrían generarse son: cliente del banco, cajero automático, estado de cuenta, préstamo, etc. El dominio de la aplicación se refiere al contexto o el área donde se va a desarrollar el sistema. El dominio de la aplicación se puede tomar como punto de partida para iniciar el análisis del estado actual del problema. Es necesario analizar el contexto del problema o el área de oportunidad a mejorar y a partir de aquí empezar a crear un modelo del dominio.

Una vez identificado el dominio del problema, se inicia el diseño de un diagrama de clases, se empiezan a identificar las clases, una forma efectiva de lograrlo consiste en “enumerar y cortar”. La técnica de “enumerar y cortar” consiste en enumerar una lista de clases

candidatas en las que se pueda considerar que pudieran ser parte de las clases. Posteriormente se “cortan” de modo drástico las clases menos probables de formar parte del diagrama, es decir, se eliminan de la lista aquellas clases que no se consideren factibles, de esta forma quedan en la lista unas cuantas clases esenciales. Si existe duda en la utilidad de una clase candidata, optar por dejarla en la lista, posteriormente se pueden eliminar o agregar nuevas clases. Un diagrama de clases representa un conjunto de clases y las relaciones entre ellas. En el diagrama detallado de clases, se debe modelar cada entidad mediante la enumeración de sus características que son sus atributos y métodos (Braude, pp. 205).

Un diagrama de clases se compone por tres elementos principales: nombre de la clase, atributos de la clase y métodos de la clase. Los atributos representan las propiedades de un objeto de la clase, son características que identifican de manera única a la clase. Los métodos representan el comportamiento de los objetos o de la clase, y por lo tanto son funciones o transformaciones que se aplican a los objetos de una clase y que pueden invocarse.

La representación gráfica de una clase es un rectángulo dividido con dos líneas horizontales para crear 3 secciones (ver figura 1.4), en donde cada sección especifica el nombre de la clase, sus atributos y sus métodos.

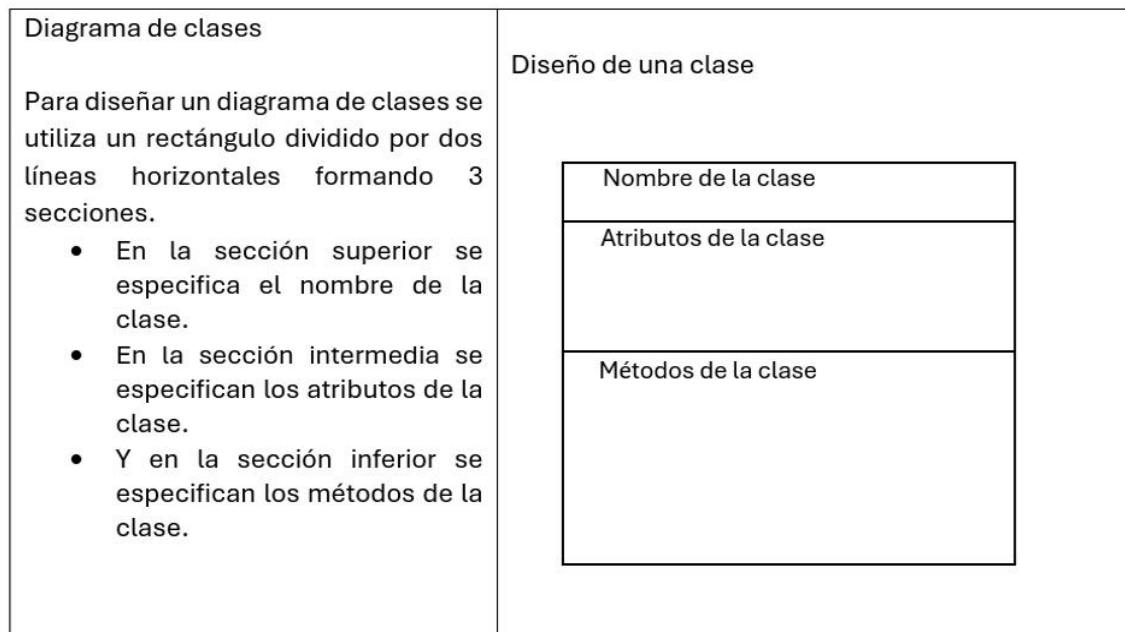


Figura 1.4 Diagrama de clases

Especificación de los elementos de una clase

El nombre de una clase no debe tener espacios en blanco ni caracteres especiales. Se debe iniciar cada palabra con mayúscula, por ejemplo: **ProductoEnAlmacen**. Los nombres de clases empiezan con letra mayúscula, y los nombres de objeto con letra minúscula. La palabra o palabras que forman parte del nombre de la clase deben ser congruentes con su significado.

Los atributos de una clase corresponden a sustantivos, que son propiedades que definen el objeto. Cada atributo puede tener un nombre, un tipo y un nivel de visibilidad (Pressman, p. 726). El tipo sigue al nombre y se separa de él mediante dos puntos. La visibilidad se indica mediante -, #, ~, +. Este símbolo precede al nombre de la clase e indica respectivamente visibilidad privada, protegida, paquete o pública.

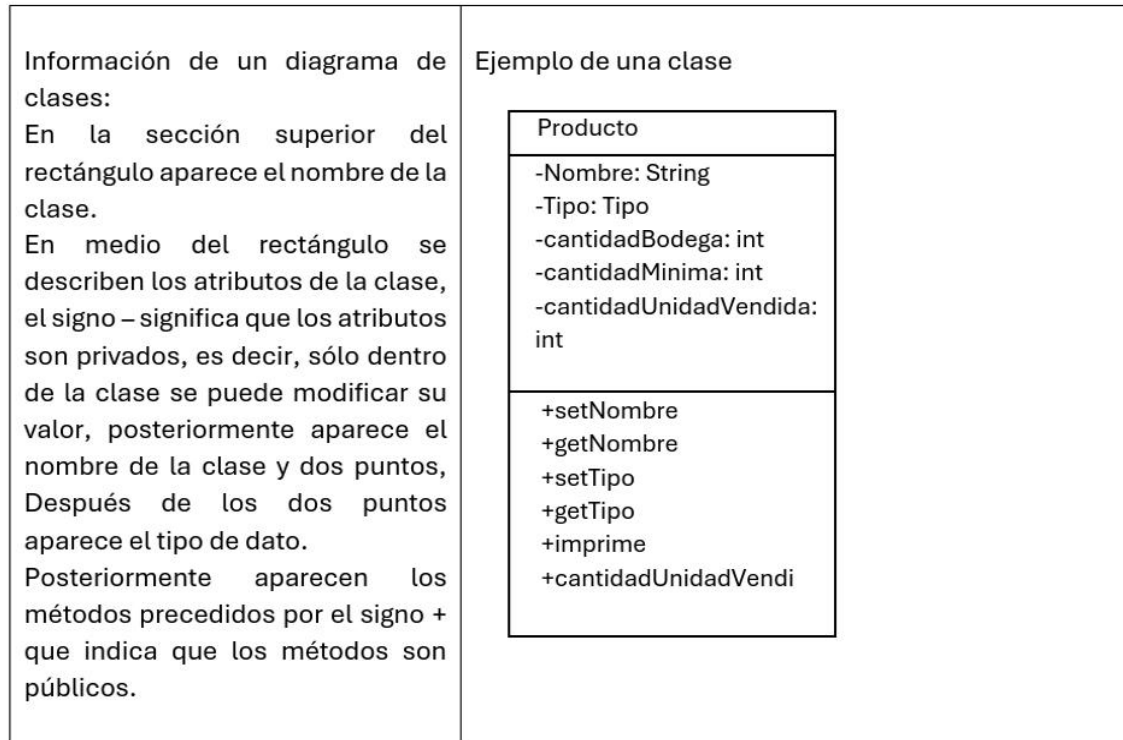


Figura 1.6 Ejemplo de diagrama de clases

Diagrama de secuencia

Los diagramas de secuencia en UML se utilizan para modelar interacciones entre los actores y los objetos involucrados en el sistema, así también las interacciones de los objetos entre sí. Por cada caso de uso que se haya identificado en el sistema, existe al menos un escenario principal que representa la secuencia de acciones más habitual (también se le llama historia de éxito del usuario o escenario). También es frecuente que existan además uno o más escenarios secundarios que representan secuencias de eventos alternativos. Estas secuencias de eventos se detallan mediante un conjunto de pasos numerados.

En UML un escenario es una descripción formal de los eventos o sucesos que ocurren cuando se lleva a cabo un caso de uso. Se puede decir que un escenario es una representación textual del diagrama de secuencia, que define la secuencia de eventos entre los actores y las funciones que lleva a cabo el sistema. Los escenarios son una técnica de modelado que permite describir cómo interactúa un usuario con una función del sistema para lograr su objetivo.

Para iniciar el diseño de un diagrama de secuencia, se inicia describiendo los escenarios como un conjunto de pasos numerados como se muestra a continuación.

Por ejemplo, para el caso de uso <<Pagar con tarjeta de crédito el importe de un tramo de autopista de peaje>> la secuencia de pasos en su escenario principal es:

- a. El sistema solicita la introducción de la tarjeta de crédito.
- b. El usuario introduce la tarjeta.
- c. El sistema solicita el cargo del importe a la institución bancaria.
- d. La institución bancaria confirma el cargo a la tarjeta.
- e. El sistema acepta la operación y expulsa la tarjeta.
- f. La interfaz del sistema indica al usuario que puede continuar su camino.

Los pasos definidos en el escenario sirven de base para diseñar el diagrama de secuencia. Al diagrama de secuencia también se le conoce como diagrama de interacción o evento. Un diagrama de secuencia describe los diferentes casos de uso según la interacción o eventos enviados entre los objetos de la arquitectura del modelo del análisis. Es una herramienta útil que permite encontrar el tipo de relación que existe entre las abstracciones de datos inicialmente identificadas. Muestra los mensajes que se intercambian las instancias de las clases en el tiempo.

De manera ideal, es conveniente realizar un diagrama de secuencia por cada caso de uso, aunque estrictamente no es necesario. Si el caso de uso es sencillo y no existen muchas interacciones entre los objetos del sistema, se puede optar por no realizar el diagrama. Los objetos que se ven en los diagramas de secuencia deben corresponder estrechamente con las operaciones disponibles para cada clase. Debido a que en la etapa de análisis se busca comprender a fondo el problema, se pueden manejar mensajes textuales en vez de los métodos de las clases.

Los diagramas de secuencia muestran los objetos que han participado en la interacción, así como la secuencia de mensajes intercambiados. Los diagramas de secuencia describen los diferentes casos de uso según la interacción o eventos enviados entre los objetos de la arquitectura del modelo del análisis. Describe aspectos dinámicos de un sistema. Cada objeto en el diagrama es representado por una línea vertical correspondiente al eje del tiempo donde el tiempo avanza hacia abajo conforme se llevan a cabo los eventos que ocurren en el tiempo, los cuales son enviados de un objeto a otro. Es importante tomar en cuenta el orden en que los eventos ocurren y la dependencia entre ellos, en otras palabras, qué consecuencias tiene el envío de un evento. Es importante que no existan interrupciones en las líneas que representan los eventos y que exista continuidad con los eventos externos del sistema.

Pasos para diseñar un diagrama de secuencia:

- Los objetos de las clases se colocan de izquierda a derecha en la parte superior del diagrama y se representan por un rectángulo.
- Los objetos se representan en la interacción por líneas discontinuas verticales.
- El tiempo se representa por líneas con punta de flecha horizontales, que avanzan de arriba hacia abajo.

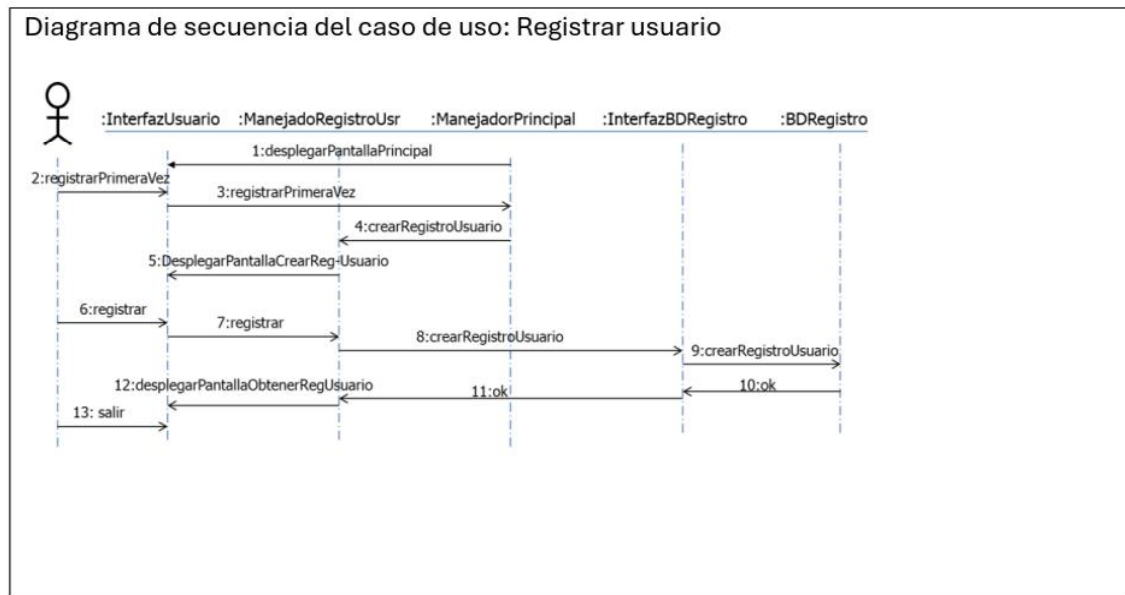


Figura 1.7 Ejemplo del diagrama de secuencia “Registrar usuario”

En la figura 1.7 se muestra el ejemplo del diagrama de secuencia “Registrar usuario”. En este ejemplo, se observan los objetos involucrados, InterfazUsuario, ManejadorRegistroUsr, ManejadorPrincipal, InterfazBDRegistro y BDRegistro. Estas clases se envían mensajes comenzando con el ManejadorPrincipal quien tiene la responsabilidad de enviar mensaje para que se despliegue la pantalla principal del sistema. Los mensajes son intercambiados entre las diferentes clases a partir de la opción seleccionada por el actor del caso de uso que solicita su registro en la base de datos.

La clase ManejadorPrincipal es la clase orquestadora que al recibir la solicitud del actor envía un mensaje a la clase ManejadorRegistroUsr para que solicite el despliegue de la interfaz para la captura de los datos de registro del usuario. Una vez desplegada la pantalla, y que el usuario captura su información, se envían nuevamente mensajes entre las clases hasta registrar los datos del usuario en la base de datos.

Como se puede observar en el diagrama de secuencia de la figura 1.7, cada flecha muestra el orden en que son enviados los diferentes mensajes entre los objetos. También se muestra cuál es la clase que inicia la interacción de los mensajes y cuál es la participación del actor.

La figura 1.8 muestra el diagrama de secuencia “Solicitar libro prestado”. La representación de un diagrama de secuencia puede variar dependiendo de la herramienta utilizada, aunque la esencia de su representación es la misma. En la parte superior del diagrama aparece el actor

y las clases involucradas en el escenario. Se presentan rectángulos alrededor de las clases, indicando que son objetos específicos. La interacción comienza en la parte superior del diagrama desplegando la interfaz principal, en la cual el usuario solicita el préstamo de un libro, proporcionando sus datos y los del libro solicitado. Los objetos involucrados son:

- Título, que contiene los títulos de los libros existentes en la base de datos.
- Información, en donde se almacena la información proporcionada por el usuario.
- Préstamo que contiene la información de los usuarios que han solicitado libro.
- Ejemplar, que contiene la información de todos los ejemplares de los libros con los que cuenta la base de datos.

En el diagrama de secuencia se verifica la existencia del título del libro solicitado, y si el título del libro existe, entonces se busca que exista un ejemplar de dicho título. Si existe un ejemplar, posteriormente se procede a verificar que exista el prestatario en la base de datos además no debe tener adeudos de libros, en cuyo caso se continúa el proceso para el préstamo del título solicitado.

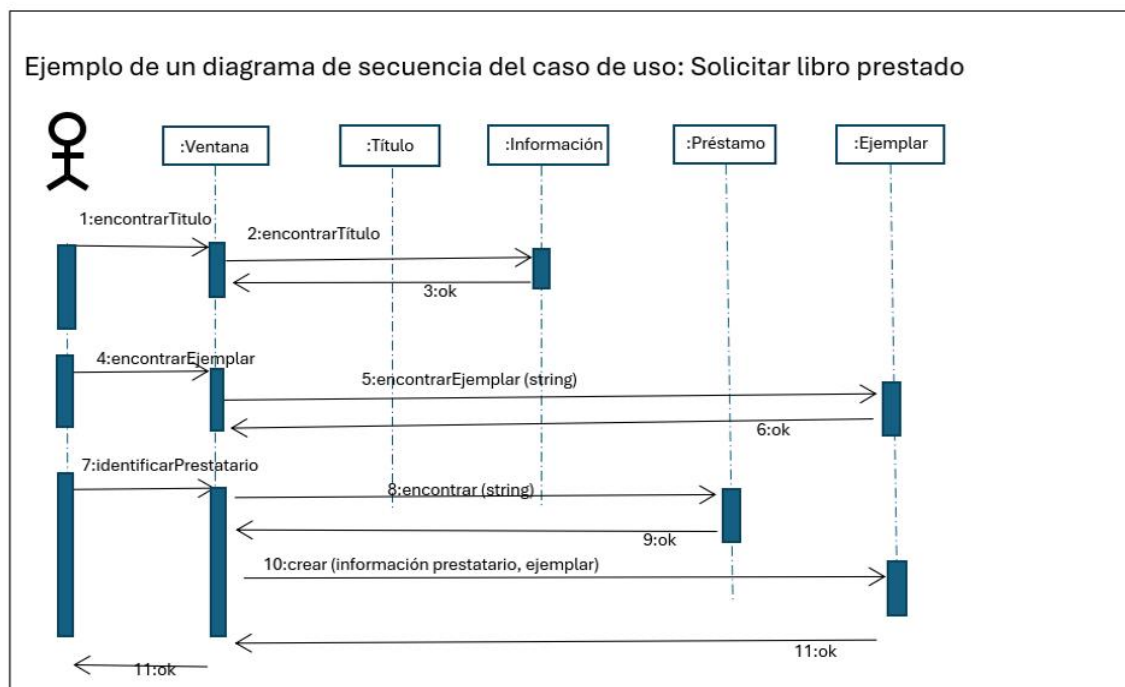


Figura 1.8 Diagrama de secuencia “Solicitar libro prestado”

1.4 Estudio de Factibilidad

La factibilidad se define como la disponibilidad de los recursos necesarios para llevar a cabo los objetivos o metas señaladas. Un estudio de factibilidad es un instrumento para orientar la toma de decisiones. Es una investigación mediante la cual se recopilan datos relevantes sobre el desarrollo de un proyecto para tomar la mejor decisión, si procede su estudio, desarrollo e implementación del proyecto, midiendo sus probabilidades de éxito o fracaso.

Un estudio de factibilidad consiste en descubrir cuáles son los objetivos de la organización, así como determinar si el producto de software que se desea construir es útil para que la empresa logre sus objetivos.

Existe un conjunto de objetivos comunes a todas las empresas que deben considerarse al realizar un estudio de factibilidad, algunos de los más importantes son:

- Reducción de errores y mayor precisión en los procesos.
- Reducción de costos mediante la optimización o eliminación de recursos no necesarios.
- Integración de las áreas y subsistemas de una empresa.
- Actualización y mejoramiento de los servicios a los clientes o usuarios.
- Aceleración en la recopilación de datos.
- Reducción de tiempo en el procesamiento y ejecución de tareas.
- Automatización óptima de procedimientos manuales.

Es necesario evaluar un proyecto antes de iniciar su desarrollo para no invertir tiempo, dinero y esfuerzo en un proyecto que no reditúa la inversión. Algunas razones por las cuales se debe evaluar un proyecto son (Sapag, p. 1):

- Hoy en día, la preparación y evaluación de proyectos cumple un papel muy importante entre agentes económicos responsables de decidir acerca de la asignación de recursos para implementar iniciativas de inversión.
- Existen instituciones financieras que se destacan por la incorporación de análisis profesionalizados de otorgamiento de créditos.
- Con esto se logra velar por los intereses de la institución otorgante de crédito y los intereses de los inversionistas, quienes deben responder con sus bienes dados en garantía por cualquier error en una mala decisión.
- Cuando se plantea un proyecto de desarrollo de software, una de las alternativas para captar recursos para el proyecto es optar por una de las posibilidades de solicitud de crédito, en esta opción debe tomarse en cuenta la función del resultado de un análisis que muestre una rentabilidad positiva que se espera de la inversión.

Cuando se evalúa la factibilidad de un proyecto se deben considerar todos los gastos y costos que se generan con el desarrollo del proyecto, para lograr un resultado adecuado es necesario un análisis detallado de los requisitos para el desarrollo de software.

Existen diferentes tipos de factibilidad, las más comunes que se aplican en el estudio de factibilidad de un proyecto de desarrollo de software son:

- a. Factibilidad técnica. Establece si existen los recursos técnicos necesarios para construir un proyecto de desarrollo de software. Evalúa si se cuenta con el equipo, software y capacidades técnicas requeridas para implementar cada alternativa de diseño que se esté planificando. También se evalúa si la organización cuenta con la experiencia técnica para diseñar, implementar y operar el sistema propuesto.
- b. Factibilidad económica. Es un estudio de factibilidad que pretende definir mediante la comparación con los beneficios y costos estimados de un proyecto, si la idea es viable y puede llevarse a cabo y con éxito. La factibilidad económica de un proyecto es esencial para el éxito empresarial pues determina si es recomendable su implementación y posterior operación, dependiendo de los recursos económicos que se deben invertir en la realización del proyecto. Algunos de los costos económicos que se deben tomar en cuenta son: Costo por hora para el pago de cada trabajador, salario por hora, tiempo total del desarrollo en días, costo total del software a utilizar para su desarrollo, costo de renta de una oficina, costo del mobiliario requerido, costo del aire acondicionado, pago de agua, luz, teléfono, etc.
- c. Factibilidad operativa. Determina la probabilidad de que un nuevo sistema se utilice y se integre apropiadamente a las funciones de una empresa. Existen 4 principales motivos por los cuales un nuevo sistema no se utiliza: que el nuevo sistema sea demasiado complejo para los usuarios, que el sistema sea fácil de usar, pero los usuarios se resisten a utilizarlo, que el nuevo sistema introduzca cambios demasiado rápido y no permite que el personal se adapte y acepte el cambio y por último que el nuevo sistema se vuelva obsoleto muy pronto.
- d. Factibilidad legal. Se refiere a la necesidad de revisar que no existan obstáculos legales para la instalación y operación normal del proyecto. Un evaluador puede mediante una investigación exploratoria, definir el marco de restricciones legales que enfrentará el proyecto (Sapag, p. 14).

1.5 Análisis costo-beneficio

El análisis costo-beneficio tiene como objetivo fundamental proporcionar una medida de los costos que se tienen al desarrollar un proyecto de software y comparar el resultado del costo total con los beneficios esperados en la realización del proyecto; en todos los casos, los beneficios deben de superar a los costos.

El análisis costo-beneficio en combinación con el análisis de factibilidad, intenta dar certeza financiera para garantizar que la inversión es redituable y se obtendrán beneficios con el desarrollo del proyecto. El costo de un proyecto es de interés vital para todos los involucrados. Si el costo de producción del proyecto se eleva por encima de los beneficios que se esperan, aún el software mejor diseñado resulta un fracaso. Se requiere que los beneficios y mejoras que proporcione el nuevo software sean superiores que la inversión.

Algunos de los costos que deben considerarse en el análisis de costos son los siguientes:

- a. El precio del software que se utilice en el desarrollo del sistema.
- b. El costo de todos los componentes de hardware utilizados.
- c. El pago al personal involucrado considerando el tiempo invertido y el salario por hora de los trabajadores.
- d. Infraestructura. Renta del local, aire acondicionado, agua potable, instalaciones.
- e. El costo de la implantación del software. Consultoría para la instalación y puesta en funcionamiento del sistema.
- f. El costo de entrenamiento y capacitación dirigido a los desarrolladores de software y demás personal involucrado.
- g. Licencias de software y certificaciones requeridas.
- h. Entrenamiento y capacitación dirigido a los usuarios que se encargarán de utilizar el sistema, etc.

$$\text{Costo total} = a + b + c + d + e + f + g + h$$

Cada proyecto en particular tiene sus propios costos que se identifican mediante el análisis del problema, pero existen algunos otros costos que se repiten en la mayoría de los proyectos. La tabla 1.1 muestra un ejemplo de algunos de los costos que se deben considerar al realizar el análisis costo-beneficio del proyecto.

Nº	Concepto	Estimación
1	Adquisición de software	
2	Adquisición de hardware (o renta del equipo)	
3	Gastos de instalación (cableado y acondicionamiento del área de trabajo)	
4	Recursos humanos (horas-persona para el desarrollo del sistema)	
5	Gastos de consultoría (en caso de requerir algún consultor externo en cualquier etapa del proyecto)	
6	Gastos de formación del personal (desarrolladores, operadores, implantadores, usuarios finales)	
7	Gastos de material (papel, toner, impresora)	

Tabla 1.1 Ejemplo de tabla para la estimación de costos del proyecto

Algunos de los elementos que se pueden considerar para evaluar los beneficios:

- a. Mejora de procesos. Considerar la reducción de tiempo y de recursos que se logra con la implantación del sistema.
- b. Disposición de sistemas de información. Considerar la disposición de información proporcionada por el sistema que puede ser útil para la toma de decisiones y obtención de ingresos.
- c. Motivación del personal. Considerar el incremento de productividad por la creciente moral del personal al trabajar en un entorno de herramientas modernas para el negocio.
- d. Reducción de personal. Al automatizar algunos procesos de la organización, se reducen el número y tiempo de las actividades a realizar, por lo que en ocasiones es posible reducir la mano de obra.
- e. Beneficios intangibles. Otros beneficios adicionales.

Beneficios totales del proyecto = a + b + c + d + e

La tabla 1.2 muestra un ejemplo de algunos de los beneficios más comunes que se obtienen al desarrollar e implantar un nuevo sistema en una empresa.

N°	Concepto	Estimación
1	Mejora de procesos	
2	Disposición de sistemas de información	
3	Motivación del personal	
4	Beneficios intangibles	
5	Reducción de personal	

Tabla 1.2 Ejemplo de estimación de beneficios totales del proyecto

El análisis costo-beneficio es una herramienta financiera que mide la relación entre los costos y beneficios asociados a un proyecto de inversión con el fin de evaluar su rentabilidad. Un proyecto de inversión puede ser la creación de un nuevo negocio, el desarrollo de un nuevo producto de software o de otro tipo, la adquisición de maquinaria, etc.

A la relación costo-beneficio (B/C) también se le conoce como índice neto de rentabilidad y se obtiene cuando se divide el valor actual de los ingresos totales o beneficios netos (VAI) entre el valor actual de los costos de inversión o costos totales (VAC) de un proyecto.

$$B/C = VAI/VAC$$

Se dice que el proyecto es rentable cuando la relación costo-beneficio es mayor que la unidad ($B/C > 1$). Al dividir beneficios entre los costos y el cociente es mayor que 1, esto significa que los beneficios son mayores que los costos, por lo que el proyecto es rentable.

Pasos para realizar el análisis costo/beneficio (Fernández, 2022)

- Encontrar la proyección de los costos de inversión o costos totales.
- Encontrar la proyección de los beneficios o ingresos totales del proyecto.
- Encontrar la relación costo-beneficio dividiendo valor actual de los beneficios entre el valor actual de los costos.
- Analizar la relación B/C, si el valor es mayor que 1, el proyecto es rentable.
- Comparar con otros proyectos, la información histórica proveniente de estimaciones de proyectos anteriores es una información muy valiosa que puede utilizarse para realizar ajustes y consideraciones en el análisis costo-beneficio.

Tema 2: Diseño

El diseño es una actividad creativa donde a partir de una idea o una necesidad, ésta se cristaliza para obtener algo tangible y concreto. El proceso de realizar el diseño de un software y convertirlo en un programa de computadora algunas veces inicia como una idea en la mente del programador o del cliente que tiene una necesidad tecnológica. Esta idea se bosqueja burdamente en una computadora, en un pizarrón o en hojas de papel para poder compartirse con los demás participantes en el proceso de diseño y generar una lluvia de ideas que permita avanzar hacia un diseño más detallado.

El proceso de diseño no es una actividad secuencial y específica, el diseño se lleva a cabo en forma iterativa, proponiendo ideas y soluciones, compartiendo las ideas con los demás participantes del equipo y conforme se obtiene más información y detalles de las necesidades y características del proyecto, se corrigen dichas soluciones y se realizan nuevas propuestas.

En el tema 2 se muestran algunas herramientas que facilitan el proceso de diseño, se conocen los diferentes modelos que se deben realizar y cómo documentar la información del proceso.

Competencia específica: Aplica modelos, técnicas y herramientas para la etapa de diseño del software.

Introducción

El diseño de software es un conjunto de principios, procedimientos, prácticas y metodologías que producen modelos de diseño de calidad los cuales se utilizan durante la construcción de un sistema de software y que se obtienen tomando como base la información generada durante el análisis del sistema. El modelo de diseño es un refinamiento y formalización adicional del modelo del análisis, además se toman en cuenta las características del ambiente de implementación. Como resultado del modelo del diseño se obtienen especificaciones detalladas de todos los objetos, así como sus métodos y atributos. En el modelo del análisis se considera un mundo ideal para el sistema, en el modelo de diseño se considera la realidad y se debe adaptar el modelo al ambiente de implementación.

La etapa de diseño del sistema constituye el eje central de la ingeniería de software, pues en esta etapa se detalla a nivel técnico la arquitectura de software del sistema a desarrollar, así como la relación entre sus componentes, el flujo de información y las interfaces requeridas. La etapa de diseño es importante porque si no existiera un diseño se corre el riesgo de obtener un sistema inestable, que falle cuando se realicen cambios en el mismo, o que sea difícil someter a prueba.

El proceso de diseño en la ingeniería de software se enfoca en definir una estructura general para el software que permita organizar los diferentes elementos identificados en la etapa de análisis. En la etapa de diseño, el ingeniero de software analiza el documento de SRS para plantear las diferentes alternativas y proponer las mejores soluciones para crear el software solicitado. El diseño es el proceso de añadir detalles al análisis realizado y tomar decisiones para seleccionar la mejor alternativa de solución. En la etapa de diseño, se prepara el ambiente para la implementación, de manera que se logre una especificación detallada antes de empezar la implementación final.

El objetivo del diseño es generar un modelo o una representación gráfica del software que se pretende desarrollar. El diseño realizado debe cumplir con las siguientes características:

- Incluir todos los requisitos explícitos obtenidos en la etapa de análisis.
- Ser una guía que pueda ser leída y comprendida en las etapas de desarrollo, pruebas y mantenimiento del sistema.
- Proporcionar una idea completa de lo que el software hará.

La etapa de diseño desarrolla un modelo de implantación basado en los requisitos funcionales y no funcionales documentados durante la etapa de análisis y agregando detalles técnicos para la implementación del software.

La fase de diseño se encarga de desarrollar diferentes modelos, cada uno de estos modelos o diseños se enfocan en representar diferentes características necesarias en la etapa de implementación del software. Por lo general se inicia desarrollando el diseño de los datos y de

las clases utilizadas por el sistema, a continuación, el diseño de la arquitectura del software, para posteriormente seguir con el diseño de la interfaz de usuario y por último llegar hasta el diseño en el nivel de los componentes.

2.1 Diseño de procesos propuestos

El diseño de software define procesos y modelos que inician a partir del análisis realizado. En este contexto, se describe el producto desde diferentes perspectivas. La etapa de diseño se enfoca en 3 procesos principales que son el diseño de la arquitectura, diseño de la interfaz de usuario y por último el diseño en el nivel de componentes.

- **Diseño de la arquitectura.** El diseño de la arquitectura o diseño arquitectónico es la representación gráfica que organiza el sistema en subsistemas que proporcionan algún conjunto de servicios relacionados. El diseño de la arquitectura da soporte a la estructura del software, los estilos y los patrones de diseño que se utilizan para alcanzar los requerimientos definidos por el sistema. En este proceso se define el estilo arquitectónico o la arquitectura de software que adoptará el sistema, las propiedades de los componentes que lo constituyen y sus interrelaciones. El diseño de la arquitectura de software es importante porque permite la construcción del diseño de los componentes del sistema y cómo se interrelacionan entre sí, qué datos de entrada se requieren para cada componente y qué salidas se esperan para cada uno, cómo fluye la información, etc. Su propósito principal es construir un sistema que en un momento determinado pueda ser escalable, robusto, de alto desempeño, de buen rendimiento, seguro y de alta calidad.
- **Diseño de la interfaz de usuario.** Es el proceso de definir la forma en la que un sistema de software o un usuario interactúan con el sistema que se está construyendo. Es una estrategia que crea un medio de comunicación efectivo entre los seres humanos, los sistemas, los dispositivos electrónicos y todo aquel elemento que deba relacionarse con el sistema en desarrollo. Cuando la comunicación es entre el ser humano y la computadora, generalmente los elementos a través de los cuales se logra la interacción son menús y botones, aunque actualmente existen otras formas de interacción como comandos de voz y pantallas sensibles al tacto entre otros.
- **Diseño en el nivel de componentes.** Es el proceso que permite definir los detalles del procesamiento de cada uno de los módulos o procedimientos que formarán parte del sistema. El diseño en el nivel de componentes transforma los elementos estructurales de la arquitectura de software en una descripción detallada de sus componentes en cuanto a procedimiento.

Antes de iniciar el diseño de cualquier diagrama, definir requisitos o empezar a programar, se debe entender el problema, tener una visión clara de lo que el cliente quiere y de sus necesidades. Una vez entendido el problema, se puede iniciar el diseño. Un sistema inicia con una idea para resolver un problema que genera un beneficio y valor para los usuarios.

Cuando se elaboran los modelos de diseño, éstos no tienen que ser perfectos, puede ser que no hayan seguido estrictamente alguna metodología de diseño, o tal vez puede ser que no aborden algunos aspectos del código que, aunque bien se podrían abordar en el diseño, no es necesario detallarlos tanto porque un diseño detallado requiere demasiado tiempo y con frecuencia el diseño cambia conforme se agrega más información al planteamiento del problema. Un modelo de diseño debe servir como guía para elaborar el código y detectar posibles errores. Su propósito principal es comunicar de manera clara lo que hará y que finalmente contemple todo el sistema.

La etapa de diseño es el mejor momento para detectar y corregir errores, porque aún no se ha iniciado el proceso de codificación. Resulta más caro realizar cambios en la etapa de construcción del software, o más caro todavía cuando el sistema haya sido entregado al cliente.

Una vez que se hayan realizado los diferentes modelos en el diseño es necesario asegurarse que los diseños sean correctos. El modelo de diseño debe ser evaluado por el equipo de desarrollo de software con el propósito de determinar si contiene errores, inconsistencias u omisiones, si existen mejores alternativas y si es posible implementar el modelo dentro de las restricciones, costo y plazo establecido.

2.1.1 Herramientas CASE para el diseño

El análisis y diseño de software son dos actividades que en muchos casos son difíciles de separar, en el análisis se elaboran modelos abstractos y de alto nivel, mientras que en el diseño se detallan las características de los modelos generados en el proceso del análisis. Algunas herramientas que utiliza el ingeniero de software en el análisis y diseño de sistemas son:

- **Diagrama de flujo de datos (DFD).** Es una representación gráfica de los flujos de datos de un sistema de información o de un proceso de negocio. Con un DFD se pueden representar flujos de datos entrantes, salientes, así como datos almacenados.
- **Diagramas HIPO (Hierarchical Input Process Output).** Representa los módulos de un sistema en forma jerárquica. Los analistas de sistemas utilizan el diagrama HIPO para obtener una vista organizada de las funciones de un sistema. Descompone las funciones en forma jerárquica.

- **Inglés estructurado (Structured English).** Se utilizan palabras en inglés para describir procedimientos en paradigmas de programación estructurada. Básicamente es una descripción de los requerimientos de un código.
- **Pseudocódigo.** Con esta herramienta se describe el procedimiento de forma más cercana al lenguaje de programación, pero en este caso se pueden agregar comentarios y descripciones. El pseudocódigo contiene más detalles de programación que Structured English, proporciona una metodología para llevar a cabo la tarea, como si la computadora estuviera ejecutando los pasos.
- **Tablas de decisión.** Con esta herramienta se analizan las condiciones que se pueden presentar en un programa. Una tabla de decisión ayuda a eliminar y prevenir errores. También agrupa información similar en la misma tabla con el propósito de analizar la toma de decisiones en el programa de una manera más fácil y conveniente. Para crear una tabla de decisión el desarrollador de software debe identificar todas las decisiones posibles y determinar acciones para todas las condiciones identificadas.
- **Diagrama entidad-relación (E-R).** Denominado por sus siglas en inglés como diagrama E-R (Entity Relationship), es un diagrama que permite representar un modelo de base de datos relacional, enfatizando el concepto de entidades del mundo real y la relación que existe entre las entidades.
- **Diccionario de datos.** Es una colección central de información sobre los datos de un programa. Es un listado de datos organizado que almacena el significado y origen de los datos, formato, nombre, tipo de contenido, y descripción de negocio, de forma que se cuente con un documento para el entendimiento de la base de datos.
- **Diagramas UML.** El lenguaje de modelado unificado (UML) es un lenguaje gráfico que se utiliza para definir, visualizar, desarrollar y documentar un sistema de software, permitiendo especificar el sistema desde diferentes enfoques o “planos” tales como funciones del sistema, estructuras de los datos, flujo de información en los procesos, esquemas de las bases de datos, etc. Los diagramas UML utilizan componentes estándares para su representación como componentes, clases y casos de uso que se enlazan a través de flechas, las cuales representan relaciones y de esta forma muestran la arquitectura, el desarrollo y la estructura de un software. El modelado de un sistema es indispensable en cualquier tipo de proyecto, pero principalmente cuando el proyecto aumenta en tamaño y número de participantes, porque los diagramas UML son el medio de comunicación eficaz entre los involucrados en el sistema.

2.2 Diseño arquitectónico

El diseño arquitectónico o arquitectura del software es el más alto nivel de diseño de la estructura de un programa, es una estructura de organización muy general en el desarrollo de software que proporciona un marco de trabajo bien definido para organizar el código fuente de un sistema de software. Son patrones o lineamientos que sirven de guía para la construcción de un software. Uno de los objetivos de la arquitectura de software es crear una estructura que sea fácilmente escalable.

Cuando se desarrolla un sistema de software que contiene pocas líneas de código y es pequeño en cuanto a las funcionalidades que proporciona, no es evidente la necesidad de definir una arquitectura de software; pero conforme un sistema de software crece debido a la necesidad de agregar nuevas funcionalidades, el sistema se vuelve más complejo, los algoritmos y las estructuras de datos no son capaces de soportar los cambios requeridos en el sistema y empiezan a presentar problemas de diseño y complicar el proceso de mantenimiento del sistema. En estos casos se requiere más tiempo y esfuerzo para realizar los cambios solicitados.

Una arquitectura de software puede mejorar el entendimiento de la complejidad que se presenta en los sistemas de software. La arquitectura de un sistema es muy importante porque hace posible que un software sea mantenible y que sea posible gestionar su complejidad. Una buena arquitectura lleva a minimizar los recursos humanos para construir y mantener el sistema; la arquitectura representa decisiones de diseño y una mala arquitectura provoca que:

- Al realizar cambios se produzcan efectos colaterales que pueden impactar en la calidad del sistema.
- El tiempo y esfuerzo de desarrollo se eleven incrementalmente al realizar modificaciones en el sistema.

Un buen diseño debe realizarse correctamente desde el inicio, dedicar tiempo a la organización de los elementos del sistema para definir la arquitectura correcta que sea capaz de adaptarse a las diferentes necesidades que se presenten durante la construcción del software.

Algunas de las ventajas de diseñar la arquitectura del software son:

- Facilita la comunicación entre los participantes del proyecto. La arquitectura de software constituye una representación de alto nivel del sistema que puede utilizarse como punto de discusión entre los participantes del proyecto. Es un plan de diseño que se utiliza para negociar los requerimientos del sistema y como una forma de estructurar las discusiones con los clientes, desarrolladores, gestores e ingenieros del software.

- Mejora la robustez de un sistema. Contar en forma explícita con la arquitectura del software en una etapa temprana del desarrollo del proyecto, permite considerar elementos para mejorar la robustez del sistema. La robustez es la capacidad de un sistema para recuperarse de una falla, un sistema debe estar protegido contra errores y ofrecer diagnósticos que permitan identificar fallas, por ejemplo, la introducción incorrecta de parámetros proporcionados por el usuario.
- Permite la reutilización de software a gran escala. La reutilización de software es el empleo de componentes creados en desarrollos anteriores cuyo propósito es reducir el tiempo y simplificar el desarrollo, mejorando la calidad y reduciendo costos.
- Mejora la flexibilidad en el mantenimiento de software. La facilidad de realizar cambios cuando son requeridos se define como flexibilidad de un sistema, y es una característica deseable en todo sistema. La mayor parte de los sistemas deben ser extendidos de manera imprevista para adaptarse a las necesidades del usuario. La extensibilidad es la característica que determina si un sistema puede ser ampliado y reimplementado con facilidad en el desarrollo de nuevos servicios.

Las arquitecturas más comunes en el desarrollo de software son:

- Arquitectura cliente/servidor
- Modelo-vista controlador
- Arquitectura jerárquica
- Arquitectura de microservicios

Arquitectura cliente-servidor

Es un modelo de software en el que existe un proveedor de recursos o servicios llamado servidor y un conjunto de computadoras (llamadas máquinas cliente) que requieren de los servicios proporcionados por el servidor (ver figura 2.1). Una máquina cliente realiza peticiones a otra computadora llamada servidor, la cual contiene la información o el servicio que da respuesta a la solicitud del cliente.

Algunos ejemplos de los servidores más comunes son: servidor de correo, servidor de páginas web, servidor de impresión, servidor de nombres de dominio (DNS), servidor de bases de datos, etc. La mayoría de los servicios de internet se basa en la arquitectura cliente-servidor (como diseño de páginas web y servicios on-line).

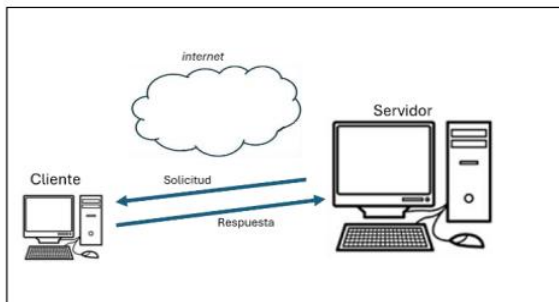


Figura 2.1 Arquitectura cliente-servidor

El funcionamiento de un servidor de correo, por ejemplo, trabaja de la siguiente forma: el mensaje llega primero al servidor, donde el mensaje queda almacenado en el buzón del destinatario. Cuando el destinatario se conecta al servidor, éste le enviará todos sus mensajes pendientes. Por esta razón no importa si el destinatario está conectado o no a internet en el momento en el que se le envía el mensaje.

Modelo-vista-controlador

Como se muestra en la figura 2.2, el modelo-vista-controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en 3 componentes distintos:

- **El modelo.** Es responsable de tener acceso a la capa de los datos, definir las reglas del negocio (un ejemplo de regla de negocio, “si la mercancía pedida no está en almacén, consultar tiempo de entrega del proveedor”). Contiene la representación de los datos que maneja el sistema, su lógica de negocio y sus mecanismos de acceso.
- **La vista.** Es responsable de recibir datos del modelo y mostrar al cliente, puede también dar servicio de actualización. La vista o interfaz de usuario contiene la información que se envía al cliente, generalmente a través de formularios, así como los mecanismos de interacción con éste.
- **El controlador.** Es responsable de recibir eventos de entrada y contiene reglas de gestión de eventos, por ejemplo, “Si sucede X, entonces realizar acción W”. Actúa como intermediario entre el modelo y la vista, gestionando el flujo de la información entre ellos.

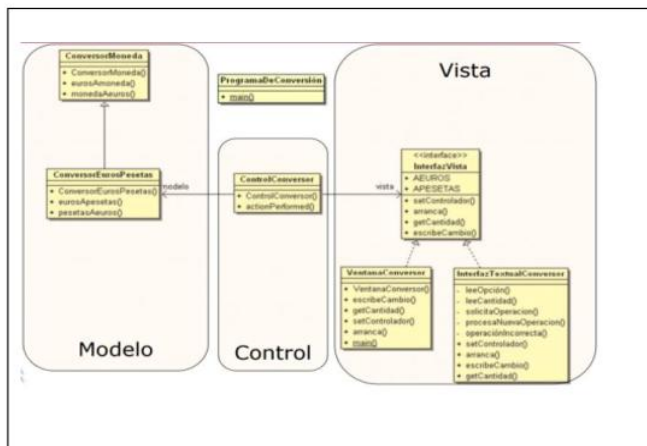


Figura 2.2 Arquitectura Modelo-vista-controlador

En el modelo-vista-controlador cuando el usuario interactúa con el sistema (ver figura 2.3), solicita la información a través del controlador, el controlador es el encargado de solicitar los datos al modelo y enviar dichos datos a la vista, que responde visualmente, mostrando la información solicitada al usuario.

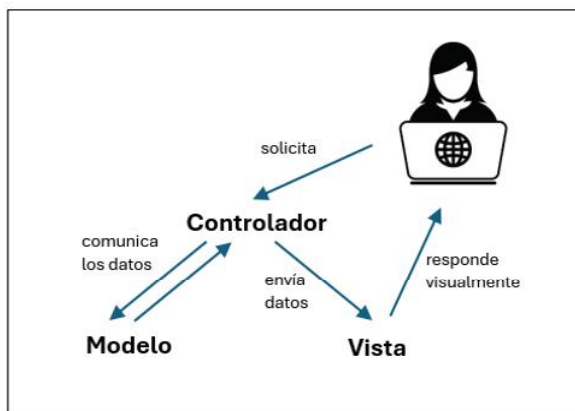


Figura 2.3 Modelo vista controlador

Arquitectura jerárquica

La arquitectura jerárquica es una forma de estructurar los módulos principales de un sistema en múltiples niveles que manera que cada uno de ellos cumpla una función específica (ver figura 2.4).

Es una arquitectura sencilla que permite organizar las funcionalidades del sistema de una forma visual.

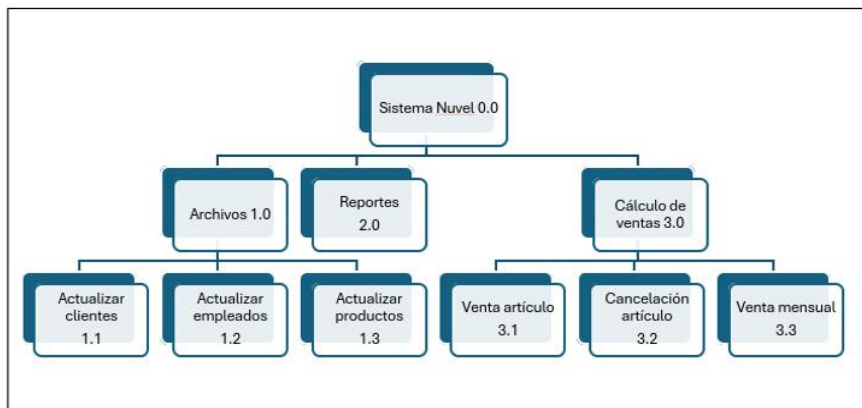


Figura 2.4 Arquitectura jerárquica

Arquitectura de microservicios

Es una estructura de organización en el desarrollo de software que permite construir un software como un conjunto de pequeños servicios los cuales son autónomos e independientes, cada uno de los cuales implementa una funcionalidad completa del negocio. Un servicio se ejecuta en su propio proceso y se comunica con mecanismos ligeros como una API (Fowler, nov 2024).

En la figura 2.5 se muestra un ejemplo de arquitectura de microservicios, los servicios que se gestionan en forma independiente se pueden actualizar sin afectar toda la aplicación. Utilizar una puerta de enlace de API tiene la ventaja de desacoplar los clientes de los servicios. “Los servicios pueden cambiar de versión o refactorizarse sin necesidad de actualizar todos los clientes” (Microsoft, 2024).

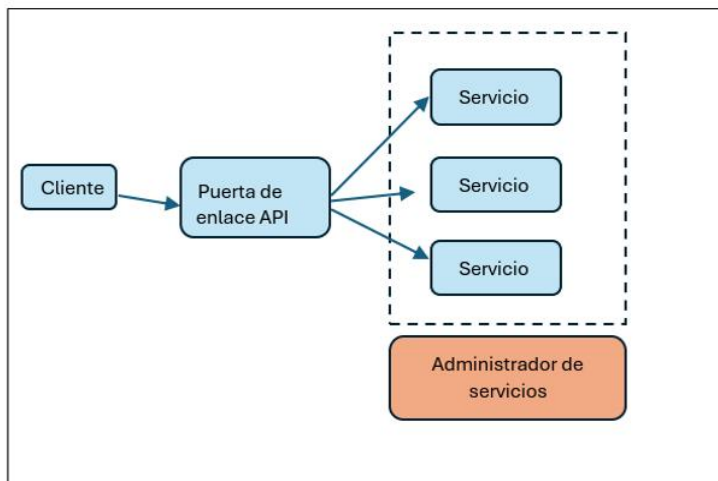


Figura 2.5 Arquitectura de microservicios

2.3 Diseño de datos

La información es una parte fundamental de toda empresa que desea tener un alto nivel de competitividad. El éxito de una compañía depende, en gran medida, de la forma en que se utilicen sus activos intangibles (conocimiento del mercado, imagen de la marca, fidelidad de los clientes, etc.). Una empresa debe ser capaz de conocer y explotar la información del entorno y de la empresa para llegar a ser competitiva.

La información aumenta el conocimiento del usuario y reduce la incertidumbre. Además, una empresa debe estar en condiciones de conocer qué es lo que requiere el mercado consumidor o qué es lo que el cliente valora (calidad, servicio, atención posventa). Las bases de datos proporcionan información valiosa para las empresas, pero requiere de soporte tecnológico y buenas prácticas para la gestión de los datos.

El diseño de una aplicación de bases de datos es una actividad fundamental porque la base de datos contiene la información indispensable para que las empresas tomen decisiones. Diseñar una aplicación que gestione una base de datos implica varias fases:

- Diseño del esquema de la base de datos.
- Diseño de los programas que tienen acceso a la base de datos.
- Diseño del esquema de seguridad para el control de acceso a los datos.
- Diseño de los reportes de información que responden a las necesidades de la empresa.

Una parte importante del proceso de diseño de las bases de datos consiste en decidir la manera de representar los “objetos” identificados durante la fase de análisis como personas, lugares, artículos, departamentos, etc. En base de datos se denomina “entidad” a cualquiera de los objetos o elementos claramente identificados en el proceso de análisis y diseño del sistema.

Se deben aprovechar las características comunes de las entidades para tener un diseño conciso y fácilmente comprensible. Al mismo tiempo se debe conservar la suficiente flexibilidad para representar las diferencias entre las entidades existentes. El diseño realizado debe poder materializarse en un futuro. Las entidades se relacionan entre sí de diferentes formas, las cuales deben incluirse en el diseño de la base de datos.

Al diseñar un esquema de la base de datos hay que asegurarse de evitar la redundancia, un mal diseño puede repetir la información, por ejemplo, en un sistema bancario se podría repetir la información de los clientes (nombre, dirección, teléfono) una vez por cada cuenta o préstamo que tuviera un cliente, lo que puede provocar problemas de inconsistencia.

Se pueden establecer reglas para nombrar los datos, con el propósito de evitar redundancias e inconsistencias.

También se deben documentar aspectos legales sobre los datos como definir aquellos datos que deben tratarse en modo confidencial (Marqués, p. 97). Además, es necesario evitar la incompletitud que se refiere a determinados aspectos de la empresa que son difíciles de modelar, por ejemplo, que el nombre y dirección del cliente del sistema bancario, se almacene con cada cuenta o préstamo, sin contar con una entidad cliente separada, lo que hace imposible almacenar los datos de un cliente que no tenga cuenta o préstamo.

El proceso de diseño de la base de datos comienza con un modelo abstracto de datos hasta llegar a la implementación. El diseño de la base de datos se realiza en 3 etapas (Marqués, p. 95):

- **Diseño conceptual.** El diseñador y los clientes describen las necesidades de información de la empresa, definiendo conceptos/entidades, mostrando las relaciones entre los conceptos y omitiendo detalles finos con el propósito de facilitar el entendimiento y comprensión del problema; los detalles e información específica se abordan en las siguientes etapas.
- **Diseño lógico.** El diseñador traduce el esquema conceptual de alto nivel del modelado de datos en un modelo de implementación del sistema de base de datos a utilizar.
- **Diseño físico.** El diseñador utiliza el esquema de la base de datos resultante del diseño lógico para especificar las características físicas de la base de datos, por ejemplo, la forma de organización de archivos y las estructuras de almacenamiento internas de los datos.

El modelo de la base de datos

El modelo de base de datos determina la estructura lógica de una base de datos y de manera fundamental determina el modo de almacenar, organizar y manipular los datos.

Entre los modelos lógicos comunes para base de datos se encuentran el modelo jerárquico, modelo de red, modelo relacional, modelo entidad-relación, modelo de objetos, modelo documental, modelo entidad-atributo-valor entre otros.

Uno de los modelos más importantes para el diseño de aplicaciones de software que gestionan bases de datos es el modelo relacional. Además, es uno de los modelos más utilizados.

Modelo de base de datos relacional

El modelo relacional representa la base de datos como un conjunto de tablas. Los conceptos básicos del modelo relacional son:

Registro. Es un conjunto de datos tratados como una unidad.

Tabla. Es un conjunto de registros del mismo tipo.

Con estos dos conceptos como base, es posible crear cualquier tipo de datos y asociarlos entre sí para representar un modelo relacional. El modelo de base de datos relacional se enfoca en crear relaciones entre un conjunto de tablas, por ejemplo, en una compañía que se dedica al alquiler de automóviles, se podría tener una tabla con dos columnas que contienen el número de placa de un vehículo rentado y el RFC de la persona que rentó el vehículo. En la figura 2.6 se muestra un ejemplo de la tabla, la cual muestra información interesante: un cliente, puede ser responsable de más de un coche en renta, más de una persona pueden ser responsables de la renta de un vehículo, es decir, comparten el alquiler, los campos utilizados en la tabla de la figura 2.6 son una referencia a un conjunto adicional de datos. El número de placa del vehículo y el RFC del cliente son campos claves que tiene la característica de ser únicos, es decir, no se repiten, cada cliente es identificado por su RFC, que es único, y las placas del carro también son únicas para cada coche.

Número de placa	RFC del cliente
XTM-2263-BC	SAEE680523LH9
XAV-2343-AF	EORJ740319CB4
PVC-1276-CB	SAEE680523LH9
MVC-8890-AB	MASG020627NK8
PVC-1276-CB	JIGY021023HI7

Figura 2.6 Tabla relación Vehículo rentado-cliente

En bases de datos se llama “entidad” a cada “cosa” u “objeto” del mundo real que es distinguible de otros objetos además de ser relevante para el problema en cuestión, por lo que sus características deben ser almacenadas para su posterior tratamiento. Una entidad tiene un conjunto de propiedades a las que se les asigna un conjunto de valores que permiten distinguir de forma unívoca cada entidad. Por ejemplo, el cliente cuyo RFC es EORJ740319CB4 identifica de forma concreta a un cliente de la compañía.

Cada campo clave en una tabla se utiliza para enlazar atributos adicionales al campo clave que están almacenados en otras tablas. Por ejemplo, la tabla de clientes que se muestra en la figura 2.7, contiene información como nombre del cliente, dirección, teléfono, pero también tiene como campo clave el RFC del cliente.

Tabla de clientes

RFC del cliente	Nombre	Dirección	Teléfono	Ciudad
EORJ740319CB4	Jorge Escobar	Av. Hidalgo 404	833-1273089	Tampico
SAEE680523LH9	Eunice Saldierna	Calle Mango 203	833-3304772	Tampico
MASG020627NK8	Gonzalo Martínez	Mariano Otero 107	812-3427663	Monterrey
JIGY021023HI7	Yolanda Jiménez	Primera Av. 346	833-3245485	Cd. Madero

Figura 2.7 Tabla de clientes

También se puede tener una tabla de automóviles en alquiler (ver figura 2.8) que contiene información de los vehículos en renta, por ejemplo, marca del vehículo, modelo, color, precio del alquiler, disponibilidad del vehículo. Es indispensable que en esta tabla exista un campo clave para identificar de forma única el registro.

Tabla de automóviles en renta

Número de placa	Marca	Modelo	Color	Disponible
XTM-2263-BC	Toyota	2022	Blanco	Sí
XAV-2343-AF	Honda	2019	Plata	Sí
PVC-1276-CB	Mazda	2024	Azul	Sí
MVC-8890-AB	Toyota	2021	Plata	No
PVC-1276-CB	Mazda	2022	Rojo	Sí

Figura 2.8 Tabla de automóviles en renta

Un modelo de datos entidad-relación se desarrolló para facilitar el diseño de bases de datos permitiendo la especificación de un esquema de la empresa que representa una estructura lógica global de la base de datos. El modelo E-R es un modelo semántico que permite describir el significado de sus instancias. “Un modelo de datos semántico enfatiza el significado y el contexto de los datos, proporcionando una comprensión de cómo los diferentes elementos de datos se relacionan entre sí” (PhoenixNAP, 2024).

La semántica del modelo es una característica importante para comprender la estructura de los datos que se están diseñando. El modelo E-R emplea 3 conceptos básicos: el conjunto de entidades, el conjunto de relaciones y el conjunto de atributos.

En una tabla de entidades, para cada atributo hay un conjunto de valores permitidos denominado dominio o conjunto de valores de ese atributo, por ejemplo, el dominio del atributo nombre del cliente, puede ser un conjunto de todas las cadenas de texto con una longitud de 30 caracteres. El dominio del atributo RFC es un conjunto de 13 caracteres, los cuatro primero alfabéticos, los siguientes seis numéricos y los últimos 3 alfanuméricos.

2.4 Diseño de interfaz de usuario

El diseño de la interfaz de usuario es una estrategia que crea un medio de comunicación eficaz entre los seres humanos y la computadora. Es la etapa de diseño que se centra en la interacción humana con la computadora por medio del uso simplificado de interfaces gráficas de usuario (GUI).

De acuerdo con la forma en que el usuario interactúa con el sistema, se pueden distinguir 3 tipos de interacción:

- Interfaz de línea de comandos. La interfaz de línea de comandos (Command Line Interface, CLI), es un conjunto de instrucciones que se dan de forma explícita para interactuar con el sistema. En este tipo de interfaz, las instrucciones se proporcionan a través de comandos, por lo que su codificación es estricta. Un ejemplo de este tipo de interfaz es la línea de comandos CMD de Windows, es poco intuitiva y requiere de conocimientos técnicos. Es para usuarios avanzados.
- Interfaz gráfica de usuario. Es una interfaz gráfica de usuario (Graphics Users Interface, GUI), que tiene las propiedades de ser metafórica y exploratoria, pues a través de elementos intuitivos que el usuario entiende como botones, íconos y ventanas que representan elementos del mundo real, el usuario puede explorar el uso de la herramienta y comprender más fácilmente cómo interactuar con la misma.
- Interfaz natural de usuario (o interfaz Software-Hardware). La interfaz natural de usuario (Natural User Interface, NUI) es un tipo de interfaz en la que no se utilizan comandos o dispositivos de entrada como teclado o lápiz óptico para la interacción con el sistema. En este tipo de interfaz, la interacción es a través de movimientos gestuales del cuerpo o las manos, lo que sirve como comando de control. También existen interfaces por medio de la voz humana, lo que se conoce como reconocimiento del habla, así como la utilización de las yemas de los dedos y control cercano a la pantalla, pero sin tocarla.

Al diseñar la interfaz de usuario, se debe considerar la usabilidad que es el criterio no funcional más importante que se centra en la facilidad con que el usuario puede utilizar una aplicación, es el grado de satisfacción de un usuario al interactuar con el sistema, es un factor subjetivo, pues depende de la percepción del usuario, sin embargo, la usabilidad se puede evaluar a través de un conjunto de propiedades que deben cumplirse en cualquier interfaz, la flexibilidad que permite que usuarios con distintos perfiles, tanto novatos como expertos puedan utilizar la aplicación en forma eficiente. Las rutas de navegación en la aplicación deben estar claramente especificadas y predecibles en su comportamiento. Minimizar el número de pasos que el usuario tiene que dar para realizar una tarea.

Algunas de las características de la usabilidad, están contenidas en lo que se denomina las reglas de oro para diseñar una interfaz de usuario (Pressman. P. 266), éstas son: dar el control al usuario, reducir la carga de memoria del usuario y construir una interfaz consistente.

a) Dar el control al usuario

Al preguntar a un usuario, qué atributos deseaba de una interfaz gráfica de usuario, el usuario responde:

“Lo que me gustaría es un sistema que lea mi mente. Que conozca lo que quiero hacer antes de necesitarlo y que me facilite hacerlo. Eso es todo” (Pressman, p.266).

Al hacer un análisis de la solicitud del usuario, lo que el cliente quería es que el sistema conociera sus necesidades, reaccionara ante dichas necesidades y le facilitara la tarea de realizar las funciones deseadas. Es decir, el usuario quiere controlar la computadora y no que la computadora controle al usuario. Cuando un ingeniero de software comprende las necesidades de los usuarios está en condiciones de diseñar una interfaz que reacciona ante su solicitud. Algunas de las características que debe tener la interfaz para dar el control al usuario son:

- “Definir los modos de interacción de manera que no obligue a que el usuario realice acciones innecesarias y no deseadas”. El estado actual de la interfaz define el modo en el que el usuario puede interactuar con el usuario, y éste, debe poder entrar y salir de este modo de interacción sin mucho o ningún esfuerzo.
- Tener en consideración una interfaz flexible. Dado que diferentes usuarios tienen diferentes preferencias de interacción, se deben proporcionar diferentes opciones de interacción (mouse, teclado, voz, etc.)
- “Permitir que la interacción del usuario se pueda interrumpir y deshacer”. Cuando un usuario realiza una secuencia de acciones debe poder interrumpir la secuencia actual hasta un punto necesario, o hacer cualquier otra cosa sin perder el trabajo que haya realizado anteriormente.
- “Aligerar la interacción a medida que avanza el nivel de conocimiento y permitir personalizar la interacción”. Un usuario que se encuentra realizando la misma secuencia de acciones de manera repetida, necesita un mecanismo de “macros” que le permita al usuario personalizar la interfaz y así reducir y facilitar la interacción.
- Ocultar los tecnicismos internos al usuario ocasional. La interfaz debe introducir al usuario en el mundo virtual de la aplicación, el usuario no debe preocuparse por el manejo del sistema operativo, de las funciones de administración de archivos ni de ninguna otra tecnología que requiera que el usuario interactúe en un nivel “interno” de la máquina.

- “Diseñar la interacción directa con objetos que aparezcan en la pantalla”. El usuario tiene la sensación de control cuando manipula los objetos en la pantalla a fin de ejecutar un trabajo en la misma forma que lo haría si el objeto fuera físico. Por ejemplo, una interfaz de aplicación que permita “estirar” un objeto.

b) Reducir la carga de memoria del usuario

Entre más cosas deba recordar un usuario, más fácil es cometer errores al interactuar con el sistema, una interfaz bien diseñada, no sobrecarga la memoria del usuario. Siempre que sea posible, el sistema debe recordar la información pertinente y ayudar al usuario con un escenario de interacción que lo ayude a recordar (Pressman, p. 267). Para reducir la carga de memoria del usuario se deben realizar las siguientes acciones:

- Reducir la demanda de memoria a corto plazo. La interfaz debe diseñarse para disminuir la necesidad de recordar acciones, entradas y resultados anteriores, esto se logra dando claves visuales que permitan al usuario reconocer acciones anteriores en lugar de recordarlas.
- Hacer que lo preestablecido sea significativo, lo que se establece al principio, debe tener sentido para el usuario promedio, con la posibilidad de especificar preferencias individuales y disponer de una opción de reestablecer los valores originales.
- Definir atajos que sean intuitivos. Cuando se use mnemotecnia para ejecutar una función del sistema, debe estar ligada con la acción, de modo que sea fácil recordar, por ejemplo, la primera letra de la tarea que se va a realizar.
- La distribución visual de la interfaz debe basarse en una metáfora del mundo real, por ejemplo, un sistema de pagos debe usar una metáfora de chequera y talonario que guíe al usuario a través del proceso de pago. Esto permite que el usuario se base en claves visuales que comprende bien, en vez de memorizar una secuencia de interacciones.
- Revelar información en forma progresiva. La interfaz debe estar organizada en forma jerárquica, hasta el momento que el usuario quiera ver las opciones de un menú, éstas deben listarse.

c) Construir una interfaz consistente

Una interfaz consistente se enfoca en mantener un estilo uniforme, una interfaz bien organizada de forma significativa y útil para el usuario y mantiene un patrón visual similar en toda la aplicación. El diseño de una interfaz consistente trata de crear un lenguaje visual coherente de manera que un usuario pueda aprender fácilmente la utilización de un sistema y pueda aplicar de manera intuitiva a través de diferentes secciones de la aplicación.

Para lograr este objetivo:

- La interfaz debe presentar y obtener información en forma consistente. Esto implica respetar las reglas de diseño en todas las pantallas, limitar los mecanismos de entrada a los mínimos posibles, y que los mecanismos para pasar de una tarea a otra se definan e implementen en forma consistente.
- Permitir que el usuario coloque la tarea en curso en un contexto significativo. Proporcione indicadores como títulos de ventana, íconos gráficos, códigos de colores consistentes que permitan al usuario conocer el contexto de trabajo en curso. Además de determinar de dónde viene y qué alternativas hay para hacer la transición a una nueva tarea.
- Mantener la consistencia en toda la familia de aplicaciones. Todas las aplicaciones que hay en un grupo, deben implementar las mismas reglas de diseño con el fin de que se mantenga la consistencia en toda la interacción.
- Si los modelos interactivos anteriores han creado expectativas en el usuario, no haga cambios a menos que sea estrictamente necesario. Es decir, si una secuencia interactiva en particular se ha convertido en un estándar, por ejemplo, Alt+G para guardar un archivo, el usuario espera en toda la aplicación utilizar dicha combinación de teclas para esa tarea, y no para ninguna otra.

El proceso de análisis y diseño de la interfaz de usuario

El proceso general de análisis y diseño de la interfaz de usuario comienza con la creación de diferentes modelos del funcionamiento del sistema, según se percibe desde fuera. Se empieza con el delineado de las tareas orientadas al usuario que se requieren a fin de obtener el funcionamiento del sistema, para luego considerar todos los aspectos que se aplican a todos los diseños de la interfaz. Al realizar el análisis y diseño de la interfaz se crean 4 modelos (Pressman, p. 269):

- **El modelo del usuario.** Establece el perfil de los usuarios finales del sistema. “Todo diseño de la interfaz debe comenzar con la comprensión de los usuarios finales del sistema, lo que incluye perfiles de edad, género, condiciones físicas, educación, antecedentes culturales o étnicos, motivación, metas y personalidad”. Los usuarios se clasifican en principiantes, intermitentes y frecuentes. Los usuarios principiantes tienen poco conocimiento de la aplicación o del uso de la computadora en general. Los usuarios intermitentes tienen conocimiento semántico de la aplicación, pero relativamente poco recuerdo de la información sintáctica necesaria para utilizar la aplicación, mientras que los usuarios frecuentes, también llamados conocedores,

tienen buen conocimiento semántico y sintáctico de la aplicación, son individuos que buscan atajos y modos de interacción abreviados.

- **El modelo de diseño.** Define una representación gráfica de la distribución y organización de los elementos contenidos en las interfaces de entrada y salida de información de un sistema en desarrollo. Es un dibujo o visión representada gráficamente del futuro de la interfaz del sistema. Se refiere a esbozos, dibujos, bocetos o esquemas trazados en cualquiera de los soportes, durante o posteriores a un proceso de observación de alternativas o investigación. Diseñar una interfaz es un acto intuitivo donde predomina la creatividad.
- **Percepción del sistema.** Es la imagen del sistema que los usuarios tienen en la mente. Por ejemplo, si se pidiera a un usuario de un procesador de texto que describiera su operación, lo que guiaría su respuesta sería la percepción del sistema; un usuario percibe a un procesador de texto como una hoja de papel en blanco en la que puede escribir, borrar y organizar su información. La precisión de la descripción depende del perfil del usuario.
- **Modelo de implementación.** Combina la manifestación externa del sistema basada en la computadora (vista y sensación de la interfaz) con toda la información de apoyo (manuales, videos o archivos de ayuda) que describe la semántica y sintaxis de la interfaz. Cuando el modelo de implementación y el modelo mental (percepción del sistema) coinciden, quienes utilizan el software por lo general se sienten cómodos con éste y lo utilizan de manera eficaz. Para lograr la fusión de estos modelos, el modelo de diseño debe haberse desarrollado de manera que incluya la información contenida en el modelo del usuario y el modelo de implementación debe reflejar la información sintáctica y semántica de la interfaz.

El proceso de análisis de la interfaz se centra en el perfil de los usuarios que interactuarán con el sistema. Se registra el nivel de habilidad, la comprensión del negocio y la receptividad hacia el nuevo sistema. También se definen diferentes categorías de usuarios, se recaban los requisitos de cada una de estas categorías, se trabaja para entender la percepción del sistema para cada clase de usuario.

Una vez definidos los requisitos generales, se lleva a cabo un detallado análisis de tareas, se identifican, se describen y elaboran aquellas tareas que el usuario realice para alcanzar las metas del sistema. Por último, se realiza un análisis del ambiente del usuario, que se centra en las características físicas del lugar de trabajo. Se responden preguntas como: ¿Dónde se encontrará físicamente la aplicación? ¿El usuario estará sentado, de pie o realizando otras tareas no relacionadas con la interfaz? ¿El hardware de la interfaz cumple las restricciones de espacio, iluminación o ruido? Con la información recabada como parte del modelo del análisis se comienza el proceso de diseño de la interfaz.

Un aspecto primordial es entender el problema antes de tratar de diseñar la solución. Entender el problema significa comprender:

- Las personas que interactuarán con el sistema.
- Las tareas que los usuarios deben hacer en su trabajo.
- El contenido que se presenta como parte de la interfaz.
- El ambiente en que se efectúan estas tareas.

Cuando se realiza el diseño de la interfaz, se deben realizar las siguientes acciones:

- Definir objetos y acciones de la interfaz (operaciones) utilizando la información identificada en el análisis.
- Definir eventos (acciones del usuario) que harán que cambie el estado de la interfaz.
- Ilustrar cada estado de la interfaz como lo vería en realidad el usuario final.
- Indicar cómo interpreta el usuario el estado del sistema a partir de la información proporcionada a través de la interfaz.

Tema 3: Desarrollo

El proceso de desarrollo de software se ha convertido en un desafío pues la tendencia de la globalización de los mercados y de la diversificación de las empresas empuja hacia la construcción de software complejo y de gran magnitud. Tradicionalmente el desarrollo de software inicia con la participación de un equipo de desarrollo pequeño y centralizado donde los integrantes del equipo se reúnen frecuentemente para revisar y trabajar en el avance del proyecto, en la actualidad, la industria del desarrollo del software sobrepasa las fronteras nacionales alcanzando una dimensión global. Los equipos de desarrollo de software pueden estar geográficamente distribuidos sin que por esta razón demeriten los resultados.

Los equipos de desarrollo de software colaboran activamente sin importar su ubicación geográfica, la empresa a la que pertenecen, ni sus diferencias de tiempo disponible. “El desarrollo global de software se ha consolidado como uno de los aspectos más relevantes en la investigación y la práctica de la Ingeniería del Software” (Vizcaíno, p. 38).

El objetivo de este tema es comprender los conceptos vinculados al desarrollo de software utilizando técnicas de flujo de trabajo que permitan la integración de software mediante el uso de herramientas para controlar las versiones de un software.

Competencia específica: Construye un software derivado de la problemática planteada en el análisis y diseño de un sistema.

3.1 Lenguajes para el desarrollo de software

Una de las actividades más importantes que debe llevar a cabo un Ingeniero de software es la codificación de un programa fuente. Cuando se codifica un programa, se traducen los requisitos del sistema a un formato que la computadora es capaz de interpretar. Utilizando un lenguaje de programación, el ingeniero de software le indica a la computadora, Tablet o teléfono celular qué es lo que debe hacer.

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras, tablets o teléfonos celulares. Un lenguaje de programación proporciona al ingeniero de software la capacidad de escribir una serie de instrucciones o secuencias de órdenes con el objetivo de controlar el comportamiento físico o lógico de un sistema informático, expresar algoritmos con precisión o generar interfaces para lograr la comunicación con los seres humanos.

El lenguaje de programación está formado por un conjunto de símbolos, reglas sintácticas y reglas semánticas que definen su estructura y el significado de sus elementos y expresiones. Un lenguaje de programación permite especificar un programa de computadora que el compilador es capaz de interpretar. Existe una gran variedad de lenguajes de programación para el desarrollo de software que van desde los especializados (como los lenguajes de pruebas de hardware) hasta los lenguajes de propósito general como C++ y java. Tan sólo para el desarrollo de páginas web en internet existe una amplia variedad de lenguajes, por ejemplo, JavaScript, Python, HTML, PHP, etc. Los lenguajes de programación permiten crear todos los servicios, sitios, entornos digitales y soportes informáticos que existen, hay una gran variedad de opciones como son: JavaScript, HTML/CSS, Python, SQL, Java, Node.js, TypeScript, C#, Bash/Shell, C++, PHP, C, PowerShell, Go, Kotlin, Rust, Ruby, Dart, Assembly, Swift, R, VBA, Matlab, Groovy, etc.

Al conjunto de instrucciones escritas utilizando un lenguaje de programación se le denomina programa de computadora. Y al proceso de crear un software o programa de computadora mediante la escritura, prueba, depuración, compilación y mantenimiento del código fuente se le denomina programación. Algunas de las principales tareas que el Ingeniero de software lleva a cabo durante el proceso de programación son:

- Desarrollar la lógica del programa o el algoritmo que sea capaz de resolver un problema en particular (semántica del programa).
- Escribir el programa en un lenguaje de programación específico.
- Realizar la compilación o interpretación del programa hasta convertirlo en lenguaje máquina.
- Realizar pruebas para verificar que el programa realice el objetivo planteado.
- Depurar el programa, es decir, corregir las fallas y errores cuando se presenten en el código.
- Integrar el programa desarrollado con otro programa principal.

- Trabajar con otros integrantes del equipo para tomar decisiones respecto al código fuente.
- Utilizar herramientas y sistemas de control de software para compartir versiones del código desarrollado con el equipo de desarrollo de software.
- Documentar toda la información generada durante el proceso de programación como son los programas, bases de datos, variables y procesos.

La sintaxis y semántica de un programa

Algunos lenguajes para el desarrollo de software como el JavaScript, las primeras versiones de Visual Basic y C++ están basados en objetos (Braude, p. 361), los cuales describen elementos del mundo real para crear programas que representen las problemáticas de la vida cotidiana. La meta real de la ingeniería de software es crear el programa correcto (es decir, el programa que se ajusta a los requisitos del cliente), pero los compiladores sólo pueden verificar sintaxis y generar código objeto, el desarrollo del código correcto es responsabilidad humana.

Se llama semántica de un programa a estudiar el comportamiento de la ejecución de un programa. La semántica del programa sólo puede ser generada por el ser humano y no por las computadoras. La sintaxis describe la forma visual de un programa, mientras que la semántica describe el comportamiento del programa. Por lo que es esencial que el profesional esté convencido de la exactitud del código antes de someterlos a compilación.

Como se menciona en (Braude, p. 361): “En principio, es posible compilar primero y verificar que sea exacto o correcto después, esto es poco efectivo”, “como corregir la sintaxis de una carta antes de estar seguro de que expresa el pensamiento adecuado”. “Más aún, los programadores tienden a omitir la verificación exhaustiva del código del programa una vez que compila sin errores (de sintaxis)”. La lógica en la programación es un aspecto importante que debe asegurarse mediante la construcción previa de algoritmos y pseudocódigos que logren verificar el programa correcto.

Para una mayor eficiencia, se sugiere seguir los siguientes pasos en el proceso de programación:

- a. Planear la estructura y diseño arquitectónico para el código.
- b. Realizar los diseños detallados que faltan, anote precondiciones, postcondiciones, registre el tiempo dedicado a la codificación.
- c. Revisar e inspeccionar el diseño.
- d. Capturar el código fuente (no compile todavía, aplique los estándares requeridos).
- e. Revisar e inspeccionar el código fuente, (no compile todavía, asegure que el código realiza el trabajo requerido, el compilador no puede revisar la lógica del programa, sólo verifica la sintaxis).
- f. Compilar el código y corregir los errores de sintaxis.
- g. Realizar pruebas en el código.

3.2 Manejadores de bases de datos

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. Una biblioteca, por ejemplo, se considera una base de datos que está compuesta por documentos y textos impresos en papel que están organizados e indexados para su consulta, es decir, es posible extraer o recuperar un libro específico debido a que cada libro tiene un número de identificación única.

Debido a la importancia de la información, las bases de datos generalmente se almacenan en computadoras que están dedicadas exclusivamente a su gestión, por sus características requieren de computadoras multiprocesador y abundante memoria. Las bases de datos han sido una de las aplicaciones más importantes desde los inicios de las computadoras.

Un modelo de base de datos se refiere a la descripción del contenedor de datos, es la estructura donde se almacenan los datos, así como los métodos para almacenar y recuperar datos de los contenedores. Los modelos de bases de datos permiten la implementación de un sistema eficiente de base de datos.

Por otro lado, un sistema de gestión de bases de datos, SGBD, (o por sus siglas en inglés DBMS, Data Base Management System) es un software que permite administrar la información de una base de datos. Proporciona un mecanismo de organización, el cual es necesario para almacenar y recuperar de manera flexible grandes cantidades de datos. Un DBMS permite utilizar, configurar y extraer la información almacenada en la base de datos para que los usuarios tengan acceso a la información utilizando herramientas específicas para la consulta y generación de informes.

Los DBMS también proporcionan mecanismos para mantener la integridad de los datos, para administrar el acceso de los usuarios, realizar copias de seguridad y recuperar información si el sistema falla. Algunos ejemplos de DBMS son MySQL, MariaDB, PostgreSQL, Microsoft SQL Server, Oracle Database, Microsoft Access, MongoDB, Redis y Apache Cassandra entre otros.

Un sistema de gestión de base de datos (también denominado sistema administrador de bases de datos) tiene como soporte un modelo de base de datos. El modelo de base de datos permite almacenar y recuperar información de una base de datos de acuerdo con su estructura de datos, es decir, los datos almacenados tienen un esquema o modelo de organización que permite un almacenamiento y recuperación eficiente de datos, de acuerdo con alguna característica específica. En la evolución histórica de las bases de datos se menciona que se han utilizado diferentes modelos, algunos de ellos son:

- Modelo de base de datos jerárquico
- Modelo de base de datos relacional
- Modelo de base de datos de red
- Modelo de base de datos multidimensional
- Modelo de base de datos orientada a objeto
- Modelo de base de datos documental

En la práctica, no todas las empresas tienen su información almacenada en alguno de estos tipos de bases de datos. Muchas organizaciones, especialmente los bancos, por haber sido los primeros en informatizarse, siguen teniendo algunos de sus datos sobre archivos electrónicos sencillos, manejados directamente a través del sistema operativo, y no con la intervención de un sistema manejador de bases de datos que facilite su gestión y su mantenimiento; aunque esta situación paulatinamente está cambiando.

Modelo de base de datos jerárquico

Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos compartidos permitiendo crear estructuras estables y de gran rendimiento. Una de las limitaciones de este modelo es su incapacidad de representar eficientemente redundancia de datos.

El modelo de base de datos jerárquico fue utilizado en los primeros mainframes que existieron. Las relaciones entre los registros de la base de datos forman una estructura de árbol, este tipo de estructura es simple, pero es inflexible al momento de recuperar la información, pues las relaciones entre los registros son del tipo uno a muchos (1:N). La figura 3.1 muestra el esquema de organización del modelo de base de datos jerárquico.

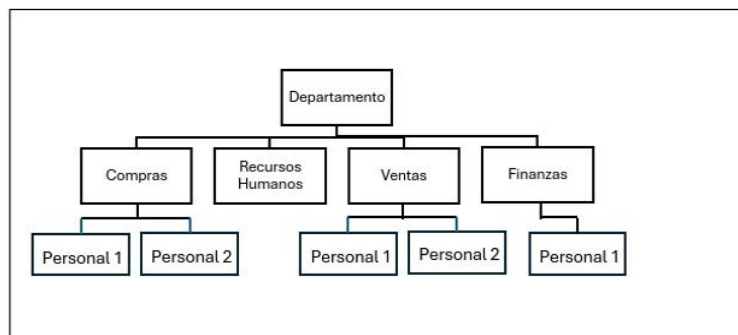


Figura 3.1 Modelo de base de datos jerárquico

El modelo de base de datos jerárquico es utilizado para almacenar información geográfica principalmente (Wikipedia, SGBD). Algunos otros ejemplos de información que por su estructura pueden almacenarse utilizando un modelo jerárquico son el directorio de Windows que tiene un directorio principal a partir del cual se pueden crear carpetas y dentro de estas carpetas se crean subcarpetas, pero el directorio raíz de todas las carpetas siempre es el directorio principal, generalmente denominado directorio C:/. Otro ejemplo de este tipo de estructura de organización es la información de un libro, donde un libro tiene capítulos y dentro de cada capítulo temas y subtemas, en la tabla de contenido de un libro se visualiza la organización jerárquica de los temas y subtemas de cada uno de los capítulos de un libro.

Las bases de datos jerárquicas son especialmente útiles en el caso de aplicaciones que manejan un gran volumen de información y datos compartidos permitiendo crear estructuras estables y de gran rendimiento. Una de las limitaciones de este modelo es su incapacidad de representar eficientemente redundancia de datos.

Para el modelo de bases de datos jerárquico se utiliza una estructura de datos llamada árbol, en donde un nodo principal llamado raíz del árbol puede tener uno o varios hijos, cada uno de estos nodos o hijos a su vez, puede tener uno o varios hijos creando un árbol.

Como se muestra en la figura 3.2 en el árbol solamente existe un solo nodo que no tiene padre y se denomina raíz. Todos los demás nodos tienen un padre, que representa un nivel superior al nodo actual. Los nodos que no tienen hijos se les conoce como nodos hojas y son los nodos con el nivel más inferior que existe en el árbol. Un nodo puede tener descendientes, los cuales son todos los nodos que se encuentran en los niveles inferiores a partir de él. Un nodo y sus descendientes pueden separarse del árbol principal y formar un subárbol. Esta característica es importante porque permite el manejo de funciones recursivas en los subárboles del árbol principal.

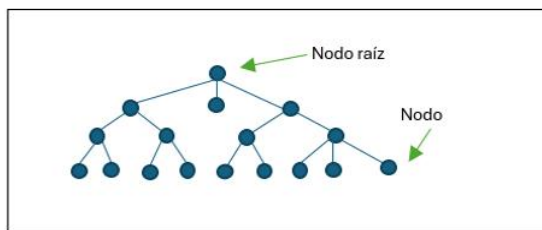


Figura 3.2 Estructura de árbol

Modelo de base de datos relacional

El modelo de base de datos relacional es un modelo muy utilizado en diferentes arquitecturas de hardware como son mainframes, computadoras medianas y microcomputadoras. Almacena la información como un conjunto de filas y columnas en una tabla. Los conceptos básicos que representan un modelo relacional son:

Registro. Es un conjunto de datos relacionados y tratados como una unidad.

Tabla. Es un conjunto de registros del mismo tipo.

Una base de datos relacional consiste en un conjunto de tablas, a cada una de las cuales se le asigna un nombre único. Cada fila de una tabla representa una relación entre un conjunto de valores. De manera informal se puede decir que cada tabla es un conjunto de entidades y cada fila en la tabla representa una entidad.

El diseño de una base de datos relacional es una actividad fundamental en el desarrollo de software, ya que un mal diseño trae como consecuencia problemas al tratar de visualizar la información solicitada. Analizar y conocer los detalles de la base de datos permite saber sus posibilidades y limitaciones, las preguntas que se pueden formular y las respuestas que razonablemente se pueden extraer de ella. Una base de datos debe representar la realidad tal y como es, así como sus distintas condiciones.

Normalización

Para tratar de realizar un buen diseño de la base de datos, se utiliza la estrategia de normalización; esta técnica se centra en el diseño de las bases de datos relacionales, y permite diseñar la estructura lógica de los datos en un sistema de información en el modelo relacional. La normalización es el proceso de organizar los datos en una base de datos.

Desde una visión general, el diseño de la base de datos facilita la obtención de los requisitos de información del sistema. El modelo relacional trata de traducir el universo real en términos de tablas. Las tablas representan entidades del mundo real, así como los enlaces o relaciones que puedan existir entre las entidades. La idea es sencilla: un conjunto de relaciones (tablas) es reemplazado por otro conjunto de tablas con una estructura más simple y regular con el objetivo de minimizar en lo posible la redundancia de información y evitar anomalías que se pudieran producir en el mantenimiento de la base de datos.

Por ejemplo, tener datos redundantes en una base de datos genera desperdicio de espacio, además, cuando se requiere una modificación en un dato, se tiene que cambiar en todos los lugares donde existe dicho dato. Es más sencillo y menos propenso a errores cambiar, por ejemplo, la dirección de un cliente en un solo lugar y no en diferentes tablas de la base de datos.

Una dependencia incoherente se refiere a una situación que impide el acceso a los datos por encontrarse en un lugar incorrecto o ilógico. Por ejemplo, buscar en la tabla de clientes el precio de un artículo, de forma intuitiva se puede inferir que el precio de un artículo debería de estar en otra tabla, como por ejemplo la tabla de artículos.

La normalización mejora la comprensión y la semántica de las relaciones entre las tablas. Existen algunas reglas de normalización de una base de datos, cada una de estas reglas se denomina “forma normal”. Si una base de datos cumple con la primera regla, se dice que la base de datos está en la primera forma normal (1FN).

Para que una base de datos se encuentre en la segunda forma normal (2FN) debe de cumplir con la primera y la segunda forma normal. Aunque existen otros niveles de normalización, la tercera forma normal (3FN) se considera el nivel más alto requerido por la mayoría de las aplicaciones.

Primera forma normal (1FN)

En la primera forma normal (1FN) cada columna debe tener sólo un valor y no debe haber grupos repetidos, es decir, cada celda no debe tener múltiples valores. Se dice que un dominio de un campo es atómico si se considera que los elementos de ese dominio son unidades indivisibles. El esquema de la relación R está en la primera forma normal (1FN) si los dominios de todos los atributos de R son atómicos.

Id	nombre	servicios
1	Juan	Corte de Cabello
2	Pedro	Corte de Cabello, Barba
3	Karen	Peinado
4	Alejandra	Uñas
5	Raquel	Uñas, Tinte, Peinado

Figura 3.3. Atributo con múltiples valores en una tabla

En la primera forma normal, cada columna de una tabla debe tener un solo valor y no debe haber grupos repetidos. En la figura 3.3 se muestra una tabla con el campo servicios, el cual no cumple la condición de contener sólo datos atómicos, por lo que no se encuentra en la 1FN. En este ejemplo, es necesario separar los datos para que sean atómicos. Una alternativa para separar los datos es crear una columna para cada dato.

Id	nombre	servicios	servicio2	servicio3
1	Juan	Corte de Cabello		
2	Pedro	Corte de Cabello	Barba	
3	Karen	Peinado		
4	Alejandra	Uñas		
5	Raquel	Uñas	Tinte	Peinado

Figura 3.4. Ejemplo de separación de servicios en diferentes columnas

En la figura 3.4 se muestra que el campo servicios ha sido separado en diferentes atributos, pero se generan grupos repetidos, es decir, atributos del mismo tipo en diferentes columnas. Los grupos repetidos generan el problema de dejar muchos campos vacíos, pues no se conoce la cantidad de atributos que tiene cada una de las entidades.

Para solucionar el problema de grupos separados, se crea una segunda tabla cuyo objetivo es almacenar los diferentes servicios prestados por la empresa, de esta forma, se puede hacer referencia a esta tabla cuando se requiere un servicio.

Para cumplir con las características de la 1FN:

- Eliminar grupos repetidos de tablas individuales.
- Crear una tabla independiente por cada conjunto de datos relacionados.
- Identificar cada conjunto de datos relacionados con una clave principal, de tal manera que los demás datos de la tabla dependan o puedan ser identificados por la clave principal.

Segunda forma normal (2FN)

La segunda forma normal (2FN) dice que una relación está en 2FN si y sólo si, la relación está en 1FN y además cada atributo que no es clave primaria depende completamente de la totalidad de la clave y no de un subconjunto de ella. Esto quiere decir que no incluye dependencias parciales, los atributos dependen de la clave primaria y no sólo de una parte de la clave primaria. Para cumplir con las características de la 2FN (ver figura 3.5):

- Crear tablas independientes para conjuntos de valores que se apliquen a varios registros.
- Relacionar estas tablas con una clave externa.

Clientes		Citas	
id	nombre	servicios	clienteid
1	Juan	Corte de Cabello	1
2	Pedro	Corte de Cabello	2
3	Karen	Corte de Barba	2
4	Alejandra	Peinado	3
5	Raquel	Uñas	4
		Uñas	5
		Tinte	5
		Peinado	5

Figura 3.5 Creación de una tabla separada de servicios

En la 2FN la identificación de un registro sólo depende de su clave principal. Si un campo es requerido en varias tablas, se puede optar en almacenarla en alguna de las tablas y en las demás tablas hacer referencia a una clave foránea. Cada registro debe de tener su propia clave primaria, por lo que se deben agregar claves principales a cada tabla. Por ejemplo, el campo dirección en un sistema de contabilidad, este campo es requerido en las tablas Clientes, Pedidos, Envíos, Facturas y Cuentas por cobrar, en este caso se podría almacenar la dirección del cliente en la tabla Clientes, y hacer referencia a las demás tablas como una clave foránea.

Tercera Forma Normal (3FN)

La tercera forma normal (3FN) menciona que no deben existir dependencias transitivas entre las columnas de una tabla, esto quiere decir que las columnas que no forman parte de la clave primaria sólo dependen de una sola clave y no dependen de otras columnas que no sean la columna clave.

ID	nombre
1	Juan
2	Pedro
3	Karen
4	Alexandra
5	Raquel

servicios	clienteid	hora
1	1	10:30
1	2	12:30
2	2	12:30
3	3	18:00
4	4	16:00
4	5	10:00
5	5	10:00
3	5	10:00

ID	nombre
1	Corte de Cabello
2	Corte de Barba
3	Peinado
4	Uñas
5	Tinte

Figura 3.6 Tablas con atributos de una sola entidad

Cada una de las tablas en una base de datos debe contener información exclusivamente de una sola entidad que contiene un conjunto de atributos que son características de dicha entidad y no de ninguna otra (ver figura 3.6). Cada entidad debe tener un campo clave que es una referencia para todos los demás atributos.

Desnormalización

En algunos casos, es válido que las bases de datos no cumplan con las reglas de normalización. Cuando se realiza la normalización de una base de datos existe la necesidad de crear más tablas conforme se avanza en el proceso de normalización, sin embargo, al crear más tablas, se requiere mayor número de combinaciones para recuperar los datos, lo que provoca ralentizar las consultas. Con el propósito de mejorar el tiempo de respuesta en algunas consultas que son frecuentes, se puede tomar la decisión de no separar los atributos de una entidad.

La figura 3.7 muestra una tabla en la que una persona puede tener dos o más teléfonos. Si la persona tiene un solo número de teléfono, los demás campos quedan vacíos. Separar los números de teléfono en otra tabla, trae como consecuencia agregar una nueva dependencia, y generar más tiempo de búsqueda en las tablas, por lo que puede ser más sencillo dejar en una misma tabla varios números de teléfono. Aunque es una solución que rompe con las reglas de normalización, puede ser aceptable manejar la situación de esta forma.

id	nombre	telefono	telefono2
1	Juan	12121212	
2	Pedro	34343434	12345678
3	Karen	56565656	
4	Alejandra	78787878	
5	Raquel	90909090	87654321

Figura 3.7. Ejemplo de tabla no normalizada

Las reglas de normalización no consideran el rendimiento, al realizar el diseño de una base de datos, debe prevalecer la intuición y el sentido común que permitan valorar y mantener un equilibrio entre el tiempo de respuesta de las consultas y la integridad de los datos. La normalización de una base de datos se realiza por niveles, dadas las circunstancias se podría devolver una estructura de datos a una forma normal inferior (CUAIEDD/FCA-UNAM).

Algunos de los SGBD más comunes para bases de datos relacionales son MySQL, MariaDB, PostgreSQL, Microsoft SQL Server, Oracle, etc. Los SGBD por lo general pueden ser utilizados en modo gráfico y cada uno de ellos posee su propia interfaz, pero el soporte para todos los ambientes gráficos es un lenguaje estándar de consulta que se conoce como SQL (Structure Query Language).

El lenguaje SQL puede ser utilizado en línea de comandos lo que presenta algunas ventajas:

- Minimizar los riesgos de seguridad, al no utilizar interfaces gráficas.
- Cuando se conoce el lenguaje estándar SQL, se puede trabajar en cualquiera de los SGBD relacionales, pues el SQL es la base para cualquier plataforma.
- Por seguridad, los servidores dedicados a bases de datos no tienen instalados ambientes gráficos, en muchas ocasiones desde el servidor, el SQL es la única forma para tener acceso a la base de datos.
- El lenguaje SQL consume menos recursos que las SGBD con interfaces gráficas.

Modelo de base de datos de red

Un modelo de base de datos de red está formado por una colección de registros, los cuales están conectados entre sí por medio de enlaces creando una red. Cada registro es similar a una entidad como las empleadas en el modelo entidad-relación, contiene un atributo o valor único, que funciona como campo clave, así como un conjunto de datos relacionados, el enlace es la asociación entre dos registros exclusivamente, por lo que se denomina una relación binaria.

El modelo de base de datos de red admite que un registro se relacione con varios registros, por lo que a partir de un registro se pueden seguir diferentes caminos. El modelo de red es una relación muchos a muchos (N:N), pues a partir de un registro, se pueden relacionar más de un registro como se muestra en la figura 3.8.

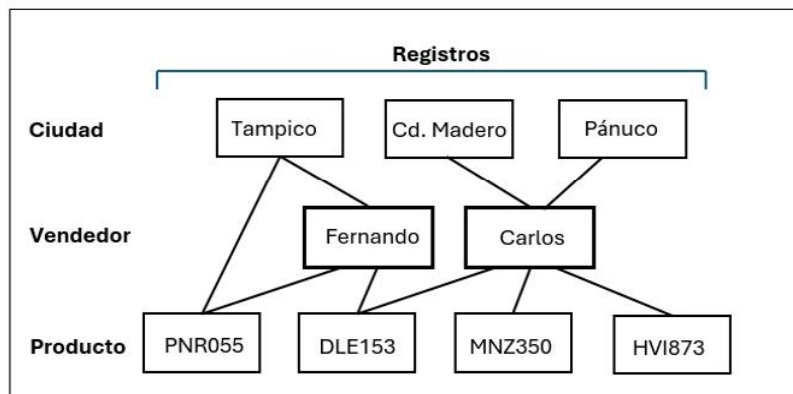


Figura 3.8 Modelo de base de datos de red

Modelo de base de datos multidimensional

El modelo de base de datos multidimensional se fundamenta en el modelo relacional, el cual tiene dos dimensiones (filas y columnas). La estructura multidimensional maneja N dimensiones, por lo que presenta mayor complejidad, aunque es fácil de mantener y entender pues los registros mantienen una estructura similar a las bases de datos relacionales.

Es un modelo de base de datos muy popular en el proceso analítico de transacciones en línea (On-Line Analytical Processing, Procesamiento analítico en línea). Una base de datos multidimensional se puede considerar como una ampliación de las dos dimensiones de una hoja de cálculo, esta ampliación adicional o nueva dimensión en los datos permite generar consultas más atractivas que involucren datos más complejos.

Para la realización de consultas, las bases de datos multidimensionales se caracterizan por una mayor versatilidad que las bases de datos relacionales. A partir de entradas de las bases de datos relacionales se crean cubos multidimensionales utilizando el lenguaje de consulta SQL.

Este tipo de base de datos presenta una solución en el campo de la Inteligencia de negocios (BI, Business Intelligence) cuyo objetivo es agilizar la consulta de grandes volúmenes de datos. Para este fin utiliza estructuras de datos multidimensionales llamados cubos OLAP, que extraen información de grandes bases de datos o sistemas transaccionales y presentan la información en forma de vistas para generar informes de ventas de negocios, marketing, informes de Dirección, etc.

La figura 3.9 muestra un cubo OLAP, donde la información de los productos y las ciudades en donde se venden dichos productos, se pudieran almacenar en una tabla, pero para agregar información adicional, por ejemplo, quién es el empleado encargado de distribuir el producto en la ciudad indicada se requiere la consulta de otra tabla.

Al consultar información de más de una tabla se generan cubos que permiten realizar consultas más complejas.

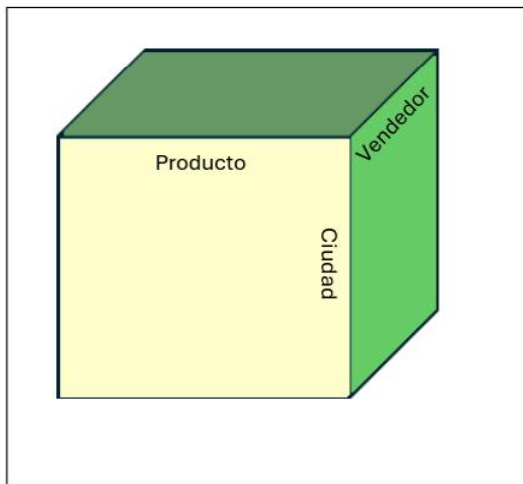


Figura 3.9 Modelo de base de datos multidimensional

Las bases de datos multidimensionales se caracterizan por facilitar un análisis muy útil para el negocio, que extrae datos de forma selectiva y realiza consultas de distintos tipos. Este tipo de modelos tiene una desventaja, pues no se puede modificar la estructura una vez creada, por lo que cuando se requiere un cambio, es necesario rediseñar nuevamente la estructura del cubo.

Modelo de base de datos orientado a objetos (OO)

Las bases de datos orientadas a objetos permiten al diseñador especificar tanto una estructura de objetos compleja como las operaciones que se pueden aplicar sobre dichos objetos. Estas bases de datos se pueden integrar directamente con aplicaciones desarrolladas con lenguajes de programación orientados a objetos como Java y C++. En una base de datos OO, la información se representa como objetos y se requiere un sistema manejador de base de datos orientado a objetos (ODBMS, Object Database Management System) para la manipulación de los objetos.

Los ODBMS necesitan manipular objetos y modelos de datos que permitan expresar su comportamiento, así como las relaciones entre ellos. Estos manejadores se fundamentan en los pilares del modelo orientado a objetos que son: abstracción, encapsulamiento, herencia y polimorfismo.

El modelo orientado a objetos utiliza objetos para representar el mundo real y sus problemas. Cuenta con 5 conceptos básicos que son: objeto, mensajes, clases, herencia y polimorfismo. Un objeto es una entidad en el sentido que lo manejan los gestores de bases de datos, se les llama “entidades” en bases de datos mientras que en la tecnología orientada a objetos se les llama “objetos”, pero en ambos casos son los sustantivos o sujetos del mundo real que son fundamentales para el planteamiento y solución de los problemas, los cuales pueden ser tangibles o intangibles.

Un objeto tiene un estado, un comportamiento y una identidad propia. Un mensaje permite la comunicación entre los objetos. Una clase es una estructura o plantilla del objeto que permite definir instancias del mundo real. La herencia permite definir atributos similares de dos o más objetos, manteniendo también atributos propios de cada objeto. El polimorfismo permite definir comportamientos diferentes de un objeto, aunque el nombre del comportamiento es el mismo, por ejemplo, el comportamiento “correr”, puede correr una persona, un tigre o un carro, dependiendo del ente u objeto que realice la acción, el comportamiento es diferente.

El modelo orientado a objetos encapsula el código y los atributos de un objeto en una entidad llamada objeto de manera que sólo pueden tener acceso a la información encapsulada objetos del mismo tipo. Para que el resto del sistema pueda tener acceso a la información del objeto se le envían mensajes. Un mensaje es una solicitud entre objetos en la que no se tienen detalles específicos de la implementación. La capacidad de modificar la estructura de un objeto sin afectar al resto del sistema es una de las principales ventajas del modelo orientado a objetos.

Aunque los modelos relacionales son líderes en el mercado, los modelos de bases de datos orientados a objetos cubren requerimientos tecnológicos y de información que los modelos relacionales no tienen. Por ejemplo, los sistemas CAD/CAM, multimedia, del medio ambiente y geográficos requieren de modelos de datos complejos difíciles de representar como registros en una tabla (CUAED-UNAM, 2017).

Modelo de base de datos documental

Una base de datos orientada a documentos (o base de datos documental) permite almacenar, recuperar y gestionar datos semiestructurados. Aunque cada base de datos documental difiere en detalles, comparten el principio de que los documentos encapsulan y codifican datos o información siguiendo algún formato estándar, entre las codificaciones utilizadas están XML, YAML, JSON, y BSON.

Los documentos dentro de una base de datos documental son similares a los registros en una base de datos relacional, pero son menos rígidos como se muestra en la figura 3.10, pues no se requiere que se ajusten a un esquema estándar, ni deben tener las mismas secciones o atributos.

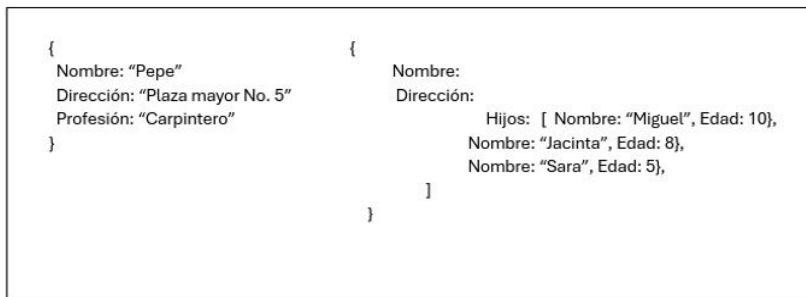


Figura 3.10 Registros con atributos diferentes en una base de datos documental

Los registros en las bases documentales no tienen un esquema específico y sus columnas no son similares por lo que no pueden almacenarse en tablas como las bases de datos relacionales. Las bases de datos documentales son llamadas bases de datos NoSQL. “Todas las operaciones en SQL consumen y retornan relaciones de acuerdo con el álgebra relacional. Este uso de las relaciones proporciona elegancia y simplicidad, pero introduce un conjunto de limitaciones” (SARASA, p. 23).

La primera actividad que se debe realizar cuando se almacenan datos en una base de datos relacional, es definir la estructura de las tablas, es decir qué columnas tiene cada tabla y tipo de datos que tiene cada columna. Las bases de datos documentales, al no tener un esquema fijo, permite agregar campos libremente a los registros de la base de datos sin necesidad de redefinir la estructura lo cual es útil cuando se trata de campos de datos personalizados y no uniformes.

Aunque una base de datos documental podría gestionar datos con diferentes formatos y sin un esquema coherente, por lo general para optimizar la gestión de la base de datos, se utiliza un formato de archivo para documentos, y una estructura fija para los datos, de esta forma se procesan mejor las consultas.

3.3 Construcción del sistema

Existen muchos beneficios de construir un software siguiendo un conjunto de prácticas efectivas. “No da igual cómo esté construido” (Sánchez, p. 224). Algunos beneficios son: produce código más fácil de entender, facilita el mantenimiento del sistema, acorta el tiempo de desarrollo del software, se reducen los errores y facilita la depuración del programa.

Prácticas efectivas en la construcción de software:

- Agregar comentarios que documenten el código.
- Utilizar variables locales donde sea posible en vez de variables globales.
- Definir nombres de variables que auto documenten el código.
- Dividir el programa en módulos de tamaño que sean manejables.
- Desarrollar y alimentar un depósito de componentes para almacenar los componentes de software que hayan sido desarrollados y que podrían ser utilizados en la construcción de nuevo código.
- Reutilizar código donde sea posible.
- Escribir una instrucción por línea.
- Optimizar el código desarrollado.
- Validar la información introducida por el usuario en los campos de entrada.
- Realizar pruebas de escritorio a cada función realizada.
- Dejar sangría cuando al codificar instrucciones de control para indicar visiblemente cuáles instrucciones se realizan dentro de un bloque.

El proceso de codificación del software

En el proceso de codificación, el Ingeniero de software debe llevar un control de los diversos cambios que se realizan en el código fuente. Existen herramientas para llevar el control de un proyecto de software en desarrollo, una de estas herramientas son los sistemas de control de versiones. Un método que muchas personas eligen para controlar las versiones es copiar los archivos en otro directorio y tal vez agregar alguna marca temporal que permita identificar cada versión. Este método es muy común y simple, pero también muy propenso a errores, pues es fácil confundir cuál es el directorio en el que se está trabajando y accidentalmente sobrescribir el archivo equivocado (Chacon, p.10).

Una versión, revisión o edición de un producto de software, es el estado en el que se encuentra el mismo en un momento determinado de su desarrollo o modificación. Y se llama control de versiones a la gestión de los cambios que se realizan sobre los elementos de algún producto de software o su configuración. Aunque el control de versiones de un sistema puede llevarse a cabo sin herramientas especializadas, es recomendable disponer de herramientas para el control de versiones que faciliten la gestión.

Un sistema de control de versiones (VCS, Version Control System) es una herramienta que registra los cambios en un archivo o en un conjunto de archivos, para que posteriormente se puedan recuperar versiones específicas del archivo. Un VCS ofrece herramientas de colaboración para compartir e integrar dichos cambios con otros usuarios que también utilicen la herramienta de control de versiones. Debido a que opera a nivel de archivos, un VCS monitoriza la adiciones, eliminaciones y modificaciones en las líneas de texto que contiene ese archivo.

Algunas de las funciones más comunes de un VCS son:

- Permite regresar archivos seleccionados o incluso el proyecto completo a un estado previo. Una vez que un VCS ha empezado a monitorizar un sistema de archivos de código fuente, conserva el historial de cambios y el estado del código fuente durante el historial de un proyecto. De esta forma, existe la posibilidad de "deshacer" o revertir un proyecto de código fuente a un estado conocido reciente. Si se detecta un fallo en una aplicación en vivo, el código puede revertirse rápidamente a una versión estable conocida.
- Permite comparar cambios entre dos archivos. Si un archivo ha sido modificado y se tiene una versión previa del mismo, es posible comparar cada línea de código que ha sido agregada, eliminada o modificada.
- Permite reconocer quién realizó la última modificación en un archivo que pudiera estar causando conflicto.
- Permite recuperar archivos perdidos o borrados. Si por equivocación se ha borrado un archivo, éste puede ser recuperado porque Git guarda el historial de los cambios de los archivos que tiene bajo su custodia.
- Permite mantener copias de seguridad externas del código fuente. Al usar un VCS de forma colaborativa, se debe crear una instancia remota del VCS para compartir cambios entre desarrolladores. Esta instancia remota del VCS se puede alojar de forma externa con un tercero de confianza como Bitbucket o GitHub. A continuación, se convierte en una copia de seguridad externa. En un caso desafortunado (como el del robo de un portátil), la instancia remota del VCS conservará una copia del código fuente.

Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (Por ejemplo, para un cliente específico, alguna funcionalidad específica, algún formato o plantilla específica de la aplicación). También un diseñador gráfico se puede beneficiar con la utilización de esta herramienta, puede mantener cada versión de una imagen con esta herramienta. Algunas herramientas de Control de versiones en el mercado: CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar, PlasticSCM, SCCS, Mercurial, Perforce y Git.

Un sistema de control de versiones es una valiosa herramienta con numerosos beneficios para un flujo de trabajo de equipos de software de colaboración. Cuando un proyecto de software tiene más de un desarrollador manteniendo archivos de código fuente, es indispensable utilizar un VCS.

El sistema de control de versiones GIT

Git es un sistema de control de versiones open source y libre que administra desde pequeños hasta grandes proyectos de software con rapidez y eficiencia. Es fácil de aprender y permite seguir la pista de un proyecto con un eficiente desempeño. Una característica importante de Git es la utilización de un sistema distribuido, lo que permite obtener una copia del proyecto completo en un almacenamiento local ("clone"). Permite múltiples respaldos, cada desarrollador tiene una copia completa del sistema en el servidor principal, cada una de estas copias, puede utilizarse para reemplazar el sistema en el servidor principal, en caso de algún conflicto o de llegar a un sistema corrupto.

Git también permite la resolución de conflictos entre diferentes versiones de un mismo archivo. Durante el ciclo de vida de un proyecto de software desarrollado por un equipo de trabajo, es muy probable que los miembros del equipo tengan la necesidad de realizar cambios en el mismo archivo de código fuente al mismo tiempo. Un VCS monitoriza y ayuda a resolver los conflictos entre los diferentes archivos de los desarrolladores. Estas operaciones de resolución de conflictos dejan un registro de auditoría que ofrece información sobre el historial de un proyecto.

Los sistemas de control de versiones como Git poseen un modelo de ramificación en la que pueden tener múltiples ramas locales independientes, cada una de estas ramas, contiene una versión específica del software, es decir, aunque el proyecto sea el mismo, los archivos contenidos en las ramas pueden variar o incluso algunos pueden ser borrados o actualizados. Cada rama puede ser creada y modificada por alguno de los integrantes del equipo. Se crea una ramificación para trabajar una idea, y regresar al punto donde se inició la ramificación, aplicar los cambios, experimentar y si se considera conveniente, mezclar las versiones.

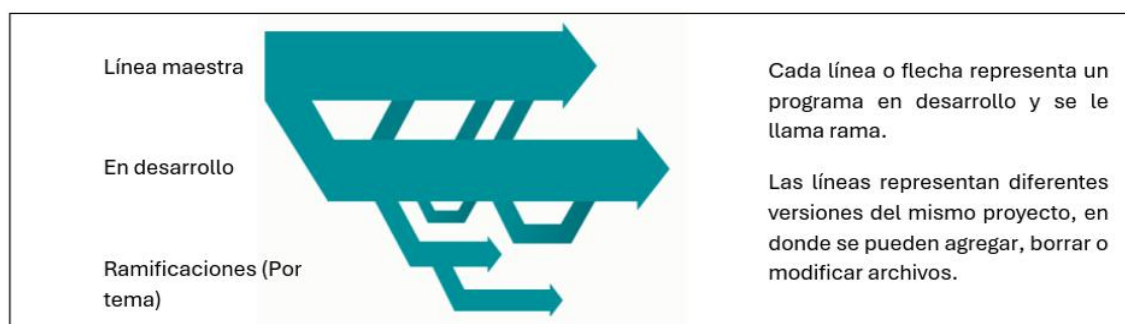


Figura 3.11. Ramificaciones en Git

La figura 3.11 muestra la gestión de las ramas en Git. Un proyecto en Git es representado por una línea, denominada línea maestra, y contiene una copia del proyecto principal. De la rama maestra se puede obtener una copia o clone del proyecto, la cual puede ser modificada por

algunos de los participantes, esta copia se puede utilizar para realizar los cambios con comodidad, sin afectar el proyecto principal. También, de acuerdo con las características del proyecto se podrían identificar funcionalidades o módulos del programa que pudieran ser copiados y puestos en nuevas ramificaciones clonando parte de los archivos para ser gestionados en forma independiente. Una vez realizados los cambios y si se considera conveniente se integran los cambios a la rama en desarrollo, o también se pueden integrar los cambios a la rama principal (línea maestra).

Cuando se utiliza Git, la mayoría de las operaciones se realizan localmente, lo que permite ganar ventaja en velocidad respecto a los sistemas centralizados que constantemente tienen que comunicarse con un servidor central. Git fue escrito en lenguaje C y tiene un manejo efectivo de grandes espacios de almacenamiento, reduciendo el tiempo de ejecución asociado con los lenguajes de alto nivel.

Flujos de trabajo distribuidos en Git

La naturaleza distribuida de Git proporciona flexibilidad en la forma de colaborar en proyectos, cada desarrollador cuenta con un repositorio local (en su computadora) y puede compartir su código fuente actualizando el repositorio central o remoto (repositorio público en el cual otros pueden basar su trabajo y al cual pueden contribuir). Esto abre un enorme rango de posibles flujos de trabajo para administrar un proyecto y/o un equipo de trabajo.

Un flujo de trabajo en el desarrollo de software es la forma de gestionar los procesos y tareas que se llevan a cabo para construir el sistema, define el orden en que ocurren los eventos, la forma en que interactúan las personas y la secuencia en que se deben actualizar las versiones del código fuente. En el flujo de trabajo también se deben considerar los roles de los participantes, cada uno de los participantes debe tener funciones definidas, así como responsabilidades. Algunos flujos de trabajo para trabajar en un proyecto de desarrollo de software son: el flujo de trabajo centralizado, flujo de trabajo administrador-integración y flujo de trabajo dictador-tenientes (Chacon, p. 128).

Flujo de trabajo centralizado

En este esquema existe un repositorio central, el cual acepta código de los diferentes participantes del proyecto, y todos los participantes se sincronizan con él. Esto significa que, si dos desarrolladores obtienen una copia del código desde el repositorio central y ambos hacen cambios, sólo un desarrollador puede actualizar el repositorio central sin inconvenientes, el segundo desarrollador debe fusionar su trabajo con el trabajo realizado por el primer desarrollador para no sobrescribir los cambios ya realizado. Una segunda posibilidad, es que los participantes trabajen en módulos independientes, por lo que ambos participantes pueden actualizar el código, ver figura 3.12 (Chacon, p. 128).

Git no permite a los usuarios sobrescribir código, los integrantes del equipo se deben de poner de acuerdo para coordinar y producir una nueva versión del código. El flujo de trabajo centralizado es muy atractivo debido a que es sencillo y muchos desarrolladores están familiarizados con su filosofía, por lo que se sienten cómodos utilizando este modelo.

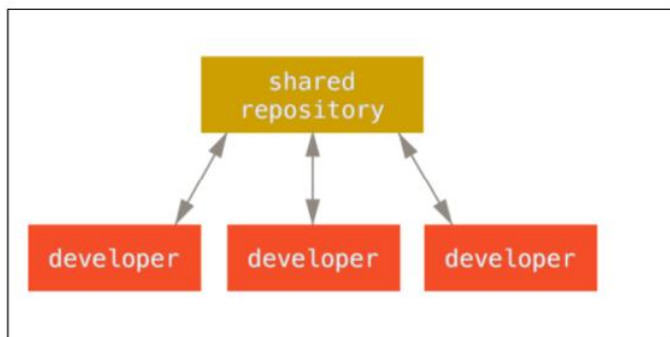


Figura 3.12. Flujo de trabajo centralizado

Además de utilizarse en equipos pequeños de trabajo, también se adapta para equipos de trabajo grandes y medianos. El modelo de ramificación de Git permite que cientos de trabajadores trabajen exitosamente en un mismo proyecto gestionando docenas de ramas simultáneamente (figura 3.13).

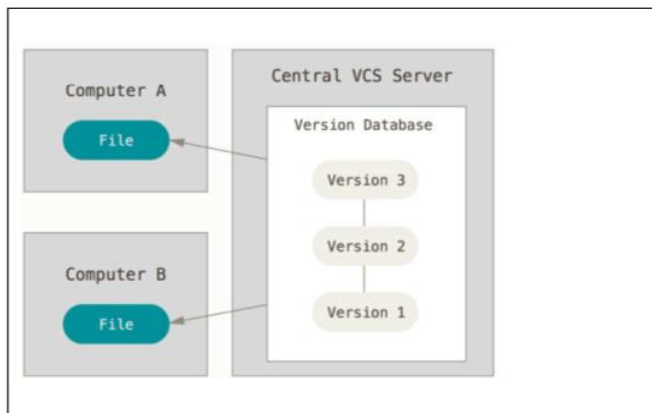


Figura 3.13. Repositorio central y repositorios locales

En el flujo de trabajo centralizado, todos los desarrolladores tienen permiso de lectura y escritura en el repositorio central. El sistema de control de versiones se encarga de advertir a los desarrolladores cuando hay un archivo que ha sido modificado y deben ser revisados antes de fusionar las diferentes copias.

Flujo de trabajo gestor de integración

Git permite múltiples repositorios remotos, cada uno de los cuales es gestionado por un desarrollador (Chacon, p. 129). En cada repositorio remoto es posible tener un flujo de trabajo donde cada desarrollador tiene acceso de escritura a su propio repositorio público y acceso de sólo lectura a todos los demás repositorios. Este escenario a menudo incluye un repositorio canónico o central que representa el proyecto final.

Para contribuir al proyecto, el desarrollador debe crear su propio clon público del proyecto y actualizarlo con las modificaciones realizadas (“push your changes to it”).

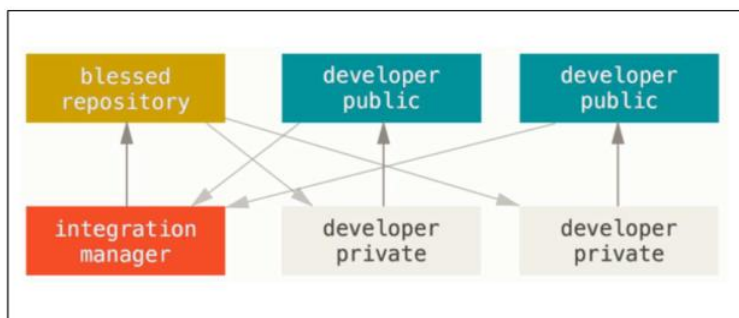


Figura 3.14 Flujo de trabajo gestor de integración

Una vez que el desarrollador ha actualizado su código y desea compartirlo para contribuir con el proyecto principal. Éste envía una solicitud al administrador del proyecto principal (Gestor de integración) para solicitar que agregue los cambios que realizó en el código (“pull in your changes”). Entonces el gestor de integración puede agregar el repositorio del desarrollador como un repositorio remoto, probar los cambios localmente, mezclar los cambios en su rama, actualizar su propio repositorio y enviar las modificaciones al repositorio final (ver figura 3.14).

El flujo de trabajo gestor de integración es muy común cuando se utilizan herramientas basadas en hub como GitHub y GitLab, en las cuales es sencillo bifurcar un proyecto y crear una rama independiente en donde se pueden actualizar los cambios realizados en las bifurcaciones en las que están trabajando cada uno de los participantes del proyecto. Una de las principales ventajas de este enfoque es que los desarrolladores pueden continuar con su trabajo y el gestor de integración del proyecto principal, puede obtener las modificaciones realizadas por los desarrolladores del proyecto en cualquier momento. Los contribuidores o desarrolladores no tienen que esperar a que se integren sus cambios en el proyecto, ellos pueden continuar trabajando a su propio ritmo.

Flujo de trabajo dictador-tenientes

Cuando los proyectos son de mayor tamaño, con cientos de integrantes en el equipo de desarrollo, un flujo de trabajo similar al que se utiliza para desarrollar el kernel de Linux es efectivo (Chacon, p. 130). El flujo de trabajo dictador-tenientes es una variante del flujo de trabajo repositorios múltiples. En este modelo, participan varios gestores de integración (también llamados tenientes), cada uno de los cuales tiene a su cargo ciertas partes del repositorio. Todos los tenientes tienen un gestor de integración conocido como dictador benevolente.

El dictador benevolente es el encargado de actualizar los cambios desde su directorio o repositorio hacia los repositorios de referencia, desde los cuales todos los colaboradores obtienen las actualizaciones (ver figura 3.15), el proceso trabaja de la siguiente manera:

1. Los desarrolladores trabajan en su propia rama llamada rama temática, se define de esta manera porque el desarrollador está trabajando el código de un tema o funcionalidad específica. Una vez que el desarrollador realiza las modificaciones necesarias, envía su trabajo a la rama maestra. La rama maestra es el repositorio de referencia, el cual es actualizado por el dictador.
2. Los tenientes mezclan las ramas temáticas de los desarrolladores en su propia rama maestra.
3. El dictador mezcla las ramas maestras de los tenientes en su rama maestra (rama maestra del dictador).
4. Por último, el dictador actualiza el código desde su rama maestra hacia el repositorio de referencia central referencia.

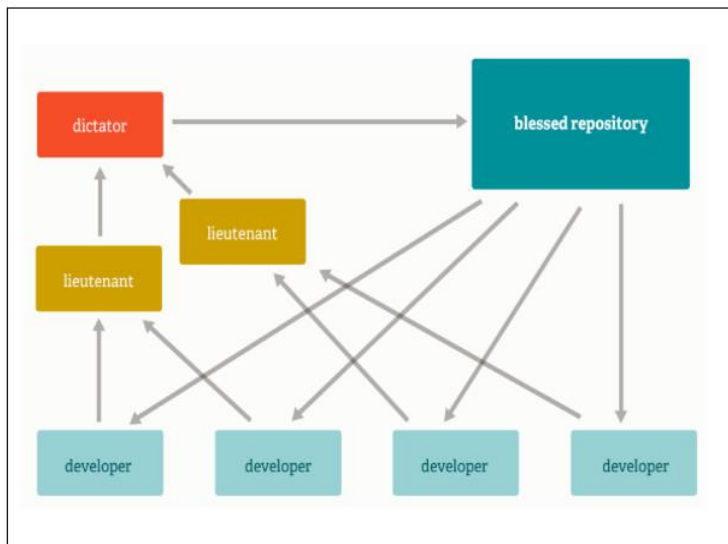


Figura 3.15. Flujo de trabajo dictador-tenientes

Los sistemas de control de versiones como Git proporcionan herramientas para crear ramas que permitan gestionar las versiones de un código fuente. Eventualmente, las ramas deben mezclarse para fusionar el código. Muchos equipos de desarrollo de software emplean una cantidad de tiempo excesiva para copiar y mezclar un intrincado conjunto de ramificaciones (Fowler, 2024). Existen dos equipos de trabajo que facilitan a los participantes en el desarrollo de software el manejo efectivo de las ramas, concentrándose en la integración del trabajo de múltiples desarrolladores y encaminando los esfuerzos hacia la producción y liberación de nuevas versiones del software.

Para el mantenimiento y gestión del código fuente producido por los participantes en el desarrollo de software, se pueden identificar dos equipos principales:

1. El equipo de integración de código. Son los encargados de combinar el trabajo desarrollado por múltiples programadores en un todo coherente. Revisan los archivos de cada una de las ramificaciones, analizan las diferencias y características de cada uno de los archivos y toman decisiones para la integración del código.
2. Equipo de producción de software. Quienes utilizan las ramificaciones para codificar versiones específicas de un proyecto agregando funcionalidades o características especiales. Utilizan las ramas secundarias en Git para ayudar a gestionar la ruta desde la integración de un código base hasta llegar al desarrollo de un código fuente que se ejecuta en producción.

Si varios desarrolladores trabajan en el mismo código base, es muy difícil editar los mismos archivos, por lo que cada desarrollador debe copiar el código base en un conjunto de archivos que pueda gestionar en forma personal. Para la coordinación de los integrantes del equipo, se deben definir roles que especifiquen las funciones y responsabilidades de cada uno de los integrantes permitiendo el mantenimiento y gestión de código en forma eficiente.

Además de las herramientas tecnológicas y computacionales indispensables para trabajar en equipo, es necesario considerar la importancia de la comunicación y el respeto entre los integrantes del equipo. Cuando un equipo está bien integrado y liderado, el trabajo fluye positivamente entre los participantes a través del compañerismo, intercambio de conocimiento y apoyo mutuo; porque los problemas que se presentan se resuelven favorablemente.

Cuando se inicia un proyecto de desarrollo de software, se fijan objetivos y metas por cumplir. Un equipo exitoso, además de desarrolladores de software requiere de un project manager, analistas comerciales, diseñadores UX entre otros. Sin importar el rol, todos los participantes deben involucrarse y comprometerse con los objetivos y metas del proyecto.

Tema 4:

Pruebas e implantación

Una vez que el sistema ha sido desarrollado, inicia la etapa de pruebas. Una prueba de software es la actividad de evaluar y verificar que el sistema desarrollado cumple los requisitos solicitados por el usuario. Para lograr este objetivo, se deben diseñar y ejecutar un conjunto de pruebas que permitan evaluar la calidad del sistema. La realización de las pruebas no es una tarea fácil, pues un programa puede ejecutarse un infinito número de veces desde muy diversos contextos, cada una de estas ejecuciones genera diferentes condiciones que en algún momento determinado podrían producir un error. Para la verificación del comportamiento de un programa se debe seleccionar un conjunto finito de casos de prueba que sean representativos de las condiciones más generales de las rutas de ejecución de un programa.

Como parte de la etapa de pruebas se debe desarrollar un documento llamado plan de pruebas, que es un documento eje que sirve de guía para la aplicación de las pruebas. El plan de pruebas describe el alcance, enfoque, recursos y calendarización de las actividades de prueba. En el plan de pruebas se especifican los elementos y características a probar, se identifican las tareas de prueba a desarrollar, los responsables de cada tarea, el tiempo que se debe dedicar a cada una de las tareas, así como los riesgos asociados. Un caso de prueba es un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y postcondiciones de ejecución, desarrollados para un objetivo en particular de condición de prueba, tal como para ejercitar o recorrer una ruta de un programa en particular o para verificar el cumplimiento de un requisito específico. El caso de prueba debe ser ágil, claro y entendible de forma que sea sencillo para otra persona lograr reproducirlo sin mucho esfuerzo; a mayor número y calidad de casos de prueba aplicados a un software, el software adquiere más valor y mejora su calidad.

Competencia específica: Evalúa el software construido a partir de las pruebas e implementación realizada.

Introducción

Las pruebas de software constituyen uno de los procesos más importantes en el desarrollo de software porque permiten verificar si una aplicación cumple con los objetivos planteados por el usuario, verificar si fue construido de acuerdo con estándares, detectar puntos críticos para que sean corregidos, proporcionar información importante sobre la calidad del producto, evitar pérdidas económicas y de información entre otros. Aunque son muchos los motivos para realizar pruebas en un sistema, en ocasiones no se cuenta con una estrategia adecuada para su realización. En este tema, se abordan algunos aspectos importantes para el diseño de un plan de pruebas.

Existe una gran variedad de pruebas que se pueden aplicar en el desarrollo de un sistema. No es posible aplicar todas las pruebas de software para la verificación de un sistema, existen limitaciones de tiempo y recursos por lo que es necesario especificar cuáles son las pruebas que se aplicarán a un sistema, además no todos los tipos de pruebas son aplicables en todos los casos.

4.1 Diseño de caso de prueba

El diseño de casos de prueba es una estrategia que busca producir un número de casos de prueba de tal forma que garantice la efectividad de la prueba. Su objetivo es conseguir la confianza aceptable de encontrar los errores existentes en el software, sin consumir una cantidad excesiva de recursos. Las pruebas de software representan una cantidad importante de los costos totales del desarrollo de proyectos.

La prueba de software es el proceso de comparar “lo que es” con “lo que debería de ser”, esto se refiere a comparar los resultados obtenidos del desarrollo del sistema contra la especificación de los requerimientos de software; se logra mediante la ejecución de un conjunto de casos de prueba que garanticen que se cumplen los requisitos especificados en el análisis del sistema. Cada caso de prueba contiene las descripciones de qué se va a probar, cómo se va a probar y el conjunto de datos necesarios para ejecutar cada caso de prueba.

Enfoques para los casos de prueba

Cuando se diseñan los casos de prueba, se pueden enfocar en diferentes aspectos del software, en este caso existen dos tipos distintos:

- **Pruebas de caja negra.** Se enfoca en evaluar **qué es lo que hace el software**, sin tomar en cuenta cómo se realiza el proceso. Es decir, se enfocan en los resultados o salidas producidas por el sistema. En las pruebas de caja negra, se verifica que el

elemento que se está probando, cuando se le dan las entradas apropiadas, produce los resultados esperados. Los datos de prueba se eligen de acuerdo con las especificaciones del problema y con el propósito de verificar que el programa se ejecuta adecuadamente.

- **Pruebas de caja blanca.** Se enfoca en revisar **cómo se realiza el software**, es decir, revisa cuál es la lógica del programa, el procedimiento que se sigue, el algoritmo, la secuencia de pasos y los diferentes caminos que se siguen a la hora de la ejecución del software. En las pruebas de caja blanca, se revisa en detalle el flujo lógico de las decisiones y ciclos que se siguen dentro del código fuente buscando ejecutar por lo menos una vez, cada uno de los caminos básicos. La prueba de caja blanca se basa en el diseño de casos de prueba que utilizan las estructuras de control que forman parte del diseño procedimental para generar los casos de prueba requeridos.

4.2 Pruebas de componentes

En ingeniería de software, el término componente se refiere a cualquier elemento de software, que puede ser probado y gestionado en forma independiente. Los componentes de software “son entidades reutilizables que no requieren conocimiento del software que los usa” (Braude, p. 262), los componentes pueden ser objetos en el sentido usual de la orientación a objetos, con la excepción de que satisfacen guías adicionales dirigidas a hacerlos autocontenidos.

“Las pruebas son responsables de más de la mitad del tiempo dedicado a los proyectos”, la ingeniería de software enfatiza la importancia de dedicar tiempo y esfuerzo en la aplicación de pruebas de software y de esta forma cumplir con las expectativas del cliente. Si no se encuentran errores existentes en el sistema durante la etapa de pruebas los resultados pueden ser catastróficos, “la recompensa por encontrar un defecto pronto en el proceso es al menos un ahorro de diez veces comparado con detectarlo en la etapa de integración – o peor aún – después de la entrega” (Braude, p. 394).

Los programas de software son extensos y complejos, Dijkstra señaló que no existe una mente humana que sea capaz de memorizar un programa de computadora (Sánchez, p. 232). Los programas de computadora son demasiado grandes para que una sola persona, por las limitaciones inherentes a la mente humana, sea capaz de retener con precisión todos los elementos que lo conforman.

Por esta razón, los desarrolladores deben organizar el código de forma que éste permita centrarse en diferentes partes en cada momento, pues es imposible pensar en todo el programa a la vez (“divide y vencerás”). De este principio surge la necesidad de dividir o modularizar el código fuente.

Se denomina prueba de componente a la prueba de un módulo o la prueba de una clase que puede probarse y gestionarse de forma independiente. Entre las principales características de las pruebas de componentes es precisamente su gestión en forma aislada e independiente.

4.3 Pruebas del sistema

Existe una gran variedad de pruebas de software. Algunos ejemplos son:

- Pruebas de unidad
- Prueba del camino básico
- Prueba de integración
- Prueba de interfaz
- Prueba del sistema
- Prueba de regresión
- Prueba de aceptación
- Prueba de instalación

Pruebas de unidad

Las pruebas de unidad son las primeras pruebas que se aplican al sistema. En general, las funciones son las partes más pequeñas a las que se aplican las pruebas de unidad (Braude, pp. 404-405). Las pruebas de unidad deben cumplir las siguientes características:

- Las pruebas de unidad deben ser aleatorias y evitar muestras sesgadas.
- Decidir la filosofía de las pruebas de unidad. El primer paso es decidir “qué unidades” deben probarse y quién lo hará.
- Para proyectos orientados a objetos, la organización común para las pruebas de unidad es probar los métodos de una clase, después las clases de cada paquete y luego los paquetes como un todo.
- De preferencia, la ejecución de las pruebas de unidad debe realizarlas alguien diferente al desarrollador.
- Decida cómo se documentarán las pruebas de unidad. La documentación de una prueba de unidad consiste en los procedimientos de prueba, los datos de entrada, el código que ejecuta las pruebas y los datos de salida.

- Determinar el alcance de las pruebas de unidad. Dado que las pruebas a un sistema son infinitas, debe definirse el alcance de las pruebas. Por ejemplo, si una aplicación bancaria consiste en retiros, depósitos y consultas, las pruebas de unidad podrían especificar que cada método debe probarse igual cantidad de datos ilegales, de frontera y legales. O tal vez que los métodos de retiro y depósito se prueben 3 veces más que los de consulta, etc.
- Decidir cómo y dónde obtener los datos para las pruebas. Se deben recopilar datos de entrada legales, frontera e ilegales, así también datos generados en forma aleatoria. Cuando es posible, se usan herramientas que generan datos de entrada para las pruebas analizando el código fuente y detectando las fronteras y ramificaciones de los datos.
- Estimar los recursos requeridos. Se identifican persona-mes, así como la duración requerida para realizar las pruebas de unidad. La fuente más importante para la estimación son los datos históricos.
- Registrar el tiempo, cantidad de errores, tipo y fuente. Los ingenieros que participan en el proyecto documentan el tiempo dedicado en las pruebas de unidad, cantidad y tipo de errores. Los datos obtenidos se utilizan para evaluar el estado de la aplicación, pronosticar la calidad y fecha de terminación del producto.

Prueba del camino básico

La prueba del camino básico es una prueba de caja blanca que utiliza las estructuras de control de los procedimientos para derivar casos de prueba que permitan validar los diferentes flujos por los que puede pasar un programa durante su ejecución (caminos básicos). Mediante la prueba del camino básico el ingeniero de software puede obtener casos de prueba que:

- Garanticen que se recorran por lo menos una vez todos los caminos independientes de cada programa.
- Ejerciten todas las decisiones lógicas en las vertientes falsa y verdadera.
- Ejecuten todos los ciclos en sus límites operacionales.
- Ejerciten las estructuras internas de datos para asegurar su validez.

La prueba del camino básico es una técnica que obtiene una medida de la complejidad lógica de su diseño y utiliza esta medida como una guía para la definición de un conjunto básico de casos de prueba. A la complejidad lógica de un programa, también se le denomina complejidad ciclomática $V(G)$.

Para iniciar la prueba del camino básico se utiliza la notación de grafo de flujo, cuyos componentes son:

Nodo. Representa una o más secuencias procedimentales, su notación es un círculo.

Arista. Representa el flujo de control, gráficamente es una flecha.

Región. Las regiones son áreas delimitadas por las aristas y nodos.

Existen dos tipos de instrucciones en el grafo, aquellas instrucciones que siguen un único camino posible, como las instrucciones de asignación y las instrucciones de lectura o escritura de datos. También existen las instrucciones que generan dos o más caminos posibles, como el caso del **if**, **for** y **case** (Figura 4.1).

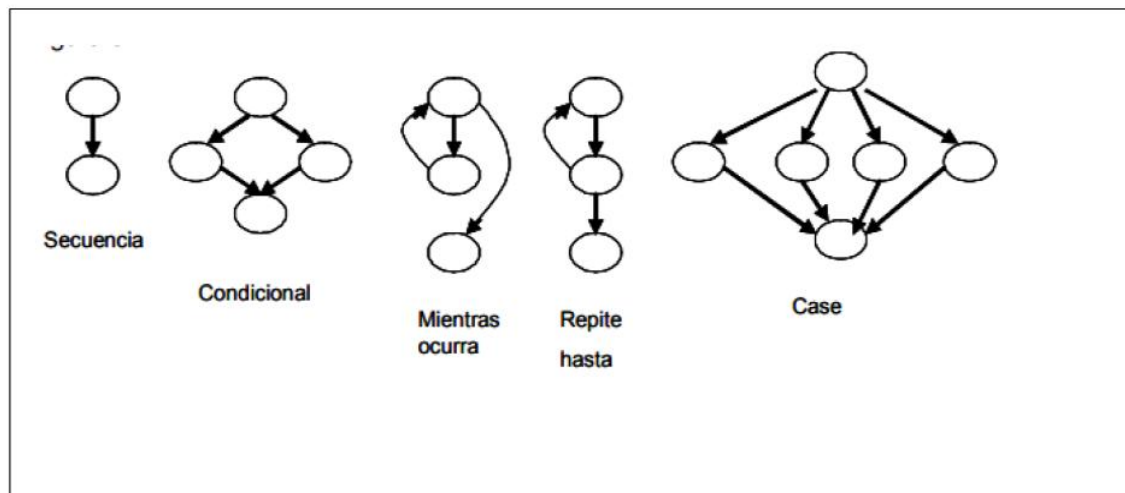


Figura 4.1. Grafo de flujo de las instrucciones de control básicas en programación

Pasos para realizar la prueba del camino básico:

1. Analice el código fuente, revisando que haya una sola instrucción por línea
2. Numere las líneas de código fuente
3. Represente el código fuente en un grafo de flujo
4. Calcule la complejidad ciclomática $V(G)$
5. Prepare un caso de prueba para cada uno de los caminos básicos, el número (G)

A continuación, se muestra un ejemplo para llevar a cabo la prueba del camino básico.

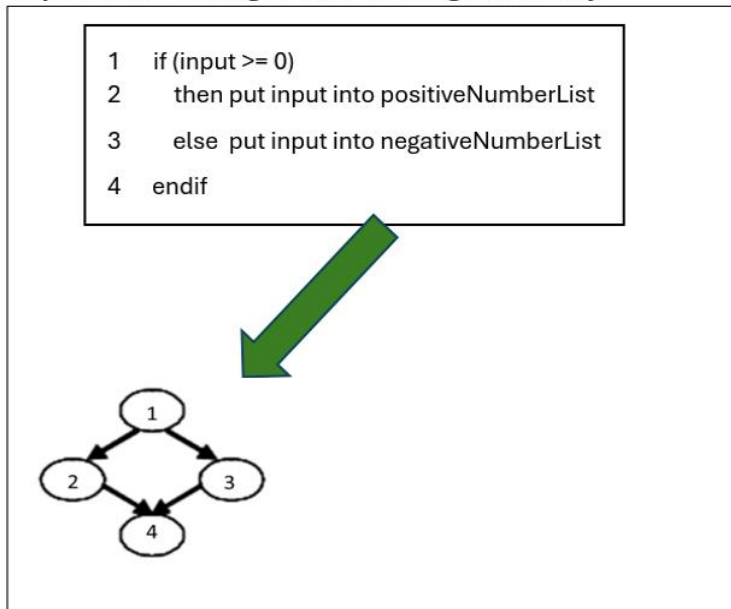
1. **Analice el código fuente revisando que haya una sola instrucción por línea.** En caso de existir dos instrucciones en una sola línea edite el código fuente para respetar esta regla, asegúrese de tabular las instrucciones que están dentro de un ciclo o condición para que de forma visual se observen las instrucciones dentro del ciclo. También debe mostrarse el inicio y fin de un ciclo o condicional.


```
if (input >= 0)
  then put input into positiveNumberList
  else put input into negativeNumberList
endif
```

2. **Numere las líneas del código fuente.** Es importante seguir un orden y la misma organización en el código fuente para facilitar realizar la prueba del camino básico, por ejemplo, poner la instrucción IF en un renglón, y en el siguiente renglón, la instrucción THEN, en todo el código donde se presente esta instrucción.

```
1 if (input >= 0)
2   then put input into positiveNumberList
3   else put input into negativeNumberList
4 endif
```

3. **Represente el código fuente en un grafo de flujo**



4. **Calcule la complejidad ciclomática $V(G)$**

$V(G)$ = Complejidad ciclomática

$$V(G) = e - n + 2$$

Donde: **e** representa el número de aristas

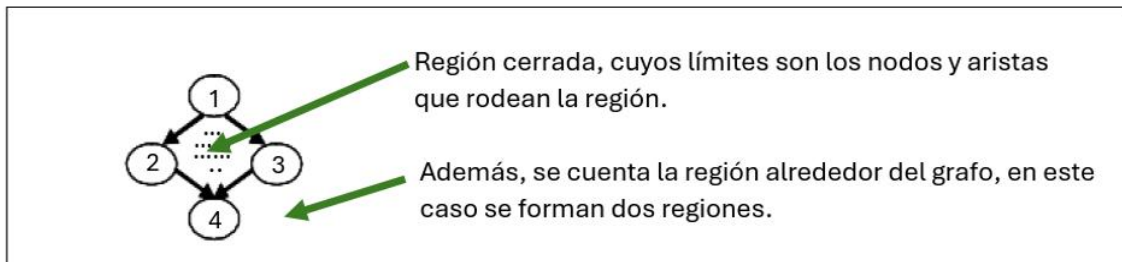
n representa el número de nodos

$$V(G) = 4 - 4 + 2$$

$$V(G) = 2$$

Otra forma de calcular la complejidad ciclomática es con el grafo de flujo. Al diseñar el grafo de flujo a partir del código fuente del programa, la unión de los nodos y las aristas forman regiones cerradas. Para este ejercicio, se cuentan el número de regiones que se forman y se suma una región más que se refiere a la región abierta alrededor del grafo de flujo. Cada región representa un diferente camino que puede ser recorrido en el programa.

$$V(G) = \text{Número de regiones cerradas en el grafo} + 1$$



5. Prepare por lo menos un caso de prueba para cada uno de los caminos básicos

$V(G) = \text{Número de regiones cerradas} + 1$

Caminos básicos:

- a) 1, 2, 4
- b) 1, 3, 4

Caso de prueba a): input = 5
Caso de prueba b): input = -214

El caso de prueba a) representa el camino cuando en la instrucción condicional entra al IF-THEN, porque la condición se cumple. El caso de prueba b) representa el camino cuando la condición no se cumple, por lo que se recorre el camino por el ELSE.

Dibujar el grafo de flujo permite visualizar de manera más clara cuáles son los diferentes caminos básicos que existen en el recorrido del grafo.

Pruebas de integración

Las aplicaciones de software son complejas, por lo que deben construirse en partes que se desarrollan por separado. La integración se refiere al proceso de ensamble de estas partes. En la construcción de software con frecuencia se deben integrar unidades parcialmente construidas, aunque este proceso tiene el inconveniente de trabajar con unidades incompletas, tiene la ventaja de ejercer integración desde el inicio del proceso de desarrollo.

Debido a que las interfaces de usuario se deben construir y probar en algún momento, se deben construir tan pronto como sea posible para que las pruebas de la aplicación puedan realizarse a través de ellas. Una alternativa es construir interfaces temporales para utilizarlas durante las pruebas de integración (Braude, pp. 439-440).

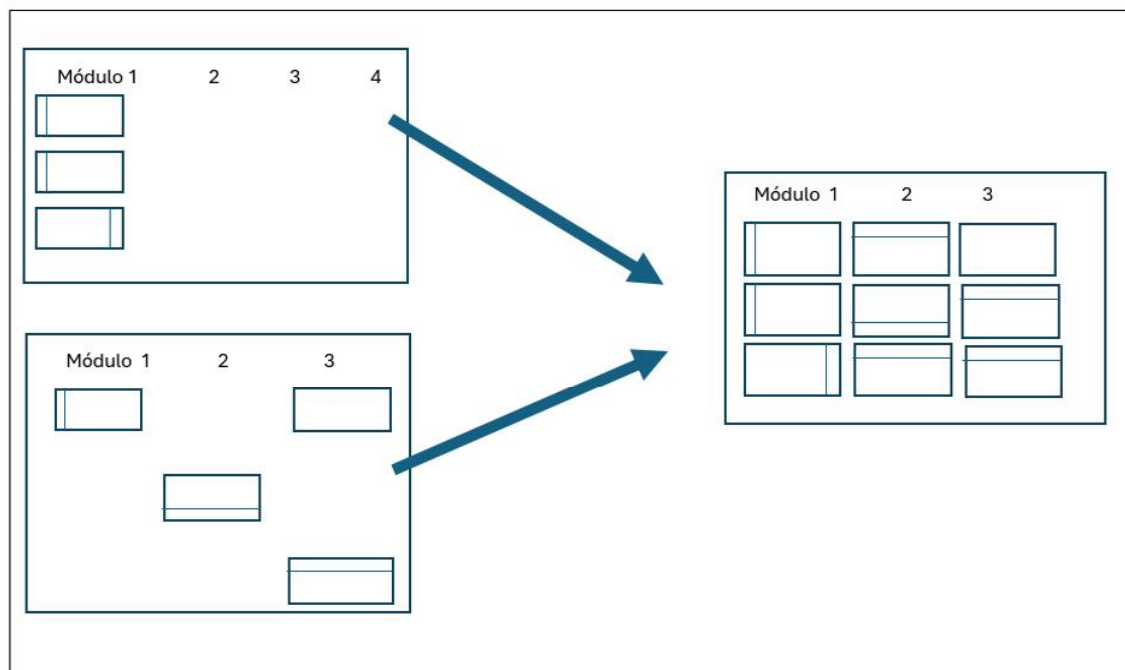


Figura 4.1. Proceso de integración de módulos en un sistema

Pruebas del sistema

Es la culminación de las pruebas de integración, consiste en pruebas de caja negra que validan que la aplicación completa cumpla con los requisitos. Siempre que sea posible, las pruebas se realizan mientras que la aplicación se ejecuta en su entorno requerido, sin embargo, en ocasiones sólo es posible la ejecución de las pruebas en un entorno o configuración que no es equivalente al entorno requerido por el cliente.

Pruebas de utilidad

Las pruebas de utilidad validan la aceptación de la aplicación por los usuarios. Asegura que la aplicación satisface los requerimientos establecidos. Una forma de realizar esta prueba es cuantificar el nivel de satisfacción que informan los usuarios. Los criterios de uso se especifican de antemano. Por ejemplo, es posible que se requiera una muestra aleatoria de 30 usuarios para evaluar la aplicación. La estadística determina el tamaño de la muestra dependiendo del tamaño de la base de datos de clientes pronosticada (Braude, p. 451).

Algunos criterios esenciales para la prueba de utilidad serían:

- Accesibilidad. Facilidad con la que entran, navegan y salen los usuarios de la aplicación.
- Rapidez de respuesta. Rapidez con que el usuario logra las metas especificadas.
- Eficiencia.
- Comprensión. Facilidad con la que el usuario entiende y usa el producto mediante la documentación y ayuda en línea.

Para medir los criterios de utilidad, es común utilizar cuestionarios que deben aplicarse a los usuarios de la aplicación. Al diseñar los cuestionarios, el reto es obtener los datos que permitan a los ingenieros enfocar el desarrollo para remediar las deficiencias más serias, sin exceder el límite de tiempo y paciencia del usuario. Puede ser costoso reunir datos de utilidad porque con frecuencia los usuarios esperan una compensación por su tiempo. Una compañía que desarrolla software para un dispositivo móvil para médicos ofrece una cena y una remuneración económica por ver y comentar imágenes en la aplicación.

Pruebas de regresión

Cuando una aplicación crece, las pruebas del sistema adquieren un significado especial. Al realizar un cambio en una aplicación, los desarrolladores necesitan validar que el cambio realizado no afecta las funcionalidades existentes. Lo más conveniente es verificar que el sistema continúe pasando por el mismo conjunto de pruebas diseñado antes de hacer los cambios.

Una prueba de regresión es el proceso de volver a aplicar el mismo conjunto de pruebas que se había aplicado antes de realizar los cambios. Si el tiempo no permite hacer las pruebas completas, se seleccionan aquellas con mayor probabilidad de fallar debido a efectos colaterales.

Pruebas de aceptación

Cuando se realiza un proyecto de desarrollo de software, tanto el cliente como el desarrollador firman un contrato en el que se estipulan los requisitos solicitados por el cliente y que deben ser cumplidos en el sistema que se desarrolla.

Al terminar el trabajo, el desarrollador obtiene una declaración definitiva del cliente que establece que la aplicación se entregó y que cumplió con los requisitos establecidos.

Las pruebas de aceptación están diseñadas para asegurar al cliente que se construyó la aplicación estipulada. Las pruebas de aceptación son similares a las pruebas del sistema, la diferencia principal es que la empresa del cliente está presente durante las pruebas por lo que es testigo del funcionamiento del sistema y se ejecutan en la plataforma en la que va a operar el sistema.

Prueba de instalación

La prueba de instalación consiste en probar la aplicación en su configuración de hardware final. Es un tipo de prueba que se encarga de verificar que el producto de software se puede instalar y desinstalar exitosamente en su entorno final.

Este tipo de pruebas es importante porque garantiza que la aplicación instalada se adapta a la configuración del equipo de hardware donde se instala, así también garantiza que no causa problemas con las demás aplicaciones y sistemas instalados en el equipo.

Versiones alfa

Se proporcionan versiones alfa a los usuarios internos o a un grupo selecto y confiable de usuarios externos para los primeros usos antes de la liberación del sistema.

El propósito de las versiones alfa es proporcionar retroalimentación a la organización que desarrolló el sistema, así también obtener información de defectos de un grupo más grande que los probadores, sin afectar la reputación del producto no liberado.

Versiones beta

Las versiones beta se dan a parte de la comunidad de clientes con el entendimiento de que informarán acerca de los errores encontrados. Las versiones alfa y beta se usan además para convencer a los clientes potenciales de que en realidad se trata de un producto que respalda las promesas de los proveedores.

Una vez que se conocen algunas de los diferentes tipos de pruebas que existen, se debe realizar un plan de pruebas seleccionando aquellas pruebas que sean las más convenientes de realizar de acuerdo con las características del software en desarrollo y la clase de error que se pretenda encontrar. Para la identificación de errores en el sistema, se sugiere diseñar un plan de pruebas.

Plan de pruebas

Un plan de pruebas es un documento donde se especifica la forma en que se llevarán a cabo las pruebas. Todo proyecto de desarrollo de software debe contar con un plan de pruebas que permita guiar el proceso de aplicación de las pruebas. Con base en el estándar ISO/IEC/IEEE 29119 se describe la siguiente estructura:

1. Introducción

En la introducción se describe el objetivo general que se pretende alcanzar con la aplicación de las pruebas

1.1 Propósito

Se describe el propósito del sistema en desarrollo, así como el propósito de las pruebas que se van a llevar a cabo.

2. Alcance de las pruebas

Aunque idealmente se quisieran aplicar una gran variedad de pruebas, por cuestiones de tiempo y de recursos las pruebas se deben enfocar en un propósito factible.

En el alcance del sistema se especifican cuáles son las pruebas que se van a llevar a cabo, y cuáles son las pruebas que no se van a llevar a cabo, se recomienda documentar las pruebas a realizar en una tabla. Es importante para especificar en qué se centraron los esfuerzos y cuáles son las pruebas que se aplicaron. Por ejemplo, se especifican cuáles son las pruebas que están dentro del alcance, es decir, cuáles son las pruebas que sí se van a aplicar y cuáles pruebas están fuera del alcance o no se van a aplicar.

Alcance de las pruebas	
Dentro del alcance	Fuera del alcance
Pruebas de unidad	Prueba del camino básico
Pruebas de rendimiento	Pruebas de seguridad
Pruebas de usabilidad	
Pruebas de regresión	

2.1 Objetivos

Especificar los objetivos que se pretenden alcanzar con la aplicación de las pruebas. Algunos ejemplos de objetivos en las pruebas son:

- Validar que todas las funcionalidades de la aplicación cumplan con los requisitos del usuario.
- Evaluar la usabilidad de la interfaz del usuario.
- Medir el rendimiento de la aplicación en diferentes condiciones.
- Evaluar que no se introduzcan errores al momento de realizar cambios en el sistema.

2.2 Definiciones, acrónimos y abreviaturas

Estipula toda palabra que pueda resultar ambigua en el documento. El objetivo de esta sección es definir conceptos en el documento que se puedan mal interpretar. Es importante definir cada término ambiguo de manera que se aclare a qué se refiere y que distintas personas que lean el plan de pruebas no tengan distintas percepciones.

Glosario	
Término	Descripción

3. Responsabilidades

Se definen los roles de los participantes en el plan de pruebas, por ejemplo, algunos de los roles más comunes cuando se llevan a cabo las pruebas son: Developer, Tester y Product owner.

Rol	Funciones que lleva a cabo
Product owner (Dueño del producto)	1. Definir las historias de usuario que especifiquen qué funciones se necesitan. 2. Definir criterios de salida que especifiquen las condiciones que deben cumplirse para que una función se considere exitosa. 3. Proporcionar datos de prueba que alimenten al sistema con información real 4. Revisar el producto participando activamente en el proceso de desarrollo para velar por el cumplimiento de sus necesidades.
Dev (Desarrollador)	1. Elaborar la especificación de requerimientos 2. Realizar el modelado del sistema, definen el conjunto de modelos a utilizar, diseñan cómo se solucionará la aplicación, definen arquitectura, interfaz y componentes del sistema 3. Construir prototipos y desarrollar el software 4. Corregir los errores una vez que hayan sido detectados por el QA 5. Configurar el entorno de trabajo utilizando las herramientas más apropiadas para llevar a cabo el proyecto
QA (Control de calidad)	1. Elaborar un documento de gestión de riesgos que se anticipe a los riesgos que puedan surgir y establezca planes de mitigación y monitoreo. 2. Generar el plan de pruebas que especifique cómo se organizan las pruebas, definir el formato de los casos de prueba 3. Elaborar un informe de reportes 4. Realizar correcciones a la documentación para optimizar el proceso de desarrollo

Además de la especificación de las responsabilidades de cada uno de los participantes, se debe especificar la interacción entre los mismos, qué productos de trabajo se van a entregar y a quién se deben entregar.

La función realizada se documenta y se entrega. Para los distintos participantes involucrados, se pueden generar diferentes versiones del documento, por ejemplo, a un desarrollador se entrega una versión detallada, mientras que al Product owner se podría entregar una versión simplificada.

Compromisos de entrega de los participantes en el proceso		
Participante	Función	A quién entrega
Product owner	1. Historias de usuario	Devs
Dev	1. Plan general 2. Especificación de requisitos 3. Modelado 4. Prototipos	Product owner y QA
QA	1. Gestión de riesgos 2. Plan de pruebas 3. Casos de prueba 4. Informe de pruebas	Product owner y Devs

4. Cronograma

Definir que conjunto de tareas se van a llevar a cabo en el proceso de realización de las pruebas, las fechas inicio y terminación.

Actividades del plan de pruebas			
Tarea	Responsable de la tarea	Fecha de inicio	Fecha de terminación
Revisión de la especificación de requisitos			
Diseño del plan de pruebas			
Aprobación del plan de plan de pruebas (reuniones del equipo y correcciones)			
Configuración del ambiente de pruebas			

Pruebas de unidad			
Pruebas de rendimiento			
Pruebas de usabilidad			
Pruebas de regresión			
Informe de defectos			
Entrega del informe QA			

5. Criterios de entrada y salida de las pruebas

Este punto concierne a las precondiciones que se deben cumplir para aplicar un conjunto de pruebas o una prueba en particular, así como las condiciones se deben cumplir para validar que el prototipo que se haya entregado sea correcto. Un ejemplo de criterios de entrada y salida de las pruebas es el siguiente.

Para iniciar las pruebas se debe cumplir:

- Entrega de los prototipos funcionales
- Cumplimiento del 80% de las funcionalidades del entregable para esta versión
- Entrega de la documentación asociada a la versión
- El sistema de gestión de incidencias debe estar activo y configurado

Para validar el incremento entregado se debe cumplir lo siguiente:

- Las fallas, errores o paros en el sistema deber ser menores al 5%
- El cliente valida el producto de trabajo entregado
- Se han corregido los errores asociados en versiones previas
- Los defectos encontrados no pueden ser de naturaleza media o superior

6. Participantes en las pruebas

Se debe de especificar el personal que participa en las pruebas, así como las responsabilidades de las cuales se encarga.

Nombre	rol
Juan Pérez	Documentar el plan de pruebas Diseñar los casos de prueba
Cristian Ramírez	Ejecutar las pruebas de los componentes del módulo 1
María Martínez	Integrar los resultados de las pruebas y elaborar el reporte final de las pruebas

7. Seguimiento de los casos de prueba

Especificar de qué forma y con qué herramienta se da seguimiento a los casos de prueba y definir los estados en los que se podría encontrar una prueba.

Estado	Descripción
No ejecutada	El caso de prueba no se ha ejecutado
No aplicable	No es posible realizar la prueba por estar mal diseñada, por no cumplir las precondiciones de entrada, o por tener una dependencia que no ha sido finalizada
Aprobado	El caso de prueba fue ejecutado y no se encontraron defectos
No aprobado	El caso de prueba fue ejecutado y se encontraron defectos, se reenvía a desarrolladores para su corrección.
En revisión	La prueba está siendo ejecutada, no se ha concluido la prueba.
Cerrada	La prueba ha sido finalizada y documentada, independientemente del resultado

7.1 Seguimiento de defecto

Para cada una de las funciones en las cuales se haya detectado un defecto, se documenta el defecto junto con la siguiente información que se utilizará para anexar en el reporte de métricas de defectos. En esta sección se define la estructura del defecto que haya sido detectado en un caso de prueba.

Id del defecto	Identificador del defecto con referencia al Id del requisito
Defecto	Nombre del defecto observado
Descripción del defecto	Descripción del detalle del defecto y la forma de cómo replicarlo
Severidad	Gravedad del defecto en escala 1 a 5
Prioridad	Impacto del defecto en escala de 1 a 5. Un defecto podría ser severo por no funcionar apropiadamente, pero a la vez podría ser no prioritario por no tratarse de una funcionalidad crítica (la prioridad es más importante que la severidad)
Tester (Probador de software)	Participante del proyecto quien detecta el defecto
Fecha	Fecha en que se detecta o se registra el defecto en la bitácora de incidencias.

Estado	Nuevo: Defecto identificado y registrado En control: Defecto validado por el desarrollador y en seguimiento Corregido: Defecto solucionado
Causa	Especifica la posible causa del error
Clasificación de calidad	Especifica la categoría de funcionalidad afectada por el defecto: funcionalidad, rendimiento, escalabilidad, etc.
Subsistemas afectados	Especifica los subsistemas que están o posiblemente están afectados por el defecto
Versión	Versión del prototipo

8. Pruebas al sistema desarrollado

Especifica la forma en que se implementarán las pruebas, tiempo y horario de las pruebas dependiendo de la prioridad de las funcionalidades. Por ejemplo:

Para las pruebas al sistema se utilizará el conjunto de datos proporcionado por el cliente o en su defecto un conjunto de datos ficticios para probar el sistema. Se puede agregar aquí un enlace al conjunto de datos si lo considera pertinente, o especificar en dónde se encuentran los datos.

Para todos los casos de prueba se realizarán pruebas manuales: las funciones a probar son:

- Registrar cliente
- Login de cliente
- Perfil de de cliente
- Seleccionar automóvil
- Rentar automóvil
- Cancelar automóvil
- Listar automóviles rentados

Id del caso de prueba	

8.1 Priorización de las pruebas

Para realizar las pruebas al sistema se considera en gran medida la prioridad de las funciones, debido a que el tiempo es un recurso limitado y las funciones tienen un nivel de impacto diferente, algunas áreas requieren más atención que otras y algunas funciones requieren la aplicación de cierto tipo de pruebas y otras no.

Cada función tiene distinto nivel de prioridad para el sistema, esta prioridad también difiere según el tipo de prueba. Cada una de las funciones se somete a varios tipos de pruebas, de acuerdo con los objetivos de cada una de las pruebas y funciones. En esta sección se listan las funciones en detalle, cuál es su prioridad y los tipos de pruebas que se aplican. Al realizar las pruebas al sistema, se debe especificar cuál es el usuario y rol que el que se ingresa al sistema. Posteriormente, detallar las acciones que se deben realizar para llegar al último paso y especificar la última acción para verificar el resultado de la prueba.

Especificación de los casos de prueba

Id	RF01
Nombre	Registrar cliente
Descripción	Validar que el usuario pueda crear una cuenta en el sistema
Datos de entrada	Nombre: María Magaña Salinas, RFC: MASM, edad: 32 años, ocupación: Ingeniero de laboratorio químico, correo: mary91@gmail.com , contraseña: contraseñaSeg
Pasos para realizar	1. Acceder a la página de registro del cliente 2. Ingresar el nombre del cliente 3. Ingresar el RFC del cliente 4. Ingresar edad 5. Ingresar ocupación 6. Ingresar correo 7. Ingresar contraseña 8. Hacer clic en el botón registrar
Criterio de aceptación	El cliente crea una cuenta con sus datos y queda registrado en la base de datos
Resultado esperado	Se registra el cliente en la base de datos exitosamente
Precondición	El cliente debe entrar en la página de registro de la aplicación. El sistema debe estar en estado operativo, sin errores críticos.
Observación	Aprobada
Prioridad	Alta

4.4 Documentación de resultados de las pruebas

Una vez realizadas las pruebas, se pueden recabar datos útiles tanto para el proceso de desarrollo actual, como para procesos futuros. El estándar IEEE-829 proporciona una guía para la documentación de pruebas.

4.5 Entrega del sistema y capacitación de usuarios

Una vez que las pruebas del software han sido realizadas y que el sistema ha sido aceptado por todas las partes involucradas, el sistema debe entregarse al cliente. Existen algunas consideraciones que se deben tomar en cuenta al entregar el sistema:

- En qué formato se entrega el sistema, en una unidad de USB, en la nube, o en algún enlace al servidor de la compañía.
- En cuántos equipos se puede instalar el sistema
- Quién será el encargado de la instalación del sistema, el mismo usuario utilizando algún manual de instalación o la compañía que implementa el software
- El equipo de desarrollo es el encargado de la instalación de la aplicación.
- Quién será el encargado de configurar el equipo de hardware para que soporte la instalación del sistema
- Cómo se lleva a cabo la capacitación de los usuarios y si tiene algún costo extra
- Quién se encarga de la capacitación de los usuarios
- Cuánto tiempo dura la capacitación
- En dónde se lleva a cabo la capacitación
- De qué forma se entregan los manuales de usuario. Pueden ser impresos, en algún dispositivo de almacenamiento, en la nube, o pueden estar incrustados como enlaces dentro del sistema

4.6 Entrega de documentación técnica y de usuario del sistema

La documentación del sistema es el conjunto de información que especifica qué hacen los sistemas, cómo lo hacen y para quién lo hacen. Esta documentación consiste en material que explica las características técnicas y las características de operación de un sistema. Es esencial para proporcionar entendimiento de un sistema a quien lo vaya a utilizar, para mantenerlo, para permitir la auditoría del sistema, para enseñar a los usuarios cómo deben interactuar con el sistema y a los operadores cómo hacerlo funcionar. Existen diferentes documentos que por lo general se clasifican en dos manuales: manual técnico y manual de usuario.

Manual técnico

El manual técnico incluye información que explica la lógica del sistema. Contiene listados de los programas, el modelo de la base de datos, el diagrama E-R, definición de variables utilizadas en los programas. En el manual técnico se muestra la arquitectura del software, variables, funciones, procedimientos, clases del programa, metodologías y estrategias utilizadas durante el desarrollo del sistema, y toda información que se considere útil para gestionar modificaciones y mejoras al sistema.

El manual de técnico es importante porque sirve para futuras modificaciones, ya que es difícil dar mantenimiento a un sistema que no cuenta con documentación, porque después de un tiempo de no revisar el código de un programa, se olvida la lógica con que se programa o pudiera ser que la modificación la realice alguien más que no conoce el código fuente. Contar con un manual técnico es una buena opción para retomar un proyecto o para darle mantenimiento a un sistema.

Manual de usuario

El manual de usuario es un documento en el que se organiza toda la información que va dirigida a los usuarios y operadores del sistema. Contiene las funcionalidades y capacidades del sistema, así como la forma de utilizarlo. El manual de usuario es esencial para proporcionar entendimiento de un sistema a quien lo vaya a utilizar.

Los manuales de usuario son guías escritas en formatos impresos (en papel) o en documentos electrónicos (PDF o XPS) que proporcionan instrucciones de cómo hacer o utilizar un sistema. Los buenos manuales de usuario educan al usuario acerca de las funciones del producto mientras que enseñan a utilizarlas de manera efectiva y están dispuestos de tal forma que puedan leerse y consultarse fácilmente.

Uno de los objetivos principales de un manual es exponer los procesos que el usuario puede realizar con el sistema implantado. Detalla cada una de las características que tienen los programas y la forma de capturar la información. Permite a los usuarios conocer el detalle de qué actividades llevarán a cabo para lograr los objetivos del sistema. También reúne la información, normas y documentación necesaria para que el usuario conozca y utilice adecuadamente el sistema desarrollado.

Consideraciones para diseñar un manual de usuario

- Definir quién es el usuario. Se debe desarrollar un perfil de usuario. También se debe tomar en cuenta el lugar donde se utilizará, la forma en que se utilizará, la experiencia que tienen los usuarios con el producto, etc.
- Escribir claro para ayudar a los usuarios a entender sus necesidades. Evitar términos técnicos.

- Si es necesario utilizar algún término técnico, agregar un glosario en el que se defina el concepto de los significados ambiguos.
- Explicar el problema que el usuario quiere resolver y posteriormente presentar la solución a dicho problema.
- Incluir una portada y carátula adecuada. Si el manual tiene derechos de autor, se debe incluir un aviso de derechos de autor en la portada.
- Poner referencias a las documentaciones relacionadas con el manual.
- Si el manual excede de 10 páginas, incluir un índice de contenido.
- Poner las instrucciones, los procedimientos y los materiales de consulta en el cuerpo del manual. Usar imágenes, tablas y gráficas para respaldar el texto. Esto hace un texto más fácil de entender.
- Elegir una fuente de letra que sea legible y una estructura adecuada para distribuir el contenido del manual. Si el manual es impreso, seleccionar el tipo de encuadernación apropiada.
- Diseñar una plantilla de documento para el manual de usuario.

Bibliografía para la materia de Ingeniería de software

1. **CUAED-UNAM (2017);** “Modelo Orientado a Objetos”; Consultado el 20 de octubre de 2024 en: https://repositorio-uapa.cuaieed.unam.mx/repositorio/moodle/pluginfile.php/2731/mod_resource/content/1/UAPA-Modelo-Orientado-Objetos/index.html
2. **Microsoft. (2024);** “Estilo de arquitectura de microservicios”; Consultado el 9 de noviembre de 2024 en: <https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>
3. **Pacheco, M. (2018);** “Normalización de Bases de datos”; Unidades de apoyo para el Aprendizaje”; CUAIEDD/FCA-UNAM. Consultado el 20 de octubre 2024 en: https://programas.cuaed.unam.mx/repositorio/moodle/pluginfile.php/872/mod_resource/content/7/Contenido/index.html
4. **Sánchez Salvador, Sicilia Miguel Ángel, Rodríguez Daniel;** “Ingeniería del software, Un enfoque desde la guía SWEBOK”; Alfaomega; ISBN:978-607-707-420-5, 2012.
5. **SARASA, A.;** “Introducción a las bases de datos NoSQL usando MongoDB”; Editorial: Barcelona, UOC, 2016. 308 p. Disponible en: <https://elibro.net/es/ereader/tecnmcdmadero/58524?page=23>. Consultado en: 23 Oct 2024.
6. **Silberschatz Abraham, Korth Henry, Sudarchan S.;** “Fundamentos de bases de datos”; Editorial: Mc Graw Hill; 5ª edición, ISBN: 0-07-295886-3.
7. **Vizcaino Barceló, A., García Rubio, F. O. ; Piattini Velthuis, M.;** “Desarrollo global de software”; Editorial: Paracuellos de Jarama, Madrid: RA-MA, 2014. 458 p. Disponible en: <https://elibro.net/es/ereader/tecnmcdmadero/106438?page=39>. Consultado en: 27 Oct 2024.
8. **Chacon Scott, Straub, Ben.;** “Pro Git”; Editorial Apress; segunda edición; Version 2.1.303; 28 abril 2021.

9. **Sommerville, Ian**; “Ingeniería de software”; Editorial Pearson, Addison-Wesley; Novena edición; ISBN: 978-607-32-0603-7
10. **Pressman, Roger**; “Ingeniería del software, un enfoque práctico”; Editorial Mc Graw Hill; Séptima edición; ISBN: 978-607-15-0314-5
11. **Weitzenfeld, A.**; “Ingeniería de software orientada a objetos con UML, Java, e internet”; Editorial Thompson; ISBN: 970-686-190-4
12. **Microsoft Ignite**; “Estilo de arquitectura de microservicios”; Microsoft; España; <https://learn.microsoft.com/es-es/azure/architecture/guide/architecture-styles/microservices>
13. **Fowler, Martin**; “Patterns for managing source code branches”, disponible en: <https://martinfowler.com/articles/branching-patterns.html>. Consultado el 27 de octubre de 2024.
14. **Fowler, Martin**; “Microservices”, disponible en: <https://martinfowler.com/articles/microservices.html> . Consultado el 4 de noviembre de 2024.
15. **Marqués, M.**; “Bases de datos”; Editorial: Castelló de la Plana: D - Universitat Jaume I. Servei de Comunicació i Publicacions, 2009. 175 páginas. Disponible en: <https://elibro.net/es/ereader/tecnmcdmadero/51645?page=105>. Consultado en: 04 Nov 2024