

ENRIQUE GÓMEZ JIMÉNEZ

ESTRUCTURA DE DATOS

Guía de estudio

CÓDIGO 0825



UNED

UNIVERSIDAD ESTATAL A DISTANCIA

Institución Benemérita de la Educación y la Cultura



**Producción académica
y asesoría metodológica**

© Enrique Gómez Jiménez
© Universidad Estatal a
Distancia

Diagramación

Corrección de estilo

Ilustraciones

Enrique Gómez Jiménez

Encargado de cátedra

Percy Cañipa Valdez

Revisor

Roberto Morales Hernández

Encargado de carrera

Roberto Morales Hernández

Esta guía de estudio ha sido confeccionada en la UNED, en el año 2023, para ser utilizada en la asignatura Estructura de datos, código 0825, impartida por la cátedra Ingeniería de Software.



Contenido

PRESENTACIÓN	6
OBJETIVO GENERAL.....	7
DESCRIPCIÓN DE LA GUÍA DE ESTUDIO	8
Tema I. Estructuras de datos en C++	2
<i>Sumario</i>	2
Objetivos específicos.....	3
Introducción	4
Guía de lectura.....	5
Capítulo 1. Introducción a C++	5
Resumen del capítulo	7
Ejercicios de autoevaluación del capítulo 1.....	8
Capítulo 2. Programación orientada a objetos (POO) en C++.	11
Resumen del capítulo	13
Ejercicios de autoevaluación del capítulo 2.....	14
Capítulo 3. Estructuras de datos y recursividad en C++.....	17
Resumen del capítulo	19
Ejercicios de autoevaluación del capítulo 3.....	20
Capítulo 4. Ordenamientos y búsquedas en C++.	23
Resumen del capítulo	26
Ejercicios de autoevaluación del capítulo 4.....	27
Capítulo 5. Pilas y colas en C++.	30
Resumen del capítulo	31
Ejercicios de autoevaluación del capítulo 5.....	32
Capítulo 6. Listas enlazadas en C++.	35
Resumen del capítulo	37

Ejercicios de autoevaluación del capítulo 6.....	38
Capítulo 7. Árboles en C++.	40
Resumen del capítulo	45
Ejercicios de autoevaluación del capítulo 7.....	46
Capítulo 8. Grafos en C++......	49
Resumen del capítulo	52
Ejercicios de autoevaluación del capítulo 8.....	53
Capítulo 9. STL: Standard Template Library	56
Resumen del capítulo	58
Ejercicios de autoevaluación del capítulo 9.....	59
Capítulo 10. Manejo de archivos en C++	62
Resumen del capítulo	64
Ejercicios de autoevaluación del capítulo 10	65
Tema II. Estructura de datos en Java.....	68
Sumario	68
Objetivos específicos.....	69
Introducción	69
Guía de lectura.....	70
Capítulo 11. Fundamentos de estructura de datos en Java/Parte I	70
Resumen del capítulo	73
Ejercicios de autoevaluación del capítulo 11	74
Capítulo 12. Fundamentos de estructura de datos / Parte II.....	77
Resumen del capítulo	79
Ejercicios de autoevaluación del capítulo 5.....	80
Guía de lectura.....	81
Capítulo 13. Fundamentos de estructura de datos en Python/Parte I	81
Resumen del capítulo	84
Ejercicios de autoevaluación del capítulo 13	85
Capítulo 14. Fundamentos de estructura de datos en Python/Parte II.....	85
Resumen del capítulo	87
Ejercicios de autoevaluación del capítulo 14	88



PRESENTACIÓN

Las estructuras de datos constituyen uno de los pilares de la lógica de programación a través de algoritmos que requieren mucho razonamiento y modelado matemático. A través de los tiempos se han ido creando bibliotecas y herramientas que coadyuvan a la implementación de estructuras que hace muchos años requería de mucha codificación y mecanismos lógicos de manejo. En C++ por ejemplo, la biblioteca STL provee de muchas funciones que permiten el manejo eficiente y sencillo de estructuras tales como pilas, colas, árboles y grafos. Java también dispone de bibliotecas similares y ni decir de Python que ofrece importantes prestaciones para el manejo de estructuras de datos simples y complejas.

El libro sobre el cual se basa esta Guía aborda los principales temas sobre los cuales se fundamenta todo el engranaje de las estructuras de datos: arreglos, pilas, colas, árboles, listas, árboles y grafos. Los algoritmos que se desarrollan en cada capítulo demuestran el uso de estas estructuras, tanto en el lenguaje C++ como en Java y Python. Con ello se demuestra las diferencias en cuanto el código de implementación, siendo la misma lógica en cuanto el funcionamiento de estas. En Java y Python se demuestra el uso de diferentes librerías que facilitan la implementación del código de funcionamiento de estructuras de datos, sin el uso de punteros como se desarrolla en los primeros capítulos del libro. También se desarrolla el tema del uso de la librería STL de C++ que también provee muchas librerías que facilitan la implementación de estructuras de datos, evitando muchas veces el uso de punteros y manejo de memoria.



OBJETIVO GENERAL

Capacitar al estudiante en la lógica de implementación de diversas estructuras de datos tales como pilas, colas, árboles y grafos, mediante el uso de diversos lenguajes de programación.



DESCRIPCIÓN DE LA GUÍA DE ESTUDIO

La guía de estudio para la asignatura estructuras de datos (0825) se propone orientar al estudiante en el desarrollo de cada temática según el libro de texto. Se divide en capítulos relacionados y lógicamente vinculados. Se espera que coadyuven a la asimilación de los conocimientos necesarios para desarrollar las competencias requeridas para comprensión de los temas tratados en el libro de texto. A continuación, se resume el contenido de los temas de la asignatura.

Tema	Descripción
I. Estructuras de datos en C++.	En este tema se desarrolla la conceptualización de estructuras de datos tales como pilas, colas, listas enlazadas, árboles y grafos. La implementación de algoritmos de mantenimiento de datos, recorridos, búsquedas y otros temas relacionados se desarrollan mediante el uso del lenguaje C++.
II. Estructura de datos en Java.	Estructuras de datos como pilas, colas, listas enlazadas, árboles y grafos se conceptualizan someramente en este tema, dado que el Tema I los trata inicialmente. La diferenciación en este tema II radica en la implementación mediante lenguaje Java de los problemas tratados y resueltos en el primer tema.

Tema	Descripción
III. Estructuras de datos en Python.	La temática de estructuras de datos como pilas, colas, listas enlazadas, árboles y grafos se resumen conceptualmente en este tema y se enfatiza en la implementación de código Python en la solución de problemas ya tratados en los dos primeros temas. La idea principal es como diferenciar las implementaciones de código en tres lenguajes bien conocidos en estructuras de datos: C++, Java y Python.

Cada tema de la guía de estudio está estructurado de la siguiente manera:

- Sumario: enlista los conceptos que se abordarán en el tema.
- Objetivos específicos: son los que se espera que usted logre al finalizar el estudio del tema.
- Introducción: ofrece un panorama general del estudio del tema.
- Guía de lectura: en un cuadro que indica las páginas del libro externo que deben estudiarse para el tema respectivo.
- Comentarios del tema: presenta las ideas principales desarrolladas en cada capítulo, complementadas con un esquema resumen del capítulo y recuadros con sitios de interés para ampliar el tema. Al final de cada capítulo, se plantean los ejercicios de autoevaluación.



Tema I. Estructuras de datos en C++

Sumario

- Introducción a C++
- Programación orientada a objetos
- Estructuras de datos y recursividad en C++
- Ordenamientos y búsquedas en C++
- Pilas y colas en C++
- Listas enlazadas en C++.
- Árboles en C++
- Grafos en C++
- STL: Standard Template Library
- Manejo de archivos en C++.



Objetivos específicos

Al finalizar el estudio de este tema, usted estará en capacidad de:

- Emplear C++ y programación orientada a objetos en la solución de problemas computacionales que involucren el uso de objetos, relaciones entre clases, herencia, polimorfismo y otros elementos.
- Emplear la recursividad, el uso de punteros y de arreglos en C++ para la solución de problemas que utilicen estructuras de datos primitivas o complejas.
- Utilizar el lenguaje C++ para implementar métodos de ordenamiento y búsqueda de elementos u objetos en arreglos de datos.
- Resolver problemas de manipulación de objetos de datos en memoria mediante el uso de pilas, colas, listas enlazadas, árboles y grafos, utilizando C++.

Introducción

Este tema conceptualiza temas relativos a las estructuras de datos tales como arreglos, pilas, colas, listas enlazadas, árboles y grafos y operaciones sobre ellos como ordenamientos, búsquedas, recorridos, mantenimientos de datos, entre otros. Para ello se utiliza el lenguaje de programación C++, su sintaxis, librerías y otros mecanismos de abstracción para la implementación de código que resuelve algunos problemas relacionados con estas estructuras.

El estudiante podrá experimentar la lógica de implementación de algunas de estas estructuras mediante recursividad o el uso de punteros a datos. Asimismo, la importancia de las estructuras de datos y el poder de solución que provee C++ para manipular la información que se maneja a nivel de memoria y de persistencia en disco. También podrá comprender la importancia que significan las estructuras de datos para resolver problemas comunes, simples o complejos, que se suscitan en la vida normal del desarrollo de software.

Guía de lectura

Para el estudio de este tema, lea detalladamente las páginas del libro que se indican a continuación:

Capítulo	Páginas
Capítulo 1. Introducción a C++	3-18
Capítulo 2. Programación orientada a objetos en C++	23-41
Capítulo 3. Estructuras de datos y recursividad en C++	47-67
Capítulo 4. Ordenamientos y búsquedas en C++	73-98
Capítulo 5: Pilas y colas en C++	105-114
Capítulo 6: Listas enlazadas en C++	124-140
Capítulo 7: Árboles en C++	149-168
Capítulo 8: Grafos en C++	176-200
Capítulo 9: STL: Standard Template Library	210-231
Capítulo 10: Manejo de archivos en C++	237-246

Comentarios del tema

Capítulo 1. Introducción a C++

1.1. Conceptos básicos de C++.

El *tipado de datos* significa que cada variable, argumento de función o valor devuelto debe ser de algún tipo soportado por un lenguaje de programación. C++ no es una excepción y adopta esta convención, considerando un tipado como el mostrado en la figura número 1.1.

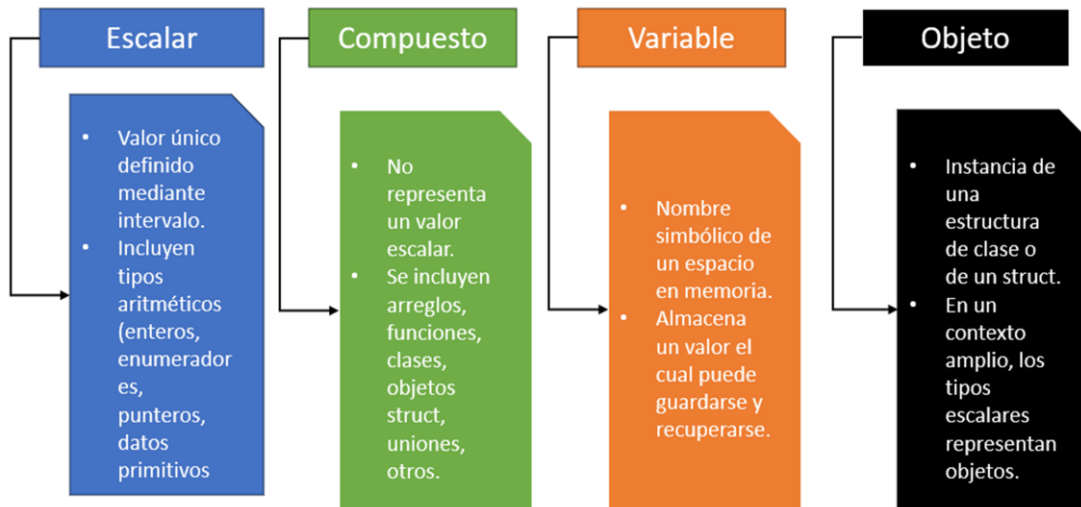


Figura 1.1. Caracterización de tipos en C++

Fuente: elaboración propia

Otro concepto importante además del tipado en C++ lo constituye el ámbito que es el alcance que puede tener un elemento de programa (clase, función, variable) en todo el campo de ejecución de éste. La figura 1.2. muestra la posibilidad de los alcances o visibilidad de los elementos.



Figura 1.2. Alcance o visibilidad de elementos de un programa en C++

Fuente: elaboración propia

Otros elementos importantes en cualquier lenguaje de programación son los elementos que componen la estructura semántica del lenguaje de programación, entre ellos están los identificadores, las palabras reservadas, los delimitadores y los comentarios. Estos elementos son importantes para estructurar programas válidos en cualquier lenguaje.

Los tipos de datos, el manejo de E/S, las operaciones y operadores, las estructuras condicionales y repetitivas, las funciones, métodos y procedimientos y los tipos de parámetros (por valor y por referencia) son, finalmente, lo que termina de complementar la estructura conformativa de un lenguaje de programación como C++.



En el libro de texto se detalla cada uno de los elementos básicos que componen la estructura de un lenguaje de programación como C++ y que se describen en los puntos 1.1. y 1.2. . Estos elementos se desarrollan claramente en las páginas 7 a la 16. Estos conceptos son básicos y de conocimiento previo al estudio de las estructuras de datos, por lo que deben ser conocidos por el estudiante de forma anticipada.

Resumen del capítulo

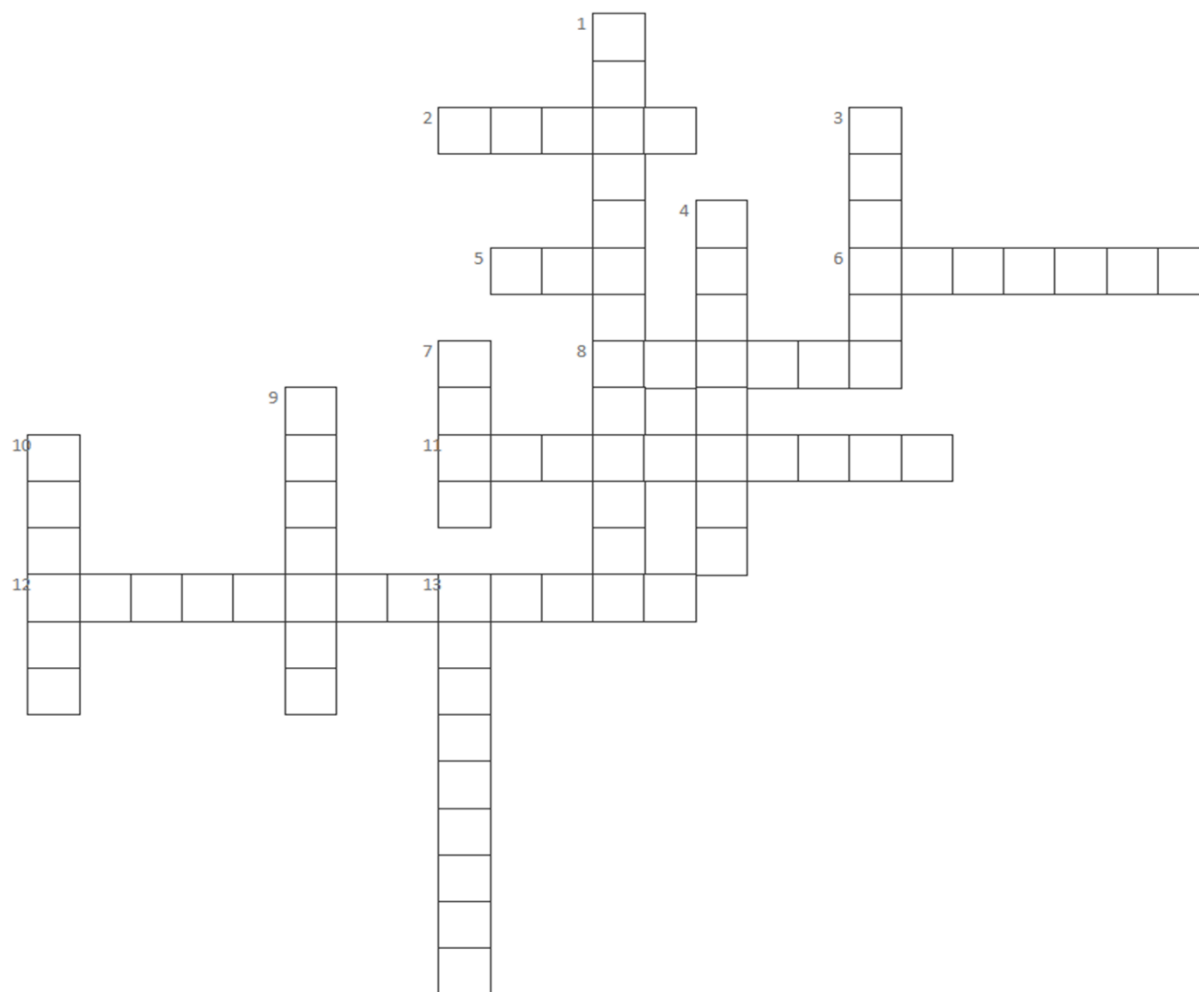
Este capítulo describe los aspectos básicos del lenguaje de programación C++ como antesala para el desarrollo de temas avanzados como lo son las estructuras de datos. En este capítulo se tratan temas tales como tipos de datos, aspectos de E/S de datos, operaciones, operadores, estructuras condicionales y repetitivas, funciones, métodos y procedimientos, parámetros por valor y por referencia y una introducción al uso de vectores.

Ejercicios de autoevaluación del capítulo 1

1. Enumere los tipos de datos que se manejan en C++.
2. Describa el mecanismo con que se maneja las entradas y salidas (E/S) un programa en C++.
3. Describa la sintaxis de las estructuras condicionales y repetitivas que se utilizan en el lenguaje de programación C++.
4. Discrimine la diferencia de utilizar parámetros por valor y por referencia en un programa C++ y en qué situaciones se pueden implementar.

Comprobación teórica

Complete el siguiente crucigrama con las respuestas correctas



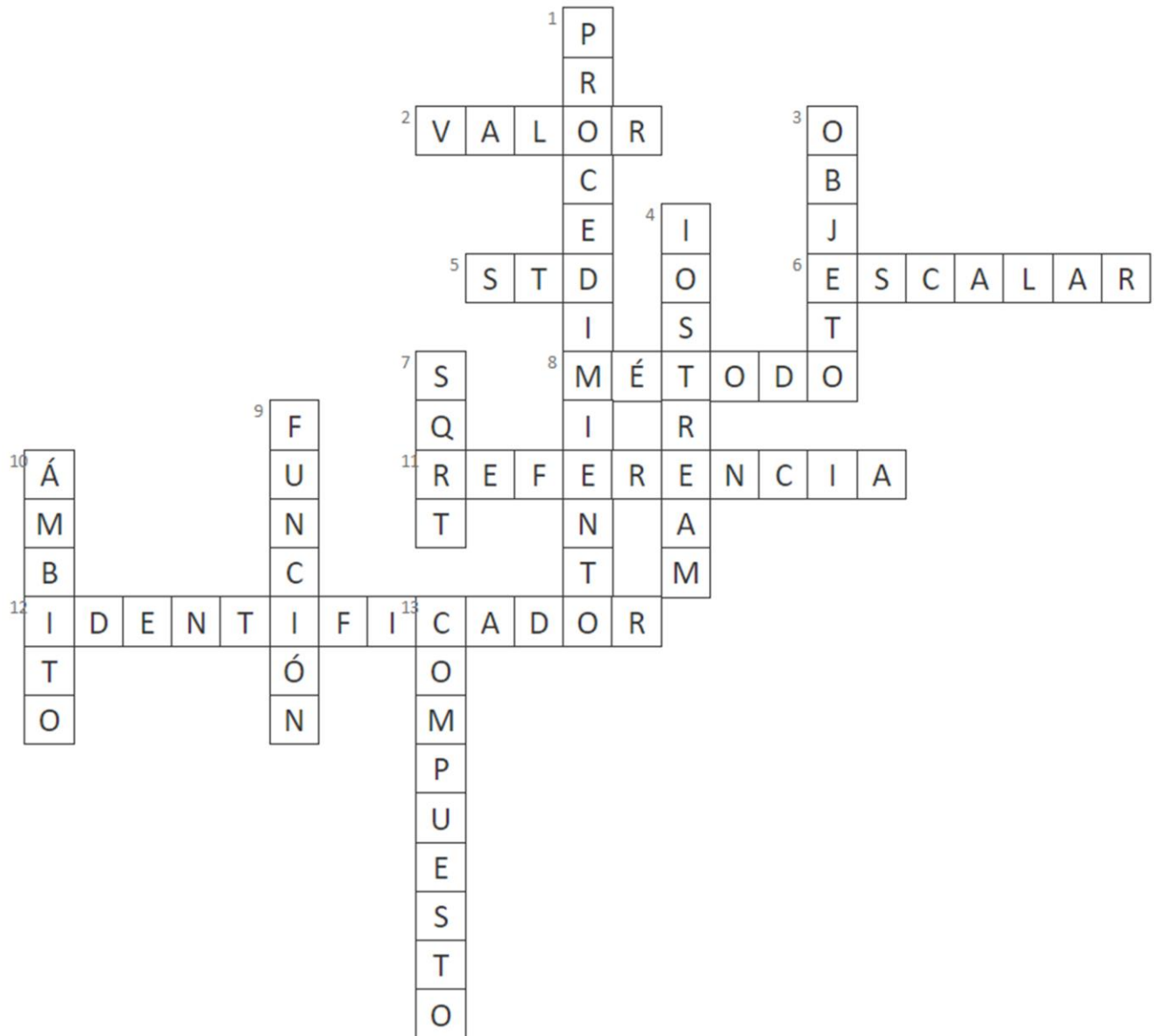
Verticales

- 1.**conjunto de instrucciones de código para realizar una tarea
- 3.**es la instanciación de una clase
- 4.**permite el procesamiento de I/O en un programa de consola C++
- 7.**permite obtener la raíz cuadrada de un número
- 9.**procedimientos encapsulados en un bloque de código
- 10.**alcance o visibilidad de un objeto dentro de un programa
- 13.**representan una conjunción de valores escalares

Horizontales

- 2.**tipo de parámetro que no cambia el valor original de una variable
- 5.**gestiona la memoria dinámica del computador en un programa C++
- 6.**representa valores numéricos únicos definidos por un intervalo
- 8.**procedimiento encapsulado dentro de una clase
- 11.**tipo de parámetro que cambia el valor original de una variable
- 12.**representan a variables, tipos, constantes, procedimientos y otros

Respuesta al crucigrama



Capítulo 2. Programación orientada a objetos (POO) en C++.

2.1. Introducción a la programación orientada a objetos en C++.

Como un punto importante a considerar en el resto del contenido del libro es importante que el estudiante conozca los conceptos básicos de la programación orientada a objetos (POO). Esto por cuanto, las clases y las estructuras (struct) serán de uso constante en todos los conceptos y los programas que se desarrollan en todos y cada uno de los capítulos de libro de texto.

2.1. Clases abstractas, asociación por composición y agregación.

Los conceptos por considerar como conocimiento previo y que se incluye en el libro de texto, en este capítulo se muestran en la tabla 2.1.

Tabla 2.1. Conceptos básicos de la programación orientada a objetos (POO)

Concepto	Definición
Clase	Es una plantilla compuesta principalmente por atributos, propiedades, constructores/destructores y métodos.
Herencia simple	Capacidad de una clase de heredar de una sola clase. En este caso, puede ser que una clase <i>Empleado</i> herede solo de otra denominada <i>Departamento</i> .
Herencia múltiple	Capacidad de una clase de heredar de dos o más clases con el fin de implementar las funcionalidades y acceso a datos a mayor envergadura. Por ejemplo, la clase <i>Matricula</i> que herede de las clases <i>Estudiante</i> y <i>Cursos</i> .
Clase abstracta	Es una clase que no se puede instanciar pero si heredar. El propósito de una clase abstracta es definir

Concepto	Definición
	funcionalidades (el Qué) las cuales las clases que hereden de ella tengan que implementar el código respectivo (el Cómo). Por ejemplo, una clase abstracta puede definir una funcionalidad como <i>Imprimir</i> que no tiene código. La clase que herede de esta deberá implementar el código <i>Imprimir</i> de acuerdo con sus necesidades.
Composición	Es un tipo de relación en la cual existe una dependencia fuerte entre una clase y otra. En otras palabras la existencia de una depende de otra. Consideremos la relación entre botones de comando, etiquetas, cuadros de texto, entre otros, dentro de un formulario. Todos los objetos dentro del formulario son dependientes de éste y si este es destruido también lo son los contenidos. También un objeto de la clase <i>motor</i> dentro de una clase <i>automóvil</i> supondrá la existencia del primero mientras exista la segunda.
Agregación	Es un tipo de relación en la cual existe una dependencia débil de una clase de otra. Supone que no se limita la existencia de un objeto dentro de otro. Por ejemplo, el pase por parámetro de un objeto a otro objeto supone que el que recibe puede usar este parámetro para sus propios fines, sin que su destrucción signifique la destrucción del objeto que fue pasado por parámetro. Por ejemplo, si existen dos objetos: <i>automóvil</i> y <i>motor</i> y el primero recibe por parámetro el segundo, de destruirse <i>automóvil</i> no supone la destrucción de <i>motor</i> .



Estudiar las páginas 23 a 44 donde se detallan los aspectos teóricos de conceptos de programación orientada a objetos (POO), tales como clases, objeto, herencia simple, herencia múltiple, clases abstractas, clases compuestas y clases agregadas. En este espacio se muestran programas demostrativos de estos conceptos, los cuales deben estudiarse y realizar prácticas similares que fortalezcan el dominio sobre estos temas.

Resumen del capítulo

Este capítulo describe los conceptos de la programación orientada a objetos y demuestra su utilización mediante programas de implementación. Cabe destacar que la composición indica una agregación fuerte, significando que una clase consta de objetos de otra clase para funcionar. Por ejemplo, una clase *Edificio* está compuesto por elementos de otra clase *Piso*. Destruída la clase *Edificio* lógicamente se destruirán todos los elementos de la clase *Piso*. Es decir, un edificio consta de pisos, significando que un piso existe si hay un edificio. La agregación, por su parte, indica que una clase es parte de otra. Por ejemplo, una clase *curso* posee elementos de una clase *modulo*. Puede que un curso desaparezca pero no significa que el módulo si lo haga, puesto que puede ser parte de otro curso.

Ejercicios de autoevaluación del capítulo 2

1. Defina los conceptos de clase, herencia, composición y agregación.
2. Describa la diferencia entre una clase compuesta y una clase agregada.
3. Defina el concepto de clase abstracta, su funcionalidad e importancia.
4. Implemente ejercicios adicionales a los propuestos y resueltos para fortalecer el dominio del tema.
5. Complete en la columna de la derecha el concepto correspondiente al término de la izquierda:

Termino	Concepto
Instancia	
Interface	
Clase abstracta	
Polimorfismo	
Herencia	

Termino	Concepto
Función virtual	
Herencia simple	
Herencia múltiple	
Objeto	
Alcance o ámbito	
Composición	
Agregación	
Argumentos	
Mutadores	
Encapsulamiento	

6. Busque los siguientes conceptos en el cuadro de sopa de letras.

Abstracción	Agregación
Atributos	Clase
Composición	Encapsulación
Herencia	Mutadores
Métodos	Polimorfismo
Propiedad	

Sopa de Letras POO

Conceptos de Programación Orientada a Objetos

T	E	A	G	R	E	G	A	C	I	Ó	N	A	P
B	N	X	A	E	M	P	D	J	G	R	T	T	U
H	C	H	B	M	U	C	P	Y	P	C	X	R	J
U	A	E	S	É	T	O	O	H	R	N	L	I	H
C	P	R	T	T	A	M	L	K	Y	T	N	B	Q
K	S	E	R	O	D	P	I	K	Q	Y	S	U	C
F	U	N	A	D	O	O	M	E	J	H	Y	T	Y
F	L	C	C	O	R	S	O	M	I	M	V	O	U
N	A	I	C	S	E	I	R	T	W	A	Z	S	B
R	C	A	I	J	S	C	F	Y	A	F	E	I	W
A	I	L	Ó	H	N	I	I	T	Q	Y	I	B	M
J	Ó	G	N	Q	R	Ó	S	A	S	J	G	Q	G
Y	N	D	B	X	V	N	M	F	C	L	A	S	E
D	H	X	Y	F	P	R	O	P	I	E	D	A	D

Capítulo 3. Estructuras de datos y recursividad en C++.

3.1. Iteración y recursividad en C++.

Una iteración es aquel conjunto de instrucciones que se agrupan dentro de un ciclo. Por ejemplo, la impresión de 10 números enteros procesados mediante un ciclo *for* es una iteración. En este caso existe una condición de terminación determinada por un número finito (hasta llegar a 10). En otros ejemplos podría ser una condición aportada por el usuario (por ejemplo que no desee continuar incluyendo datos). La recursión, por su parte, es una característica donde una función se llama a sí misma y cuenta con un criterio de finalización determinado.

3.2. Manejo de cadenas en C++

C++, al igual que otros lenguajes de programación, cuenta con funciones que permiten el manejo de cadenas. Estas funciones nos permiten concatenar dos cadenas, obtener una subcadena de una cadena, copiar una cadena dentro de otra cadena, comparar cadenas, evaluar cadenas, entre otras funcionalidades.

3.3. Punteros en C++

Los punteros es un concepto muy importante para el manejo de memoria dinámica, mediante programación. Es una variable que almacena la dirección o posición de memoria de un valor además de su contenido. Existen punteros inteligentes (*smart pointers*) los cuales son dinámicos y se autodestruyen cuando son dejados de utilizar, punteros iteradores y expresiones lambda.

3.4. Punteros a clases en C++

Mediante este tipo de puntero es posible gestionar clases en memoria. Podemos crear un objeto como instancia de una clase y sobre éste una

variable de tipo puntero que permita su direccionamiento en memoria.

3.5. Punteros y vectores en C++

Al igual que podemos establecer un puntero a una clase lo podemos hacer sobre un vector de datos. La implementación consiste en tener un vector de elementos y un puntero que cree la referencia a ese vector.

3.6. Arreglos unidimensionales dinámicos mediante punteros en C++

Los arreglos unidimensionales son estructuras estáticas que dependen de un número de elementos máximo permitidos para almacenamiento. Sin embargo, mediante punteros es posible crear arreglos dinámicos en C++ y utilizar *new* y *delete* para reservar espacios dinámicos de memoria y también eliminarlos de ésta.

3.7. Arreglos como parámetros por referencia mediante punteros en C++

El paso por referencia o por valor de una variable, a través de un parámetro en una función, es algo común en la programación de computadores. También lo es el paso de arreglos como parámetros por referencia a una función. Pero no es tan común crear esa referencia mediante punteros. En este apartado se demuestra su uso, el cual es muy importante.

3.8. Matrices dinámicas mediante punteros en C++

Al igual que se crean arreglos unidimensionales dinámicos mediante punteros en C++, también lo podemos hacer con matrices o arreglos bidimensionales. El uso de punteros optimiza la velocidad de acceso a estas estructuras de datos.

3.9. Struct mediante punteros en C++

Además del uso de clases es posible el uso de estructuras de datos de tipo *struct*. Mediante esta estructura es posible solo declarar los tipos de datos que se utilizarán, incluyendo punteros, para el manejo dinámico en memoria de éstos.



Estudiar a fondo los conceptos de punteros que se desarrollan entre las páginas 51 a 66, dado que son vitales para la implementación de estructuras de datos complejas como listas enlazadas, árboles y grafos. En este espacio se muestran programas demostrativos de estos conceptos, los cuales deben estudiarse y realizar prácticas similares que fortalezcan el dominio sobre estos temas.

Resumen del capítulo

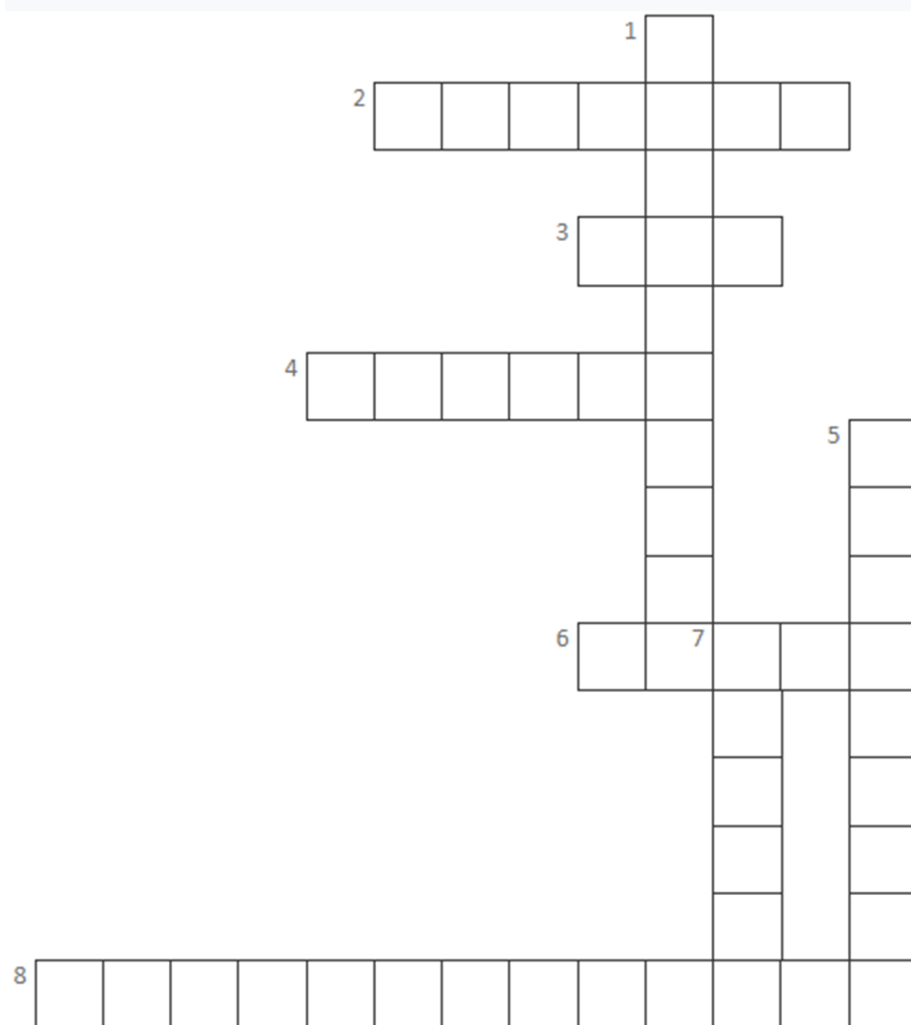
Este capítulo describe los conceptos de iteración y recursividad, como aspectos importantes de la programación. El primer concepto como un proceso tradicional de imponer la ejecución de múltiples instrucciones de código mediante ciclos controlados por un criterio predefinido o decidido por el usuario. El segundo concepto como una característica de las funciones de llamarse a si misma para conseguir ese mismo efecto de ejecución de varias instrucciones delimitados por un criterio de finalización.

También se trata el concepto de punteros, tan importante en el manejo dinámico de memoria en estructuras de datos. Se desarrolla el tema de punteros a clases, arreglos y su uso como parámetros a una función. Tal concepto es muy importante de analizar, practicar y dominar para considerar en el resto de los temas a desarrollar en el libro de texto.

Ejercicios de autoevaluación del capítulo 3

1. Discrimine la diferencia entre los conceptos de iteración y recursividad.
2. Enumere algunas funcionalidades que se pueden implementar en el manejo de cadenas en C++.
3. Describa el mecanismo de manejo de los punteros en C++.
4. Practique el uso de punteros a clases, vectores y estructuras para dominar el tema.

5. Complete el siguiente crucigrama con las respuestas correctas



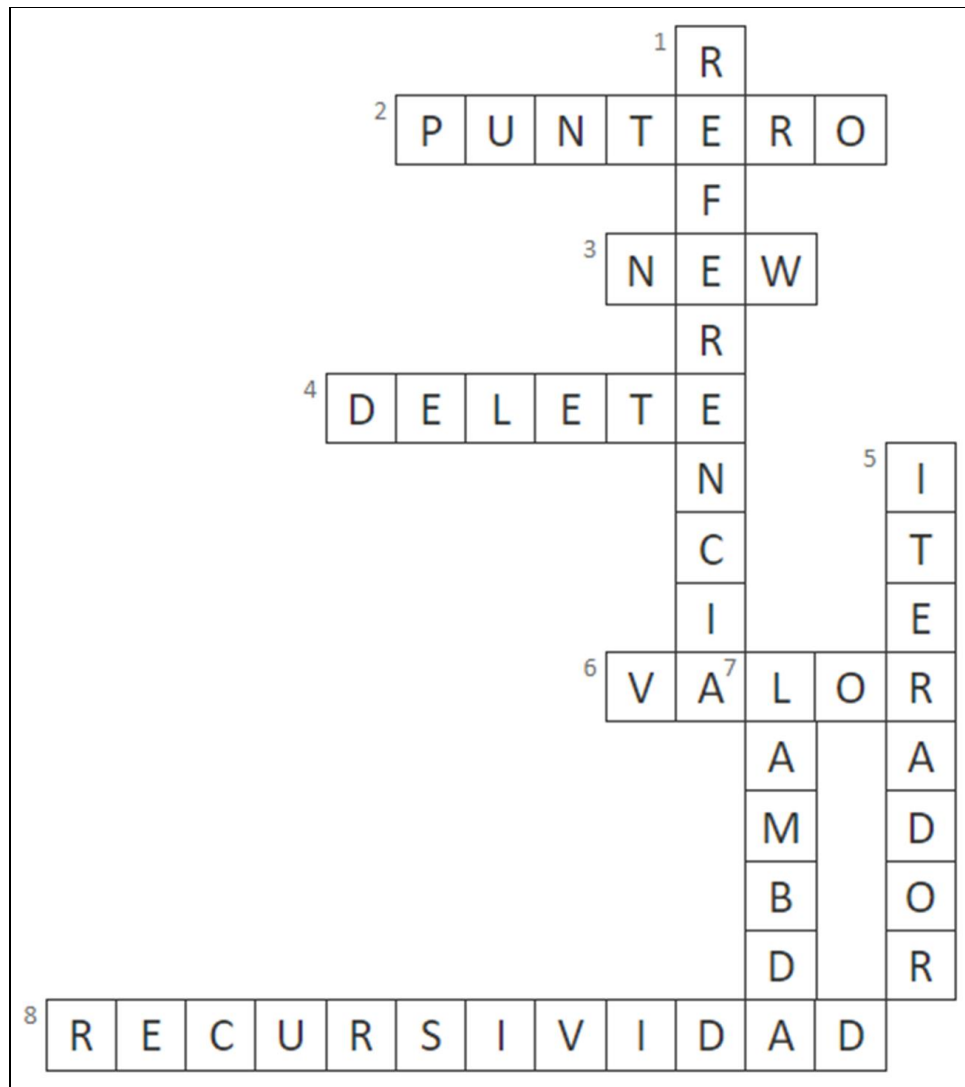
Verticales

- 1.** Tipo de parámetros que su alcance es a nivel global y que su valor se modifica en cualquier lugar del programa donde se declare.
- 5.** Objeto que navega entre el contenido de algún tipo de contenedor como una lista.
- 7.** Permiten definir objetos en tiempos de ejecución, encapsulando líneas de código.

Horizontales

- 2.** Variable que almacena la posición o dirección de memoria donde se guarda un objeto y su contenido
- 3.** Reserva el número requeridos de bytes por la declaración del dato a utilizar
- 4.** Libera el bloque de bytes reservado con anterioridad por algún puntero
- 6.** Tipo de parámetros que su alcance es a nivel local y no se modifican externamente a pesar que una función lo haga internamente.
- 8.** Característica de la programación de computadoras que permite que un subprograma se llame a sí mismo

Respuesta al crucigrama



Capítulo 4. Ordenamientos y búsquedas en C++.

4.1. Métodos de ordenamientos en C++

El ordenamiento de estructuras de datos, principalmente arreglos unidimensionales y multidimensionales, es un proceso muy importante para realizar búsquedas no secuenciales y visualizaciones ordenadas de datos. En computación existen muchos métodos de ordenamientos, los cuales se dividen en dos categorías: directos o básicos e indirectos o avanzados. La imagen 4.1. muestra estas categorías y los algoritmos que pertenecen a cada una.

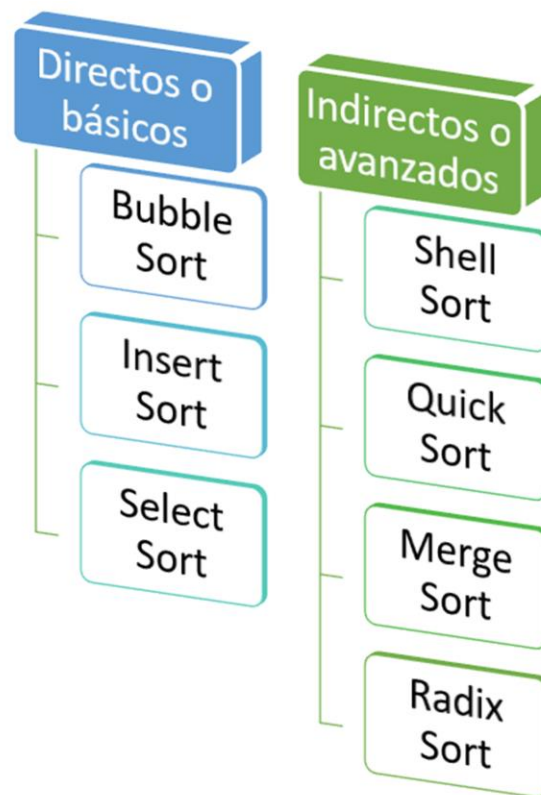


Figura 4.1. Categorías y algoritmos de ordenamiento más conocidos.

Fuente: elaboración propia

4.1.1. Método Bubble Sort.

Es un método muy fácil de implementar y comprender su funcionamiento.

Consiste en revisar cada elemento de una lista a ordenar con el siguiente elemento, intercambiándolos si éste es mayor que el siguiente. Este proceso es necesario repetirse varias veces hasta que no se necesiten más intercambios.

4.1.2. Método Insert Sort.

Este método actúa realizando un recorrido por la lista a ordenar, considerando el elemento actual y luego insertándolo entre los que ya ha realizado el recorrido.

4.1.3. Método Select Sort.

El método Select Sort consiste en buscar el dato menor entre los elementos no ordenados de una lista y colocarlo al principio de esta. Luego debe repetirse este proceso con los restantes datos de la lista, no considerando los que ya se encuentran ordenados.

4.1.4. Método Shell Sort.

El método Shell Sort es una mejora al método Insert Sort y consiste en comparar elementos separados por un espacio de varias posiciones, permitiendo que cada elemento realice pasos más grandes hacia su posición esperada, finalizando con un ordenamiento por inserción simple. Indicamos que es una mejora al método Insert Sort dado que éste es eficiente solo en listas que se encuentren casi ordenadas, mientras Shell Sort es más optimizado trabajando sobre listas bastante desordenadas.

4.1.5. Método Quick Sort.

Este método es un tipo de algoritmo divide y vencerás. Funciona seleccionando un elemento que actúa como pivote y divide el arreglo alrededor de este. Para este algoritmo hay variaciones en la determinación del pivote: elegir siempre el primero pivote, el último elemento, un elemento

aleatorio o el elemento de la mitad de la lista. Una vez elegido el elemento pivote se procede a la partición de la lista, los mayores a la derecha del pivote y los menores a la izquierda (asumiendo que el pivote está a la mitad). Seguidamente, se siguen subdividiendo las listas en sublistas hasta ordenarlas todas y realizar finalmente la unión de todas las sublistas ordenadas.

4.1.6. Método Merge Sort.

Este método consiste en dividir la lista desordenada en dos sublistas de aproximadamente la mitad de tamaño y procede a ordenarlas de forma recursiva, aplicando el ordenamiento por mezcla y finalmente une las dos sublistas en una sola lista ordenada.

4.1.7. Método Radix Sort.

Este método no se incluye en el libro de texto y solo se menciona por cuestiones informativas. Por lo tanto, queda a discreción del estudiante si desea investigar sobre este método.



Estudiar a fondo los algoritmos de ordenamiento que se desarrollan entre las páginas 73 a 91 y los algoritmos de implementación de cada uno. Se deben crear casos nuevos de prueba utilizando el código aportado en el libro y analizar el mecanismo de ordenamiento de cada uno para así conocer el funcionamiento de cada uno.

4.2. Búsquedas en arreglos en C++.

Los dos tipos de búsqueda que se explican en el libro de texto y que son las más usuales en estructuras de datos, principalmente arreglos, son la secuencial y la binaria. La búsqueda secuencial se realiza sobre una lista o vector desordenado dado que no se sabe a ciencia cierta donde se encuentra el dato buscado o si éste existe. Por el contrario, la búsqueda binaria tiene

como requisito que la lista o arreglo se encuentre ordenado previamente. Con ello se garantiza que puede buscar el valor en dicha lista o arreglo ordenado realizando particiones a la mitad de la misma y buscando dicho valor en la sublista izquierda si es menor o en la derecha si éste es mayor.



Una vez comprendidos los mecanismos de ordenamiento el estudiante debe comprender el mecanismo de búsqueda de datos en una lista ordenada, principalmente la binaria. Para la búsqueda secuencial es un mecanismo que no requiere mayor análisis puesto que consiste en buscar de principio a fin un elemento dentro de la lista o arreglo desordenado. Para comprender adecuadamente como se manejan estas búsquedas estudiar las páginas 91 a 98.

Resumen del capítulo

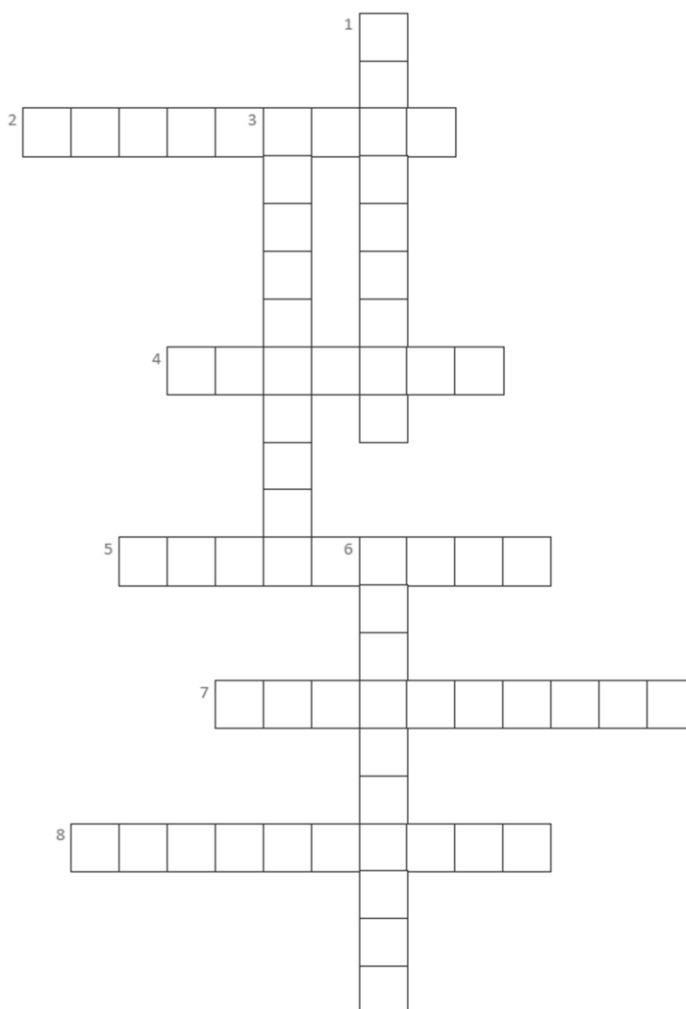
Este capítulo describe, explica e implementa diversos algoritmos de ordenamiento de listas o arreglos. Cada uno aporta un mecanismo de ordenamiento que es importante conocer y dominar su implementación para posteriormente ser utilizado en estructuras de datos.

El siguiente tema desarrollado es la búsqueda de datos en una lista o arreglo. Para ello se desarrollan los algoritmos de búsqueda secuencial o lineal, que puede usarse principalmente sobre un arreglo desordenado, y la búsqueda binaria que requiere que el arreglo esté previamente ordenado para proceder a realizar la búsqueda, particionando el arreglo en sublistas e ir buscando en ellas según el valor buscado sea menor o mayor del elemento al centro de cada sublista.

Ejercicios de autoevaluación del capítulo 4

1. Describa conceptualmente cada método de ordenamiento desarrollado en este capítulo.
2. Revise y analice los algoritmos de ordenamiento implementados en el libro de texto. Haga nuevas implementaciones de estos algoritmos creando nuevos escenarios de aplicación.
3. Describa conceptualmente las implementaciones de las búsquedas secuenciales y binarias explicadas en este capítulo del libro de texto.
4. Revise y analice los algoritmos de búsqueda lineal o secuencial y binaria que se presentan en este capítulo del libro de texto. Cree nuevos escenarios de aplicación de los algoritmos.

5. Complete el siguiente crucigrama con las respuestas correctas



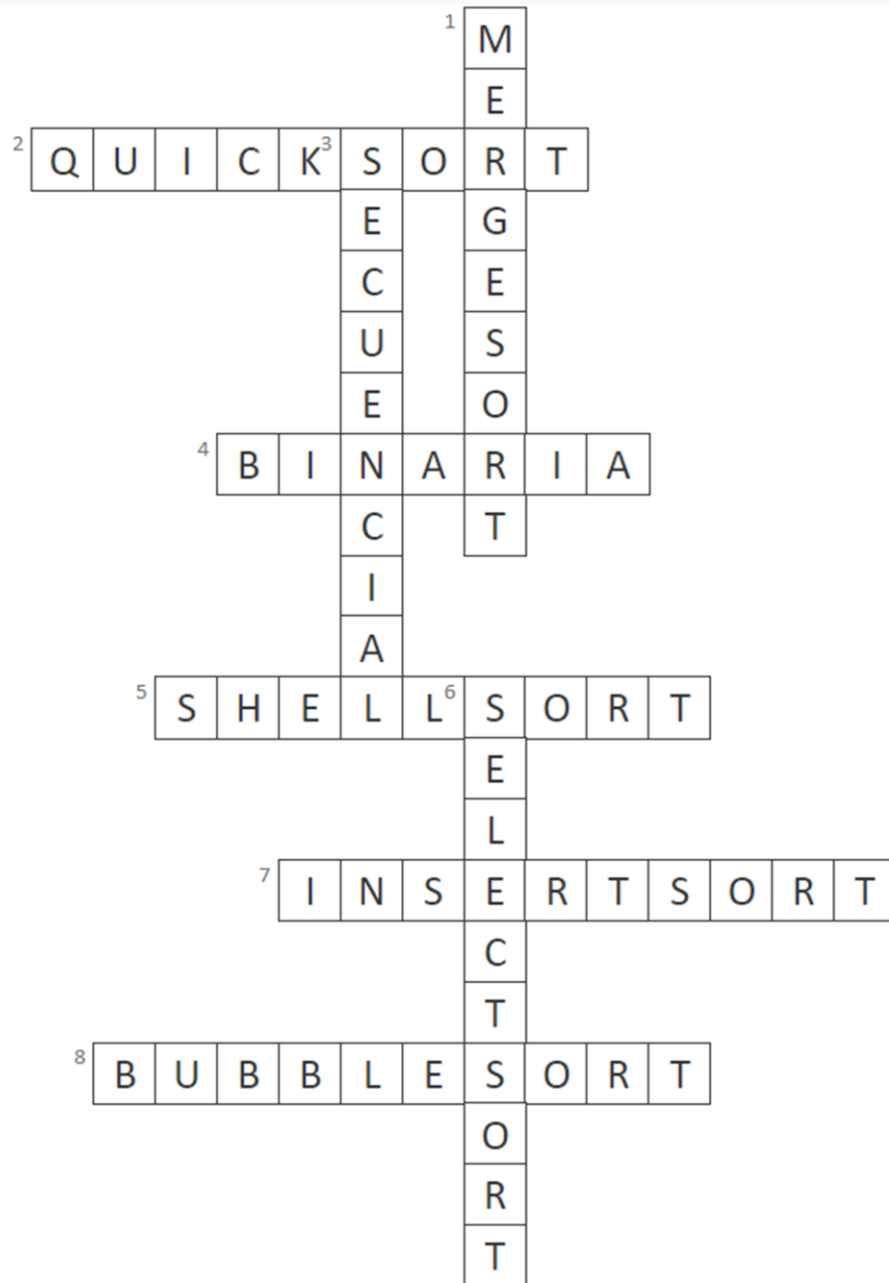
Verticales

1. Utiliza el algoritmo de clasificación por combinación en la que dividimos el problema en subproblemas y resolvemos esos subproblemas individualmente.
3. Realiza una búsqueda en un arreglo de datos ordenado o desordenado, iniciando desde la posición cero hasta $n-1$.
6. Consiste en encontrar el menor de todos los elementos del vector e intercambiarlo con el elemento que está en la primera posición. Se sigue con el segundo elemento más pequeño y se intercambia.

Horizontales

2. Recibe una lista de datos, un dato y un elemento pivote, colocar el dato en la posición correcta de la lista ordenada y luego los elementos más pequeños antes de dicho dato y todos los elementos mayores después.
4. Consiste en reducir paulatinamente el ámbito de búsqueda a la mitad de los elementos, comparando el elemento que se busca en la mitad del intervalo y si es menor en la sublista de la izquierda. Si es mayor lo hará en la sublista de la derecha.
5. Consiste en dividir el arreglo o lista de elementos en bloques (intervalos) de varios elementos para ordenarlos aplicando inserción directa a cada bloque.
7. Consiste en recorrer una lista de datos y seleccionar en cada iteración un valor que funja como clave e irlo comparando con el resto e insertándolo en el lugar correspondiente.
8. Consiste en revisar cada elemento de una lista de datos intercambiándolo con un elemento siguiente si este es mayor que él.

Respuesta al crucigrama



Capítulo 5. Pilas y colas en C++.

5.1. Pilas en C++

Una pila es una colección ordenada de elementos de cualquier tipo (enteros, cadenas, carácter, lógicos, objetos, entre otros). En esta estructura los datos se agregan o se retiran por el mismo extremo el cual se denomina “cima” o “tope”. Estas estructuras tienen muchos usos como implementaciones en sistemas operativos, compiladores, lenguajes de programación, entre otros. Utiliza un enfoque LIFO (Last Input, First Output o Último en Entrar, Primero en Salir, por sus siglas en inglés), que significa que el último elemento en ingresar a la estructura es el primero en salir, como si fuera la estiba de una cantidad de bloques de cemento.

5.2. Colas en C++.

La estructura de datos cola puede contener, al igual que las pilas, cualquier tipo de datos. La diferencia es que en una cola el primer elemento que ingresa a la estructura es el primero en salir. También es muy utilizado en sistemas operativos, lenguajes de programación, compiladores, entre otros. Utiliza un enfoque FIFO (First Input, First Output, que significa Primero en Entrar, Primero en Salir, por sus siglas en inglés). Esto sería como una cola o fila en un banco en el cual el primer cliente que ingresa es el primero que atienden. Existen varios tipos de colas tales como colas simples, colas circulares o colas de prioridad que tienen importantes usos en el manejo de estructuras de datos.

Ambas estructuras, pilas y colas, pueden ser implementadas mediante arreglos o nodos tipo struct. En el primer caso la programación será diferente en el sentido que en el primer caso deberemos manejar los índices del arreglo para simular el mecanismo de implementación de la estructura y, en el segundo caso, utilizaremos apuntadores para manipular los nodos de estas.



Las pilas y colas son importantes estructuras de datos que hay que estudiar y analizar sus mecanismos funcionales. Una vez que se conozca la funcionalidad de cada una de estas estructuras hay que comprender la implementación mediante código programado. Para ello estudia las implementaciones que se realizan en el libro de texto, mediante codificación en C++, en las páginas 105 a 114.

Resumen del capítulo

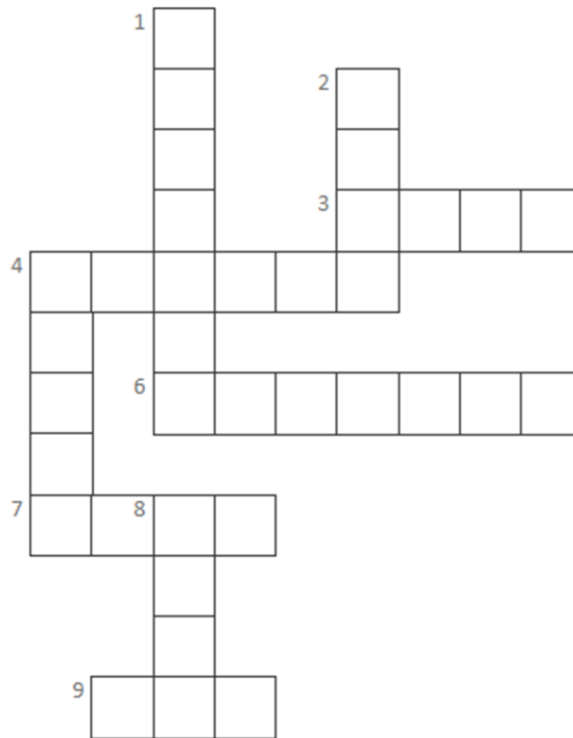
Este capítulo describe, explica e implementa algoritmos para el uso de pilas y colas. Tanto las pilas como las colas tienen una funcionalidad definida, orientada al tratamiento de problemas concretos y es importante conocer y dominar su implementación para posteriormente ser utilizadas en estructuras de datos.

Los algoritmos presentados en el libro de texto utilizan vectores y punteros para su implementación. Es decir, se utilizan librerías predeterminadas de C++ y codificación manual para los mecanismos funcionales de ambas estructuras.

Ejercicios de autoevaluación del capítulo 5

1. Describa conceptualmente el concepto de pila, importancia de uso y mecanismo de implementación.
2. Describa conceptualmente el concepto de cola, importancia de uso y mecanismos de implementación.
3. Revise y analice la implementación de código para el uso de pilas. Haga nuevas implementaciones de este algoritmo creando nuevos escenarios de aplicación.
4. Revise y analice la implementación de código para el uso de colas. Haga nuevas implementaciones de este algoritmo creando nuevos escenarios de aplicación.

5. Complete el siguiente crucigrama con las respuestas correctas



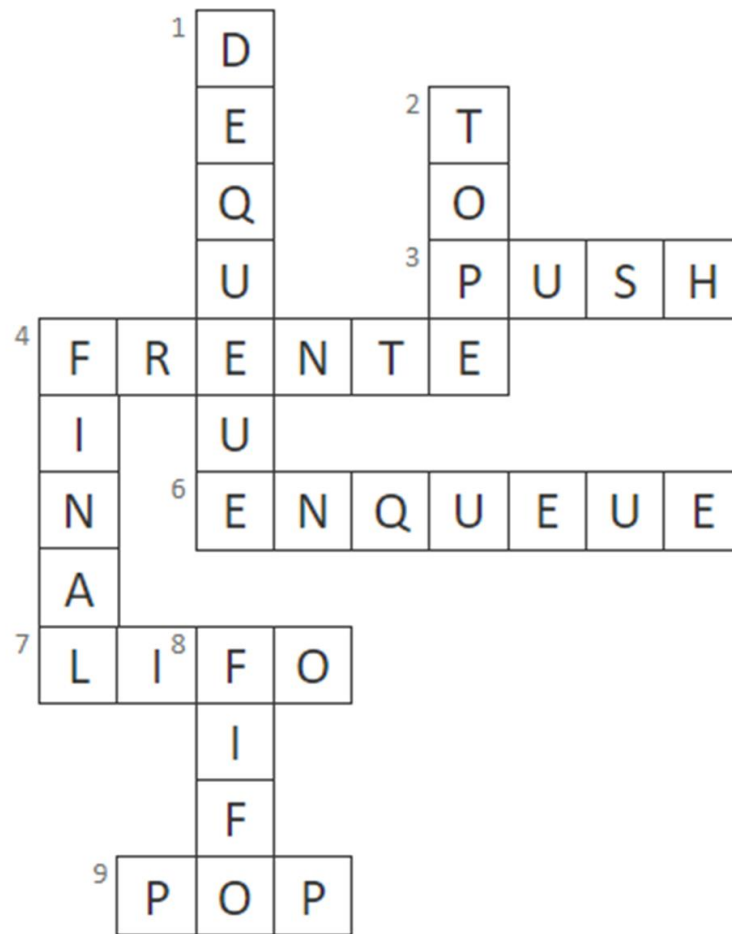
Verticales

- 1. Desencolar elemento
- 2. Primer elemento de una pila
- 4. Ultimo elemento de una cola
- 8. Primero en entrar primero en salir

Horizontales

- 2. Variable que almacena la posición o dirección de memoria donde se guarda un objeto y su contenido
- 3. Meter elemento en pila
- 4. Primer elemento de una cola
- 6. Encolar elemento
- 7. Ultimo en entrar primero en salir
- 9. Sacar elemento de la pila

Respuesta al crucigrama



Capítulo 6. Listas enlazadas en C++.

6.1. Listas simplemente enlazadas en C++

Las listas simplemente enlazadas son estructuras compuestas por nodos y punteros. Los punteros son variables que se utilizan para enlazar nodos y mantener una secuencia entre ellos. Se denominan simplemente enlazadas por cuanto el puntero que tiene cada nodo apunta al siguiente nodo de la lista, sin tener otro puntero que retorne hacia él. Es decir, los punteros solo direccionan hacia delante de la lista de nodos. Las operaciones que se llevan a cabo en estas listas son creación, búsqueda secuencial, modificación de la información del nodo y eliminación del nodo.

6.2. Listas doblemente enlazadas en C++.

A diferencia de las listas simplemente enlazadas, las doblemente enlazadas poseen dos punteros: uno que apunta al siguiente nodo y otro que direcciona al nodo anterior. Es decir, implementan un puntero hacia el siguiente nodo y otro hacia el nodo que le antecede. Se implementan las mismas operaciones de la lista simplemente enlazada: creación, búsqueda secuencial, modificación de la información del nodo y eliminación de nodos. La ventaja es la existencia de nodos siguiente y anterior lo que permite que el recorrido de los nodos de la lista se pueda realizar hacia atrás o hacia adelante.



El concepto, funcionalidad e implementación de listas simple y doblemente enlazadas se desarrollan en las páginas 125 a 137 del libro de texto. Es importante que realice resúmenes del tema, mapas conceptuales o cualquier otro instrumento para tener claro los conceptos. También recurra a la diagramación de las operaciones de inserción, recorridos, búsquedas e eliminación de nodos en estas listas y análisis del código fuente de implementación aportado y la creación de nuevos escenarios de aplicación.

6.3. Listas circulares enlazadas en C++.

Una lista circular enlazada es una lista lineal en la cual el primer nodo apunta al último nodo de esta. Es como una lista simplemente enlazada con la diferencia que el primer nodo apunta al último. Con ello el recorrido de la lista es secuencial pero a través de toda la lista y del primero pasar al último y reiniciar el recorrido.

6.4. Listas circulares doblemente enlazadas en C++.

Una lista circular doblemente enlazada tiene la particularidad de que sus nodos implementan punteros hacia el siguiente nodo y al nodo anterior, siendo que el puntero siguiente del primer nodo apunta al ultimo nodo y el puntero anterior del último nodo apunta al primer nodo.



El concepto, funcionalidad e implementación de listas circulares, simple y doblemente enlazadas se desarrollan en las páginas 137 a 143 del libro de texto. Es importante que realice resúmenes del tema, mapas conceptuales o cualquier otro instrumento para tener claro los conceptos. También recurra a la diagramación de las operaciones de inserción, recorridos, búsquedas e eliminación de nodos en estas listas y análisis del código fuente de implementación aportado y la creación de nuevos escenarios de aplicación.

Resumen del capítulo

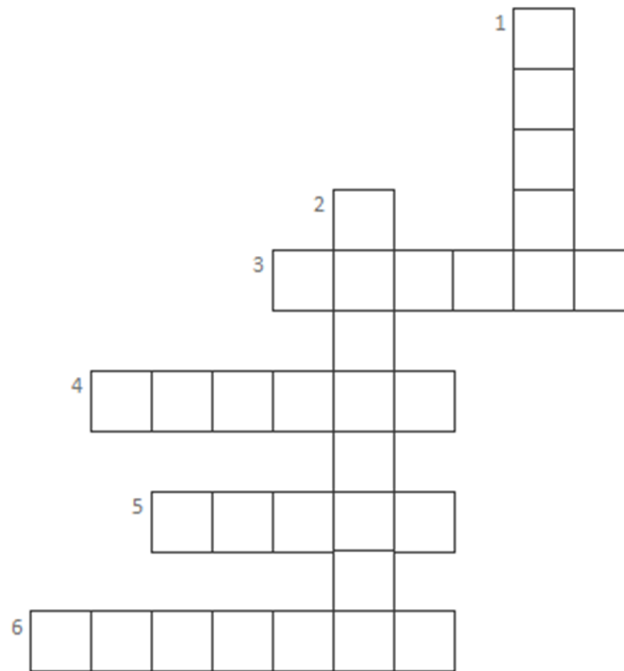
Este capítulo describe, explica e implementa algoritmos para el uso de listas enlazadas lineales y circulares. Las listas simplemente enlazadas poseen un solo puntero el cual direcciona al siguiente nodo de la lista, mientras las doblemente enlazadas poseen dos punteros: uno que direcciona al nodo siguiente y otro al anterior. Con ello se establece un recorrido solo hacia adelante en el primer caso y hacia adelante y hacia atrás en el segundo caso. En cuanto las listas circulares, simple o doblemente enlazadas, se caracterizan por implementar en el primer nodo el puntero siguiente hacia el ultimo y en el ultimo nodo el nodo anterior al primer nodo.

Los algoritmos presentados en el libro de texto se utilizan punteros para la implementación del manejo de nodos, tanto en listas lineales como en circulares. Las operaciones que se realizan sobre estas listas establece objetivos claros, tales como insertar un nodo al principio de la lista, en medio de esta o al final. También los mecanismos de recorrido según la lista.

Ejercicios de autoevaluación del capítulo 6

1. Describa conceptualmente el concepto de listas enlazadas y el contexto en el cual pueden ser utilizadas.
1. Practique los mecanismos de inserción, recorrido y eliminación de nodos en una lista, lineal y circular, simple y doblemente enlazada, mediante esquemas ilustrativos, , siguiendo la lógica de la implementación de código.
2. Revise y analice la implementación de código para el uso de listas simples y doblemente enlazadas. Haga nuevas implementaciones de esto algoritmos creando nuevos escenarios de aplicación.
3. Revise y analice la implementación de código para el uso de listas circulares simple y doblemente enlazadas. Haga nuevas implementaciones de estos algoritmos creando nuevos escenarios de aplicación.

4. Complete el siguiente crucigrama con las respuestas correctas



Verticales

1. Es una forma de estructurar datos de modo que cada nodo apunta al siguiente y el último a nadie
2. Lista ligada que sus punteros final e inicial se unen para formar un anillo

Horizontales

3. Lista ligada que solo tiene punteros hacia adelante
4. Palabra reservada C++ para crear la estructura de un nodo
5. Lista ligada que tiene punteros hacia adelante y hacia atrás
6. Variable que se utiliza para apuntar la posición de un nodo o su información.

Capítulo 7. Árboles en C++.

Los árboles son estructuras de datos muy importantes dado que permiten implementar métodos de búsqueda eficientes. También son importantes porque permiten el almacenamiento de información con cierta estructura concreta que permite su gestión. Entre las estructuras árbol existen árboles generales, binarios, rojinegros, entre otros. En el libro de texto se le dedica principal atención a los árboles binarios dada su uso más intensivo en las estructuras de datos.

7.1. Árboles binarios

Los árboles binarios se caracterizan por que cada nodo padre tiene mínimo un nodo hijo y máximo dos. Cuando un nodo no tiene hijos se conoce como un nodo hoja. La tabla 7.1. muestra los tipos de árboles binarios.

Tabla 7.1. Tipos de árboles binarios

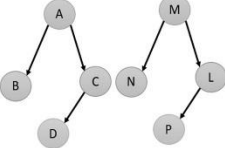
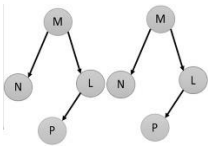
Imagen ilustrativa	Tipo de árbol	Breve descripción
	Distinto	Se consideran árboles distintos cuando las estructuras que las componen son diferentes.
	Similar	Dos árboles se consideran similares cuando sus estructuras son similares.
	Equivalente	Dos árboles se consideran equivalentes cuando son similares en cuanto estructura y sus nodos contienen la misma información.

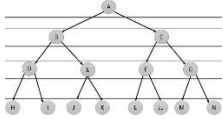
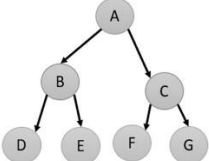
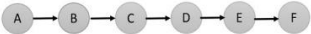
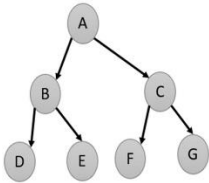
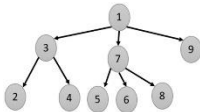
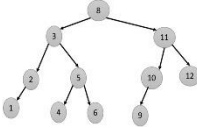
Imagen ilustrativa	Tipo de árbol	Breve descripción
	Completo	Un árbol binario es completo cuando cada nodo tiene cero o dos hijos. Existen variantes como totalmente balanceado o balanceado a la izquierda.
	Lleno	Un árbol binario lleno es aquel donde todos los nodos tienen dos hijos o ninguno.
	Degenerado	Un árbol binario degenerado es aquel en el cual los niveles son ocupados por un solo nodo y cada uno tiene un hijo. Corresponde a una lista simplemente enlazada.
	Perfectamente equilibrado	Es aquel en el que el número de nodos para todos y cada uno de los que conforman dicho árbol, tanto los que están en el subárbol izquierdo como en el derecho, deben diferir a lo mucho en una unidad.
	Multcamino	Un tipo de árbol multcamino constituye una estructura de datos no lineal, homogénea y donde cada nodo puede tener una cantidad n de hijos.

Imagen ilustrativa	Tipo de árbol	Breve descripción
	ABB	Un árbol ABB es aquel en el que sus nodos están ordenados de alguna manera: por ejemplo, por orden alfabético o numérico.

7.1.9. Recorrido de árboles binarios.

El recorrido de un árbol binario consiste en visitar cada nodo de éste en un orden determinado. Existen dos tipos de recorridos de árboles binarios: en amplitud y en profundidad. El recorrido en amplitud es aquel que transita por el árbol por niveles (del nivel superior o raíz hasta los niveles inferiores). El recorrido por profundidad recorre el árbol por medio de sus subárboles. Este tipo de recorrido se puede llevar, a su vez, de tres formas posibles: *preOrder*, *inOrder* y *postOrder*.

Para realizar el recorrido *preOrder* se deben ejecutar tres operaciones: 1. Visitar la raíz, 2. Recorrer el subárbol izquierdo en orden previo y 3. Recorrer el subárbol derecho en orden previo. La figura 7.1. muestra un ejemplo de este tipo de recorrido.

El recorrido *inOrder* también ejecuta tres operaciones: 1. Recorrer el subárbol izquierdo en orden, 2. Visitar la raíz y 3. Recorrer el subárbol derecho en orden. La figura 7.2. muestra un ejemplo de este tipo de recorrido.

El recorrido *postOrder* los pasos para realizarse son: 1. Recorrer el subárbol izquierdo en orden posterior, 2. Recorrer el subárbol derecho en orden posterior y 3. Visita la raíz. La figura 7.3. muestra este tipo de recorrido:

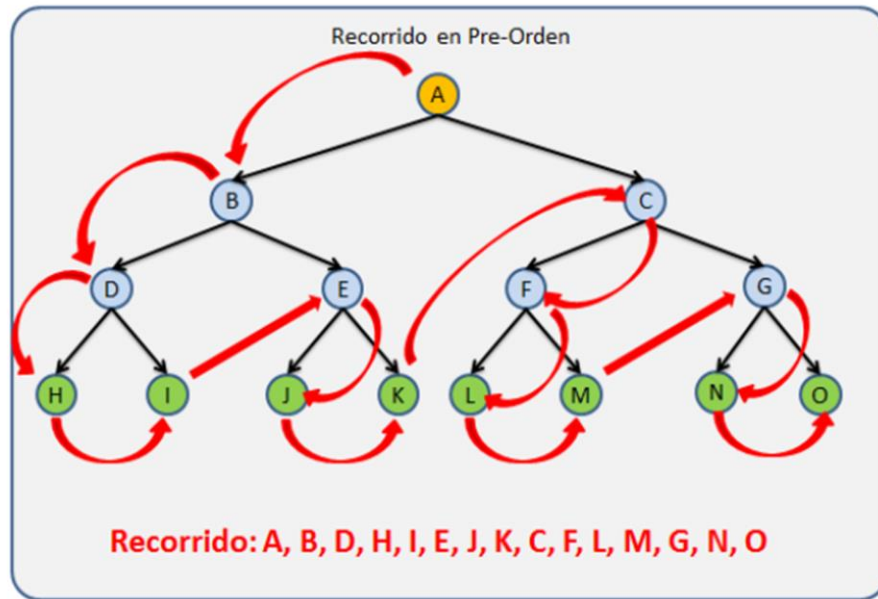


Figura 7.1. recorrido en *preOrder*.

Fuente: <https://estructuras2unincca.wordpress.com/2018/10/02/arbol-binario/>

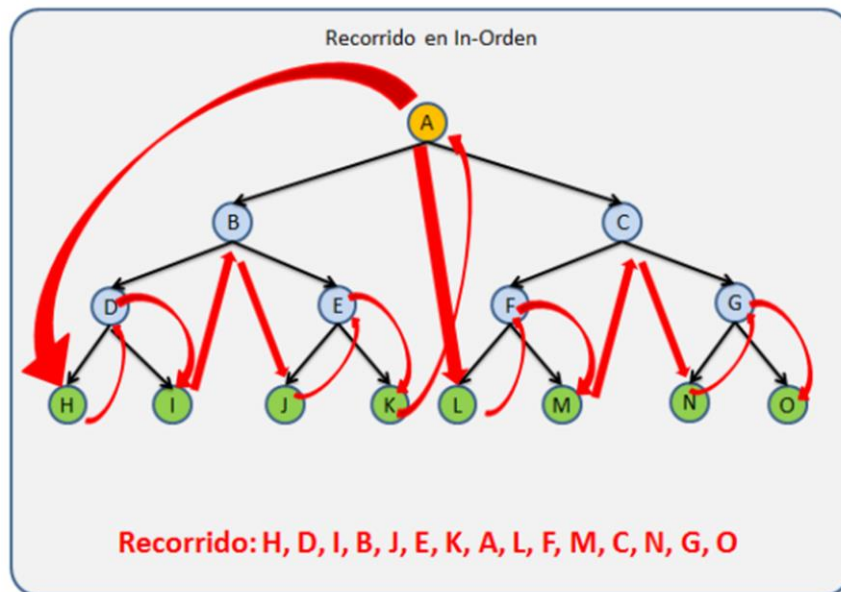


Figura 7.2. recorrido en *inOrder*.

Fuente: <https://estructuras2unincca.wordpress.com/2018/10/02/arbol-binario/>

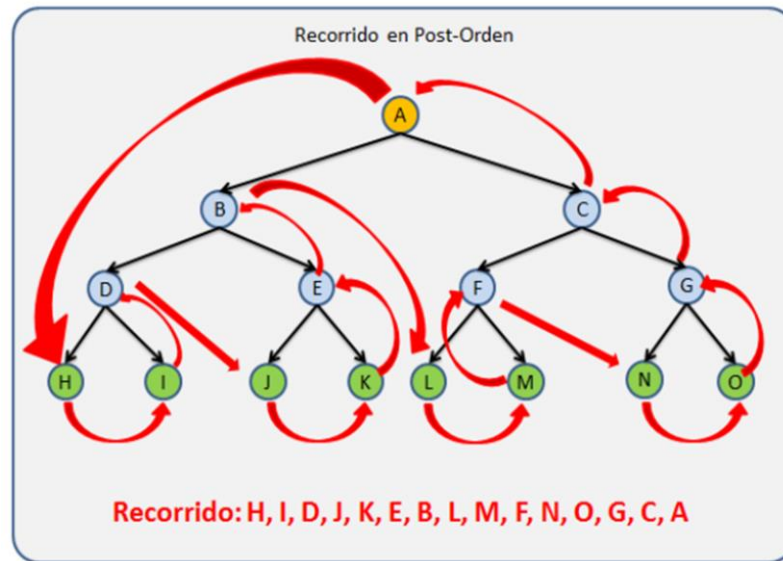


Figura 7.3. recorrido en *postOrder*.

Fuente: <https://estructuras2unincca.wordpress.com/2018/10/02/arbol-binario/>



Los conceptos de árboles binarios se desarrollan en las páginas 149 a 154 y los recorridos de éstos en las páginas 154 a 162. Es importante comprender cada tipo de árbol binario para generalizar su concepto. El recorrido que se realiza es un aspecto que también debe asimilarse por la importancia que tiene en la usabilidad de este tipo de estructura. El código de implementación para el mantenimiento de árboles binarios y los recorridos citados también se consideran en estos rangos de páginas y que deben ser concienzudamente estudiados y asimilado su funcionamiento.

7.1.11. Árbol binario AVL.

Los árboles binarios son muy eficaces para mantener un conjunto de nodos ordenados de alguna forma y a la vez realizar búsquedas sobre ellos. Sin embargo, cuando la estructura crece puede enfrentarse a una distribución de nodos desequilibrada. Supóngase que un ABB (árbol binario de búsqueda) que está ordenado numéricamente con respecto a un valor raíz, crece con más números mayores al número raíz. Ahí tendríamos un subárbol derecho más cargado que el subárbol izquierdo. Por ende, debemos acudir a lo que

se denomina un balanceo de los nodos que pertenecen a este árbol binario. Aquí es donde resurge el concepto AVL que es un árbol balanceado.

El procedimiento de balanceado se realiza mediante rotaciones de nodos a la derecha o a la izquierda según se detecte la carga máxima del árbol. Esto está delimitado por el factor de equilibrio que es la diferencia entre las alturas del subárbol derecho y el izquierdo. La fórmula es: $FE = \text{altura del subárbol derecho} - \text{altura subárbol izquierdo}$. Para un árbol AVL este valor debe ser -1, 0 o 1.



Las rotaciones de nodos para balancear un árbol binario y el mecanismo asociado más la implementación de código se ilustran en las páginas 162 a 168. Deben estudiarse y analizarse las figuras 7.23 y 7.25 sobre la combinación de balanceos y la implementación en el listado 7.2. que debe descargarse del sitio web de la editorial.

Resumen del capítulo

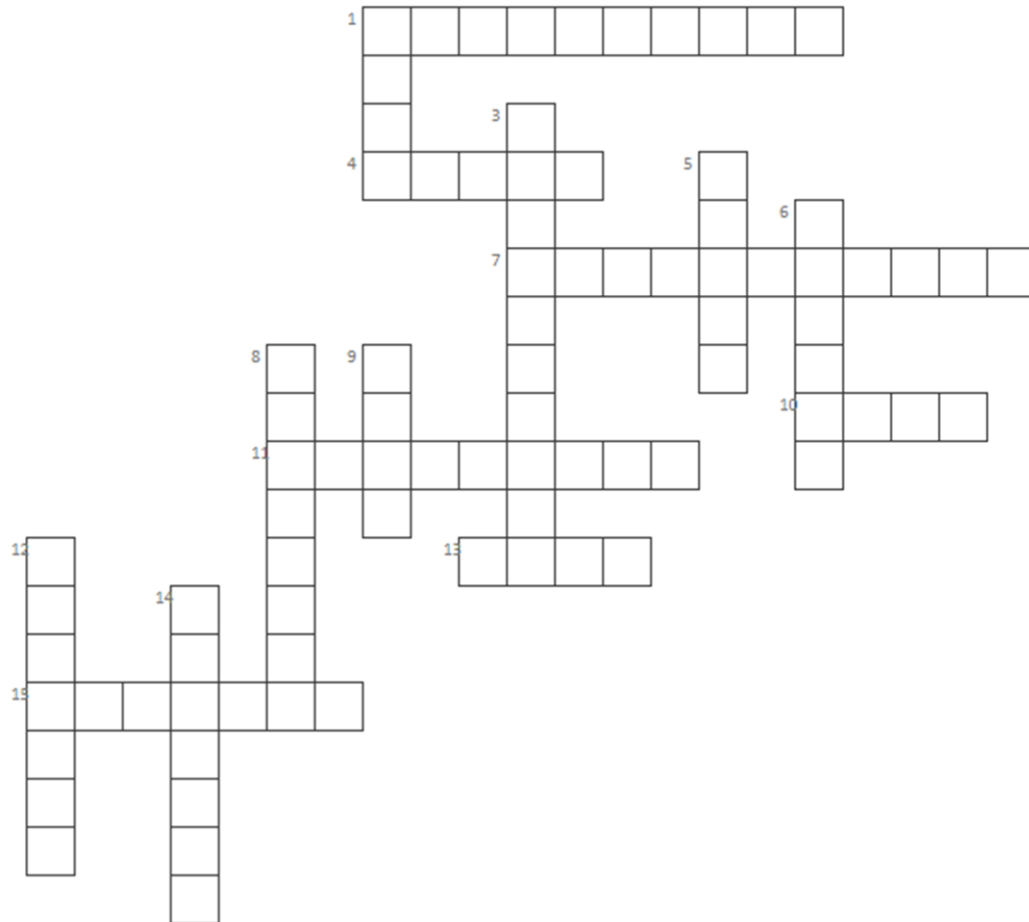
Este capítulo describe, explica e implementa algoritmos para el manejo de árboles binarios de búsqueda (ABB) y su representación balanceada (AVL). Se enumeran los tipos de árboles binarios existentes, según la estructuración de sus nodos a través de sus relaciones. Sin embargo, el libro de texto se enfoca principalmente en la conceptualización e implementación de árboles binarios de búsqueda (ABB) y la implementación de balanceos de nodos a través de árboles AVL.

Asimismo, se conceptualizan e implementan los tipos de recorridos que se pueden realizar a través de sus nodos en estas estructuras lineales. Se conceptualizan recorridos en amplitud y en profundidad y se implementa el código requerido para su funcionalidad.

Ejercicios de autoevaluación del capítulo 7

2. Defina conceptos claros como nodo raíz, nodo hoja, nodo interior, altura y nivel de un árbol.
3. Enumere los tipos de árboles binarios explicados en el libro de texto y conceptualice cada uno de ellos.
4. Defina que es un árbol binario de búsqueda ABB y su importancia de uso.
5. Practique los mecanismos de inserción, recorrido y eliminación de nodos en un árbol binario, mediante esquemas ilustrativos, siguiendo la lógica de la implementación de código.
6. Enumere y explique los tipos de recorridos: amplitud y profundidad de un árbol binario. Para el recorrido en profundidad defina los tipos inOrden, preOrden y postOrden.
7. Practique los mecanismos de recorridos de árboles binarios, mediante esquemas ilustrativos, siguiendo la lógica de la implementación de código, tanto en amplitud como en profundidad.

8. Complete el siguiente crucigrama con las respuestas correctas



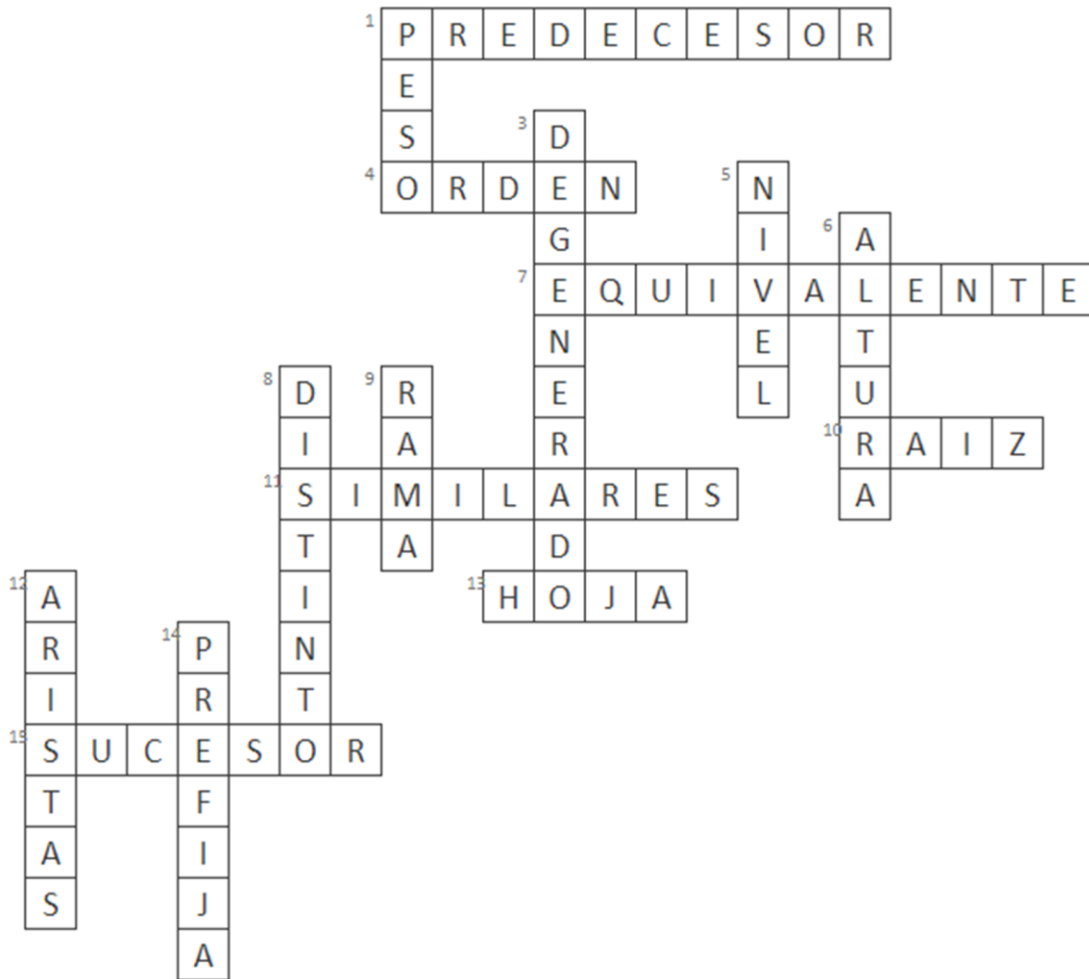
Verticales

1. Cantidad de nodos de un árbol considerando sus diferentes niveles
3. Tipo de árbol donde cada nodo solo tiene un hijo.
5. Es la distancia que un nodo tiene hasta la raíz del árbol
6. Número máximo de niveles que puede tener un árbol
8. Cuando en el árbol binario las estructuras que la componen son diferentes
9. Permiten unir nodos entre si (padres a hijos)
12. Permiten unir nodos entre si (padres a hijos)
14. Recorrido PreOrden

Horizontales

1. Nodo que antecede al nodo actual
4. Es el máximo de hijos que puede tener un nodo
7. Cuando en el árbol binario con estructura e información similar.
10. Nodo que el mayor al subárbol izquierdo y menor al derecho
11. Cuando en el árbol binario con estructura e información diferentes
13. Nodo que no tiene hijos
15. Nodo que está después del nodo actual

Respuesta al crucigrama



Capítulo 8. Grafos en C++.

8.1. Aplicabilidad de los grafos

Los grafos son estructuras de datos dinámicas que permiten representar, mediante aristas y nodos, muchas aplicaciones del mundo real como lo son carreteras de una ciudad, las líneas férreas de un sistema ferroviario, las rutas de vuelos, una red de comunicaciones, entre otras. La red social de Facebook® por ejemplo es una compleja red de conexiones entre usuarios.

8.2. Tipos de grafos

En estructuras de datos existen muchos tipos de grafos entre los cuales podemos citar: simples, dirigidos, no dirigidos, etiquetados o ponderado, aleatorios, hipergrafo, infinitos, entre otros. Cada uno tiene su propia usabilidad y su implementación dependerá del problema que se requiere resolver.



Algunos tipos de grafos se explican en el libro de texto en las páginas 176 a 181. Es importante comprender su funcionalidad y las características que tiene cada uno de ellos. Analice concienzudamente su definición y visualice las posibles aplicaciones de uso.

8.3. Representación de un grafo

Los grafos pueden ser representados mediante matriz de adyacencias, listas de adyacencias y listas de arcos. La matriz de adyacencia es la representación matricial de las relaciones entre nodos, simulando la existencia de aristas entre dichos nodos. En la matriz el valor 1 representaría que existe tal relación, mientras el cero no. La lista de adyacencia representa una lista enlazada vertical donde se registran los nodos del grafo y una lista enlazada horizontal que representa las

relaciones de cada nodo. Estas representaciones se muestran en la figura 8.1.

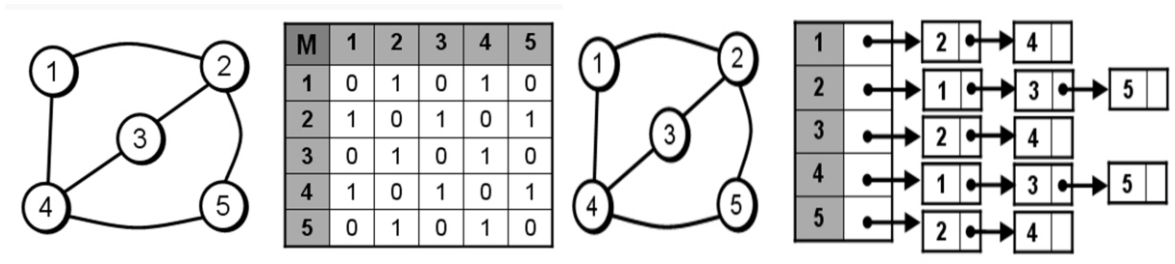


Figura 8.1. representación de grafo mediante matriz y lista de adyacencia

Fuente: <https://jonathanhigueravelasquez.wordpress.com/representacion-de-los-grafos/>

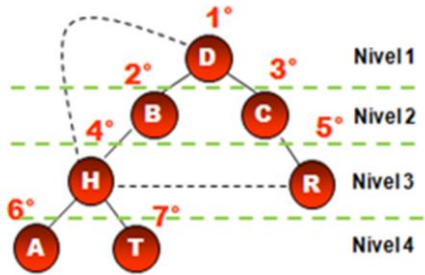
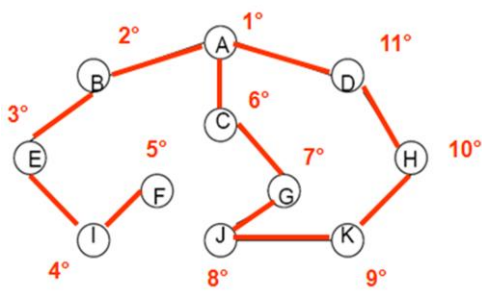
La lista de arcos mantiene una lista de nodos (igual que en las listas de adyacencia) pero no se guardan los nodos vecinos, sino que se mantiene una lista de arcos para cada nodo donde se tiene la información, permitiendo con ello determinar el elemento del nodo origen como el del destino.

8.4. Algoritmo de recorrido y búsqueda en grafos

El recorrido de un grafo consiste en iniciar el trayecto existente desde un nodo a otro. Esto siguiendo las aristas que existen entre dichos nodos, siendo en un solo sentido en un grafo dirigido y pudiendo devolverse cuando es uno no dirigido. Existen dos tipos de recorridos, al igual que en los árboles binarios: en anchura y en profundidad. Estas dos estrategias de recorrido tienen como finalidad navegar a través del grafo buscando información sea para obtenerla, modificarla, eliminarla o cualquier otro procedimiento de mantenimiento del grafo.

El resumen de estos dos tipos de recorridos se muestran en la tabla 8.1.

Tabla 8.1. Recorridos (búsquedas) en un grafo

Tipo de recorrido	Breve descripción	Imagen ilustrativa
Amplitud o anchura (BFS: Breadth First Search)	Algoritmo que inicia el recorrido de los nodos de un árbol por niveles, iniciando un nodo origen y visitando todos los nodos adyacentes al nodo raíz.	 <p>Tomado de: https://pazmorales.wordpress.com/unidad-iii/recorrido-de-grafos/</p>
Profundidad (DFS: Depth First Search)	Corresponde a un recorrido en pre orden de un árbol. Consiste en seleccionar el nodo raíz como partida, marcándolo como visitado y luego recorriendo, recursivamente, cada nodo adyacente no visitado al nodo raíz.	 <p>Tomado de: https://pazmorales.wordpress.com/unidad-iii/recorrido-de-grafos/</p>



En el libro de texto se tratan conceptualmente estos dos tipos de recorridos en las páginas 190 a 199. Asimismo, se implementa el código en C++ para demostrar su funcionalidad. Es importante estudiar y comprender tanto el manejo conceptual de grafos como su implementación mediante código.

Resumen del capítulo

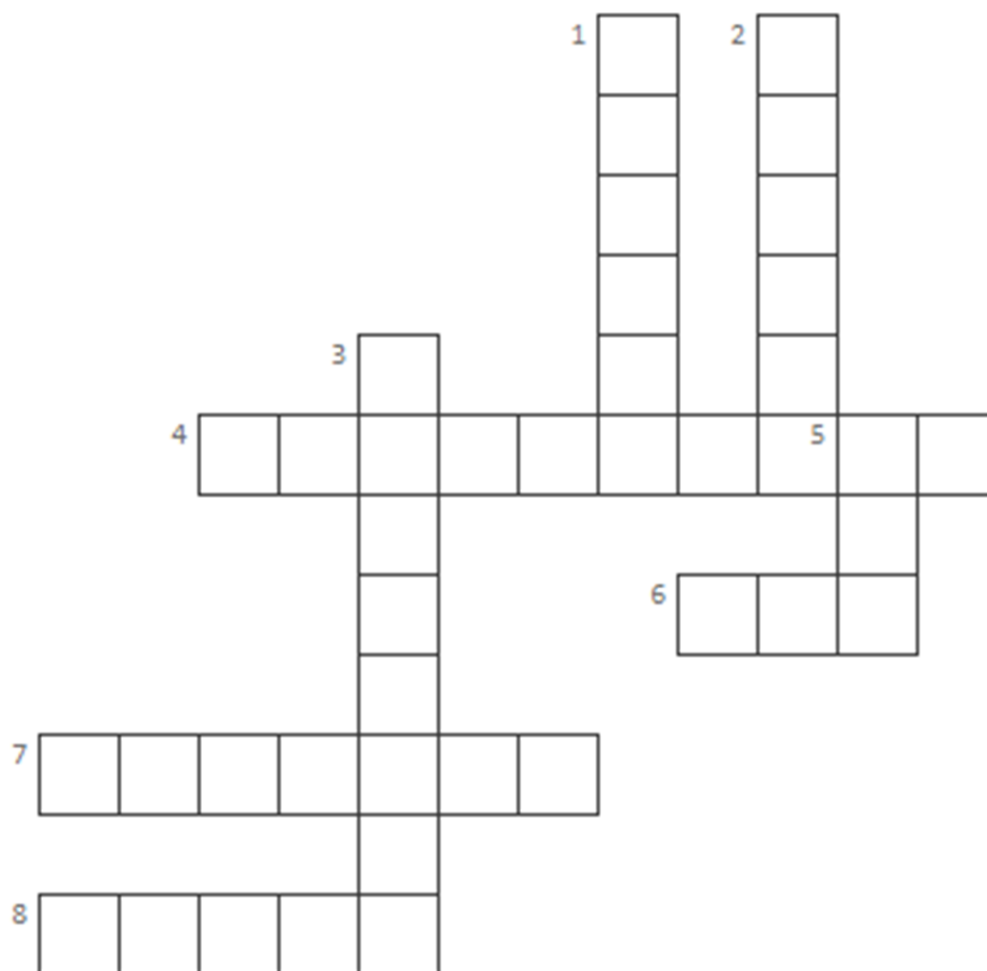
Este capítulo describe, explica e implementa algoritmos para el uso de grafos. Se explica la aplicabilidad de los grafos en ciertas actividades de la vida real: carreteras entre ciudades, distancias entre destinos, la red de comunicaciones de una empresa, entre otros. También los tipos de grafos existentes, caracterizando los grafos dirigidos, no dirigidos, ponderados, aleatorios, entre otros.

Otro aspecto desarrollado en este capítulo es como se representan los grafos: mediante una matriz de adyacencia, una lista de adyacencia o una lista de arcos. Cada uno tiene su propia implementación y mecanismos de aplicación. Finalmente, se desarrolla el tema de los recorridos de grafos a través de sus nodos, utilizando búsquedas en amplitud o en profundidad. Cada uno implementa su mecanismo de uso y principalmente utilizando recursividad.

Ejercicios de autoevaluación del capítulo 8

1. Describa cuáles son los usos que se le pueden dar a los grafos.
2. Conceptualice los tipos de grafos existentes y su representación.
3. Practique los mecanismos de inserción, recorrido y eliminación de nodos en un grafo, mediante esquemas ilustrativos, estudiando el código facilitado en el libro de texto.
4. Practique los mecanismos de recorridos de grafos, mediante esquemas ilustrativos, siguiendo la lógica de la implementación de código, tanto en amplitud como en profundidad.
5. Estudie y asimile los conceptos presentados en las páginas 199 y 200 del libro de texto.

6. Complete el siguiente crucigrama con las respuestas correctas



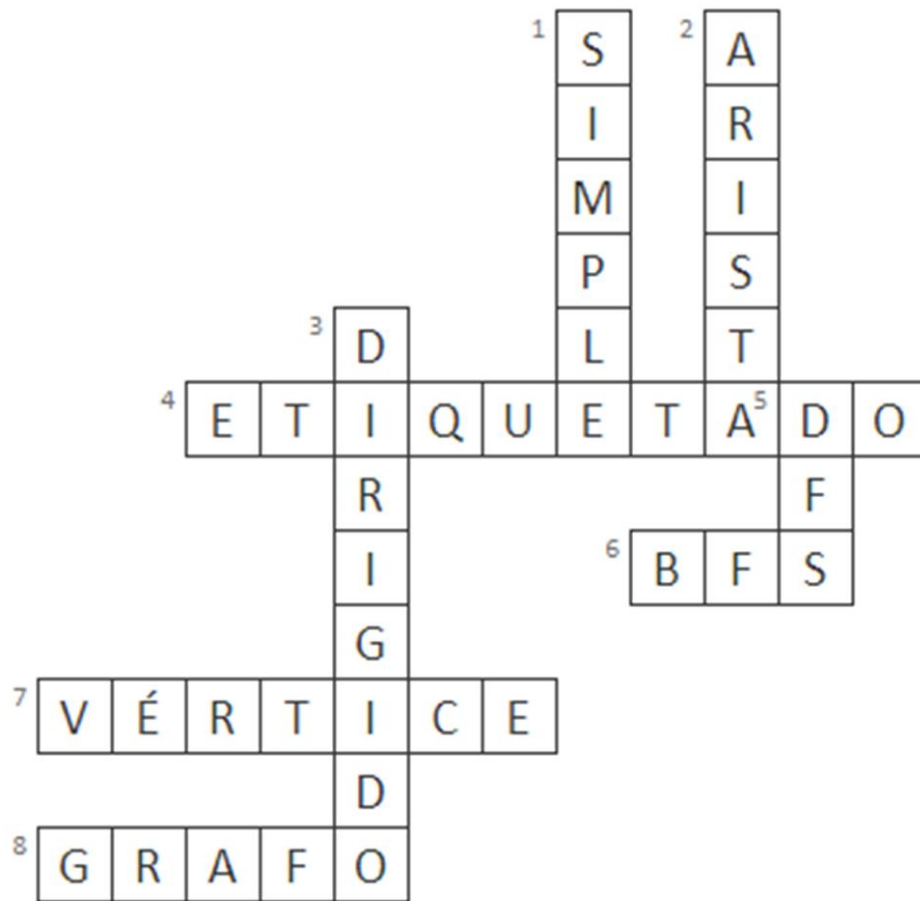
Verticales

1. Este tipo de grafo solo acepta una arista que une dos vértices cualquiera
2. Se le denomina arco también y sirve para unir dos nodos o vértices.
3. Tipo de grafo donde las aristas solo tienen un sentido que va de un nodo a otro, sin una devolución de sentido.
5. Tipo de búsqueda en profundidad en el recorrido de un grafo

Horizontales

4. Tipo de grafo donde las aristas son ponderadas con un peso que debe ser un número.
6. Tipo de búsqueda en anchura en el recorrido de un grafo
7. Registro que contiene un dato de interés y al menos un puntero para referenciar (apuntar) a otro nodo.
8. Modelo que representa relaciones entre elementos (nodos) de un conjunto

Respuesta al crucigrama



Capítulo 9. STL: Standard Template Library

STL (Standard Template Library) es una librería que contiene muchos métodos y funcionalidades que facilitan las labores de programación de un desarrollador de software. Está compuesto por contenedores, iteradores y funciones que, mediante plantillas, es muy sencillo de implementar estructuras de datos.

9.1. Contenedores STL

Los contenedores STL se clasifican en dos categorías: lineales y asociativos. Sobre estos contenedores se pueden realizar operaciones de creación, búsquedas, eliminaciones, entre otras. La figura 9.1. muestra los componentes de las categorías de contenedores STL.

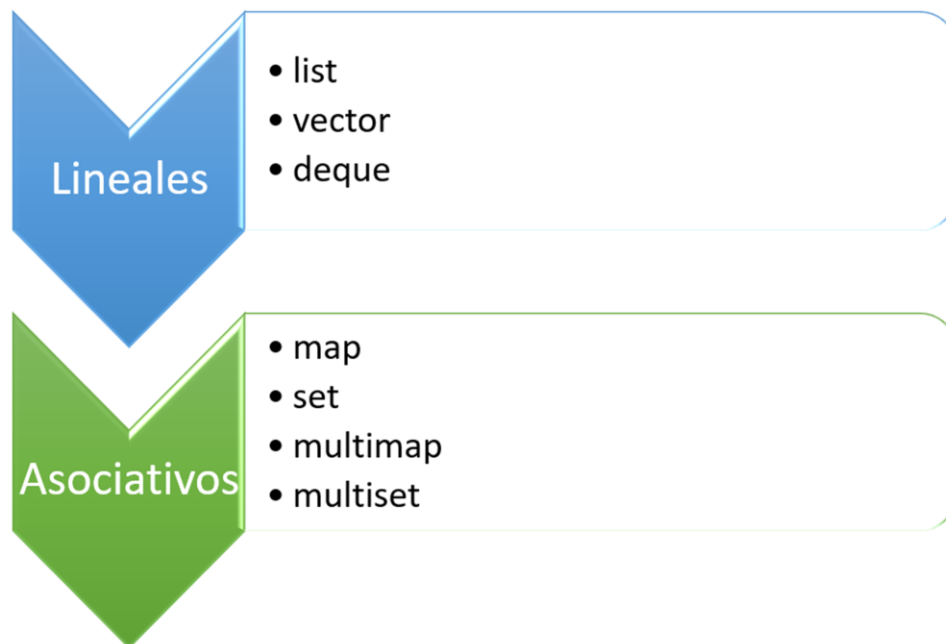


Figura 9.1. elementos de las categorías de contenedores en STL

Fuente: creación propia



La descripción conceptual y funcional de cada elemento mostrado en la figura 9.1. se desarrollan en las páginas 210 a 215. Asimismo, se desarrollan ejemplos programados que demuestran su usabilidad.

9.2. Iteradores

Un iterador actúa como un puntero en clases como vector o list de la librería STL. Mediante un iterador se puede navegar entre los elementos que se almacenan en estructuras como vectores, listas, pilas o colas, evitando el uso de punteros en C++ tal como se implementaron en los primeros 8 capítulos del libro de texto. Estos iteradores pueden navegar hacia adelante, bidireccionalmente o de acceso aleatorio. Dependerá de la estructura que se requiera manipular para usar uno u otro iterador.

9.3. Algoritmos STL

La librería STL proporciona una serie de algoritmos predeterminados que pueden ser utilizados en la manipulación de las clases que se muestran en la figura 9.1. Esto facilita mucho la tarea del programador y permite una mayor calidad en la codificación. La librería *algorithm* es la que contiene todos los métodos de STL. Estos métodos pueden ser matemáticos, de ordenamientos, búsquedas, entre otros.



La descripción conceptual y funcional de las clases y componentes de la librería STL para C++ se desarrollan en las páginas 218 a 230. Es importante asimilar los conceptos de contenedores y el manejo de iteradores de STL para la escritura de programas orientados a las estructuras de datos de una manera eficiente y moderna.

Resumen del capítulo

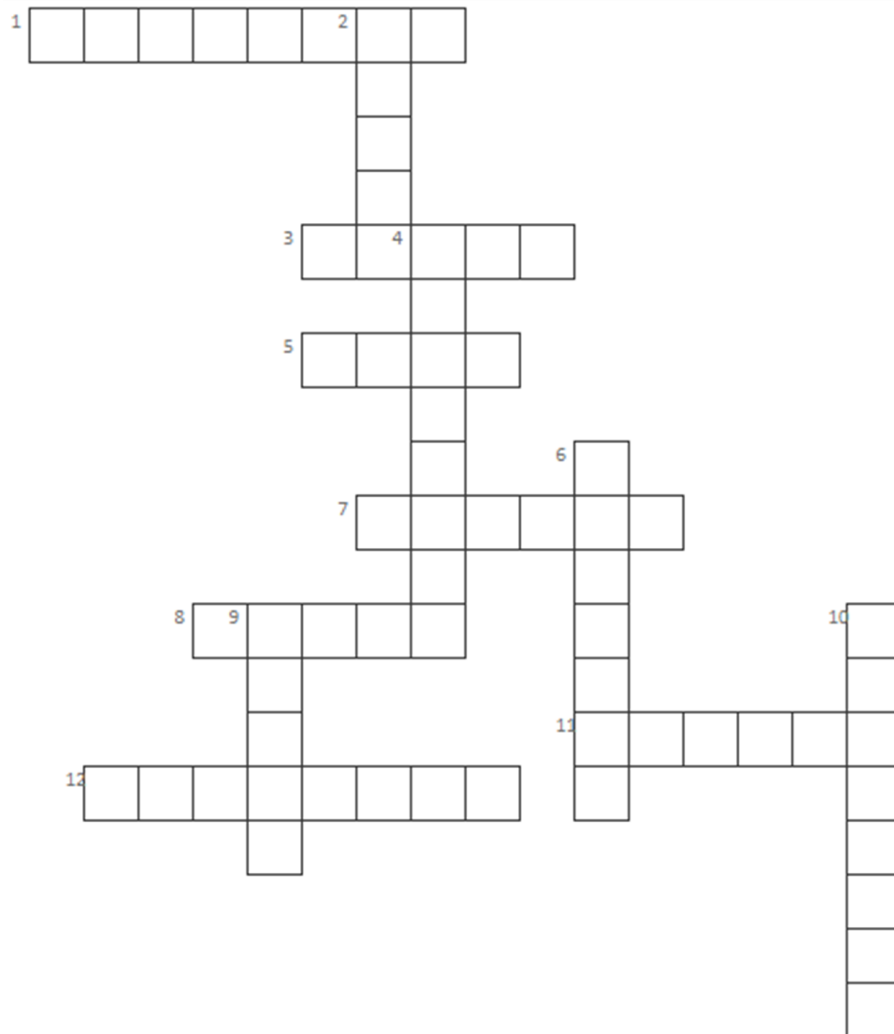
Este capítulo conceptualiza la librería STL de C++. Esta es una librería estandarizada que permite escribir programas orientados al manejo de estructuras de datos de una manera ágil y moderna. Los contenedores e iteradores facilitan la tarea de almacenamiento, ordenamiento y búsqueda de datos.

Se implementan programas demostrativos del uso de las clases contenedoras y el uso de iteradores para los recorridos de dichas estructuras. También se utilizan métodos especializados en el ordenamiento de datos y la búsqueda de estos en contenedores definidos.

Ejercicios de autoevaluación del capítulo 9

1. Defina los conceptos contenedores e iteradores en STL.
2. Enumere y explique los algoritmos que logró identificar en los programas y texto explicativo del libro de texto.
3. Realice ejercicios alternativos, replicando el código facilitado en el libro de texto para el manejo de diversos contenedores STL.

4. Complete el siguiente crucigrama con las respuestas correctas



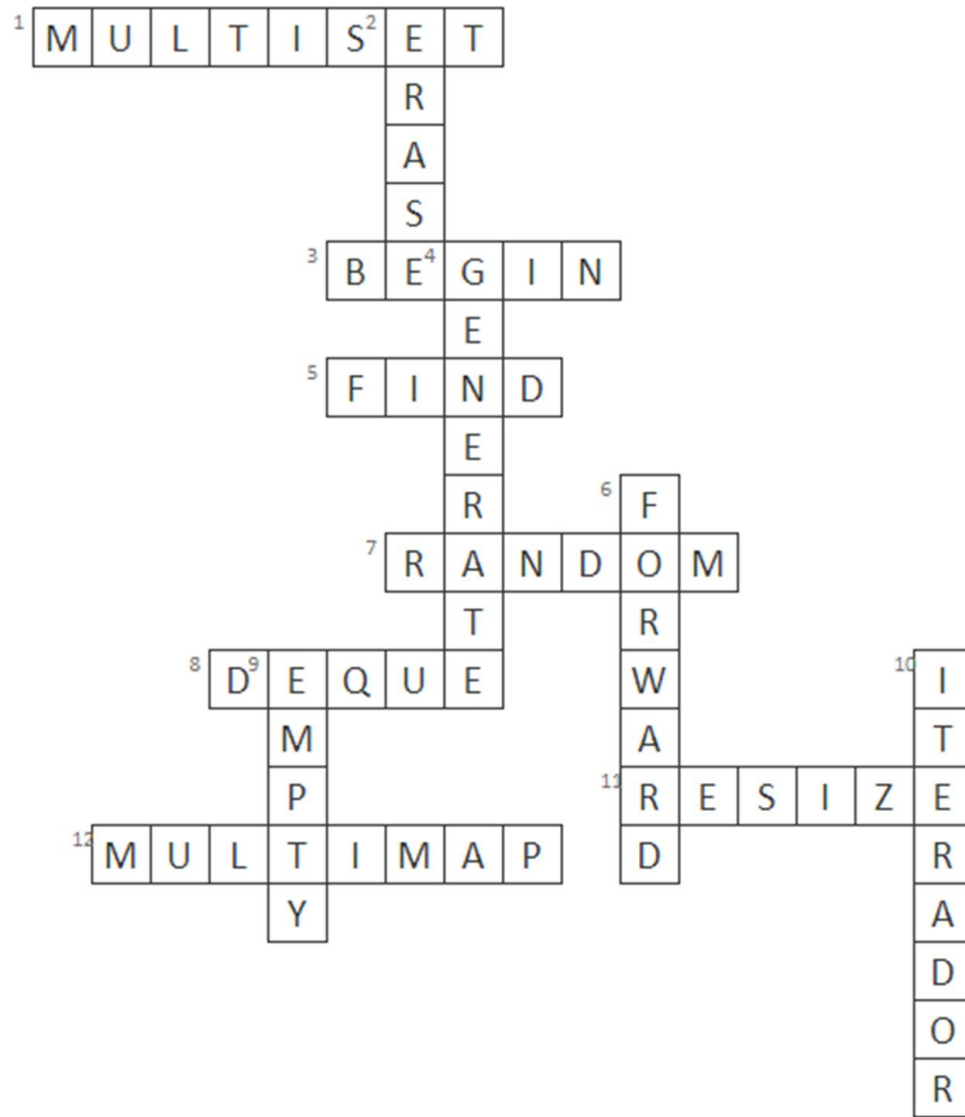
Verticales

- 2. Permite eliminar un elemento almacenado en un contenedor
- 4. Permite crear valores aleatorios
- 6. Iterador que solo avanza hacia adelante
- 9. Devuelve si el contenedor está vacío o no
- 10. Sinónimo de puntero en contenedores STL

Horizontales

- 1. Es un contenedor asociativo
- 3. Permite controlar el inicio de elementos de un contenedor
- 5. Permite buscar un elemento dentro de un contenedor
- 7. Iterador que puede avanzar o retroceder dinámicamente entre elementos
- 8. Contenedor con función de expansión y contracción en extremos
- 11. Permite redimensionar el tamaño de un contenedor dinámico
- 12. Permite la duplicación de elementos en su conjunto

Respuesta al crucigrama



Capítulo 10. Manejo de archivos en C++

El manejo de archivos es una actividad importante para el mantenimiento persistente de datos. Las estructuras de datos bien podrían crearse obteniendo la información desde un archivo de texto o binario que esté almacenado en algún tipo de dispositivo, dígame disco duro, USB o cualquier otro. También es importante porque creada una estructura de datos ésta puede guardarse en un archivo persistente para ser utilizado posteriormente.

10.1. Ficheros

Los ficheros constituyen archivos de datos que se relacionan entre si de forma permanente en un dispositivo no volátil o persistente. A través de una archivo o fichero podemos mantener la información: desde su creación, modificación o eliminación.

10.1.1. Tipos de ficheros

Los tipos de ficheros que se pueden crear, modificar o eliminar son de dos tipos: de texto o binario. En el primer tipo (de texto) la información se maneja mediante caracteres alfanuméricos, son de texto plano y puede ser editados por cualquier editor de texto. Los de segundo tipo (binarios) almacena los datos en formato binario (ceros y unos) y no se pueden editar con cualquier editor de texto.

10.1.2. Memoria intermedia (buffers)

Para poder procesar los archivos de texto o binarios se requiere la utilización de áreas de memoria RAM denominadas *buffers*. Estas áreas funcionan como repositorios temporales de datos dado que la interacción procesador, RAM y otros dispositivos es generalmente lenta y por tanto

requieren un espacio donde cargar temporalmente los datos que procesan.

Para el manejo de ficheros se utilizan clases que permiten el flujo de datos entre la unidad de disco y la memoria principal. Estas clases son *ifstream*, *ofstream* y *fstream*. La figura 10.1. muestra el propósito de estas 3 clases.

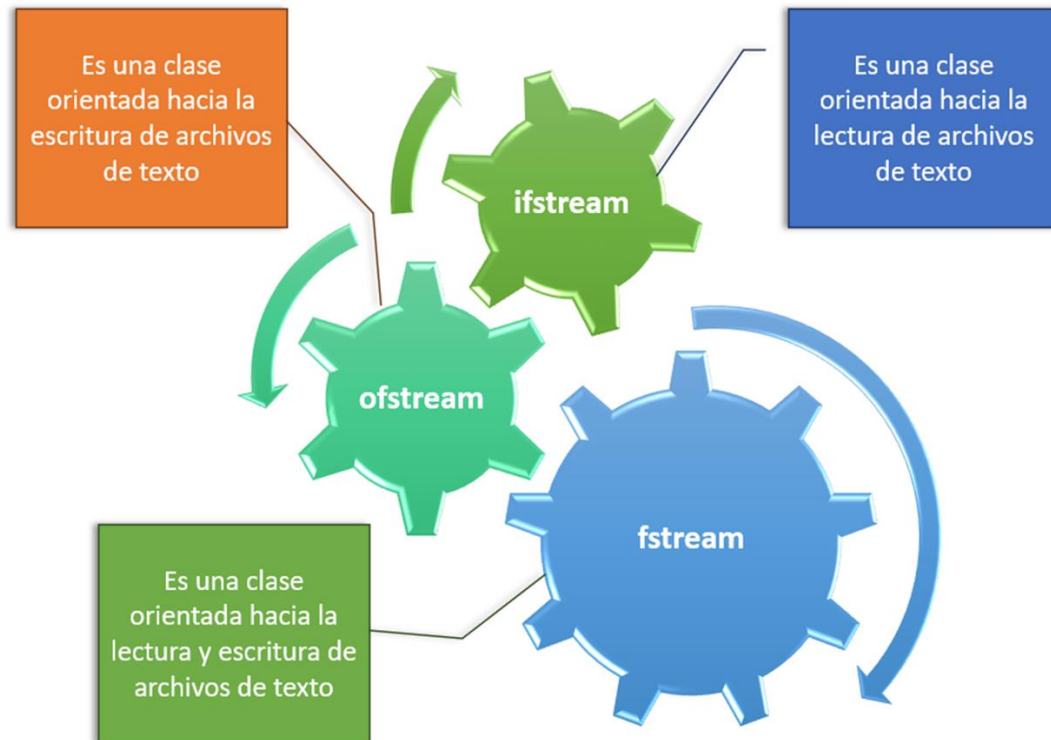


Figura 10.1. clases para la manipulación de archivos de texto

Fuente: creación propia



El libro de texto desarrolla este importante tema del manejo de archivos de texto y binario de forma resumida dado que es un conocimiento que el estudiante de estructura de datos debe haber adquirido en cursos previos. Sin embargo, es importante que repasen la teoría sobre el mecanismo de gestión de archivos, tanto de texto como binarios, estudiando concienzudamente las páginas 237 a 246 y la implementación mediante código en C++ que se explica en estas páginas.

Resumen del capítulo

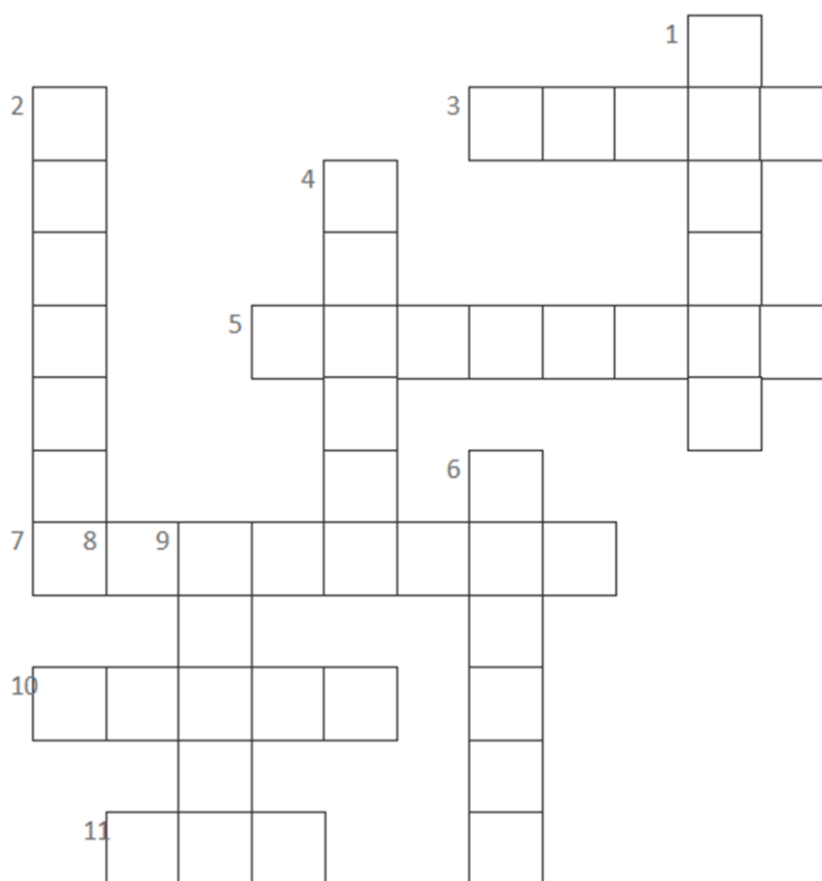
El capítulo 10 del libro de texto es una breve descripción conceptual y aplicada del uso de archivos de texto y binarios en C++. El tema es importante por cuanto se podrían usar archivos para almacenar, modificar o recuperar datos provenientes de alguna de las estructuras anteriormente desarrolladas. Por ejemplo, se podría almacenar la información de arboles binarios, listas enlazadas o grafos en un archivo, de texto o binario, para recuperarlo luego y continuar alimentándolo de información.

Los algoritmos presentados en este capítulo permiten el manejo básico de archivos de texto y binarios, lo que merece el análisis concienzudo de manejo e implementación para poder usarlos como ayuda en el almacenamiento de datos de otras estructuras. Se explican el manejo de archivos de texto y binarios y las clases que permiten esto, lo cual debe ser un conocimiento previo y obligatorio para el estudiante de estructuras de datos.

Ejercicios de autoevaluación del capítulo 10

1. Defina el concepto de archivo de texto y binario.
2. Enumere y explique las clases que permiten el manejo de archivos de texto y binarios en C++.
3. Analice las posibles aplicaciones que se le podrían dar a los archivos de texto y binarios dentro del contexto de las estructuras de datos.
4. Realice ejercicios alternativos, replicando el código facilitado en el libro de texto para el manejo de archivos de texto y binarios.

5. Complete el siguiente crucigrama con las respuestas correctas



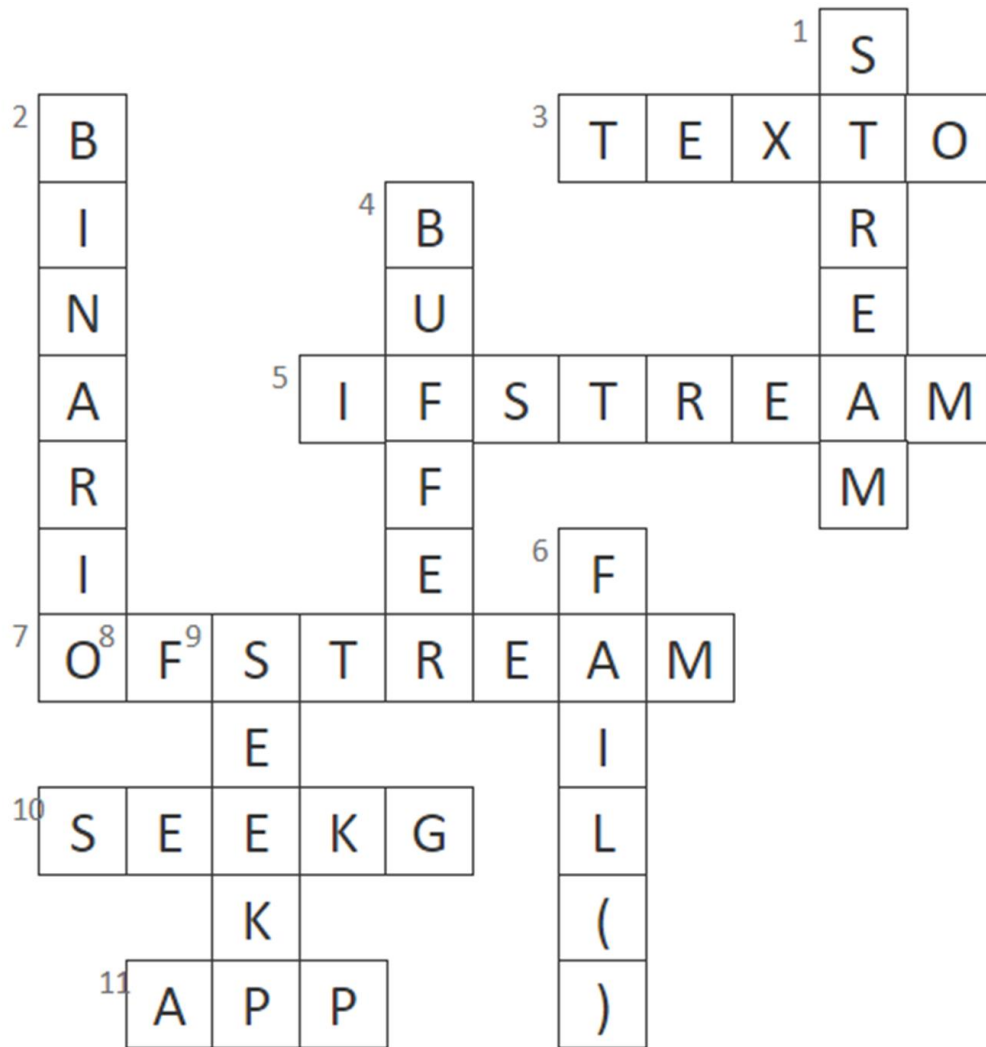
Verticales

1. Flujo de datos en un archivo de texto o binario
2. El contenido del fichero posee una representación en formato de ceros y unos
4. Almacenamiento temporal del flujo de datos para el manejo de archivos de texto o binarios.
6. Flag que puede detectar un error de formato de tipo de dato
9. Establece la posición en la que se insertará el siguiente carácter en el flujo de salida.

Horizontales

3. Los datos contenidos en este tipo de ficheros están representados por caracteres alfanuméricos
5. Clase orientada a la lectura
7. Clase orientada a la escritura
8. Clase orientada a la escritura y/o la lectura
10. Establece la posición del siguiente carácter que se extraerá del flujo de entrada.
11. Flag que permita que un registro se agregue al final del archivo

Respuesta al crucigrama



Tema II. Estructura de datos en Java

Sumario

- **Parte I:** generic collection, expresiones lambda, colas y pilas, listas enlazadas, interfaz MAP.
- **Parte II:** árboles binarios, árboles AVL, grafos, recorridos BFS y DFS, problema de la ruta más corta (algoritmo de Dijkstra)

Objetivos específicos

Al finalizar el estudio de este tema, usted estará en capacidad de:

- Explicar la fundamentación teórica del manejo de estructuras de datos en Java.
- Implementar código Java para el manejo de distintas estructuras de datos.

Introducción

Este tema enfatiza en las diferentes estructuras de datos que se explican en el tema I, tales como pilas, colas, listas enlazadas, árboles y grafos. El tema se basa en los capítulos 1 al 10, en los cuales se desarrolla conceptualmente estas estructuras y se implementan mediante código C++. La principal importancia del desarrollo de las mismas estructuras tratadas en estos capítulos estriba en la diferenciación existente, no conceptualmente dado que es la misma teoría, sino que en el manejo de la sintaxis del lenguaje. Al final, el estudiante podrá determinar estas diferencias y determinar las fortalezas y debilidades de cada lenguaje y, quizás, decantarse por uno u el otro de acuerdo a la experiencia vivida.

Guía de lectura

Para el estudio de este tema, lea detalladamente las páginas del libro que se indican a continuación:

Capítulo	Páginas
Capítulo 11. Fundamentos de estructuras de datos en Java/ Parte I.	251 – 276
Capítulo 12. Fundamentos de estructuras de datos en Java/ Parte I.	285 – 309

Comentarios del tema

Capítulo 11. Fundamentos de estructura de datos en Java/Parte I

11.1. Generics collection Java

Los genéricos permiten que las clases e interfaces sean parámetros cuando se definan. Java lo implementó con el fin que un tipo de dato (cadenas, enteros, objetos personalizados, entre otros) o un método puedan operar en diferentes tipos. También ofrece seguridad al momento de la compilación de un programa, detectando tipos no válidos o que coinciden en la compilación.

Generics collection permite a los programadores habilitar la escritura de código para implementar algoritmos genéricos y trabajar en diferentes colecciones o estructuras de datos. Por ejemplo se puede tener la declaración en Java de un ArrayList y agregarle una cadena. Luego para obtener el valor de la primera posición de ese arreglo habría que hacer un casting de tipo string. La forma sería:

```
List lista = new ArrayList();  
List.Add("Universidad");  
String cadena = (String) list.get(0);
```

Se puede observar que requerimos "castear" el elemento cero del ArrayList para obtener su valor de cadena. En lugar de ello, mediante genericidad se implementaría lo anterior de la siguiente forma:

```
List lista = new ArrayList<String>();  
List.Add("Universidad");  
String cadena = list.get(0);
```

11.2. Expresiones Lambda en Java

Una expresión o función lambda (denominada también función literal o función anónima) es una subrutina definida que no se enlaza a ningún identificador. Pueden utilizarse como argumentos que se pasan a otra función de orden superior o para construir el resultado de una función de orden superior que necesita retornar otra función.

Las expresiones o funciones lambda se utilizan principalmente en la programación funcional, permitiendo crear códigos de programación más concisos y significativos que pueden referenciar métodos anónimos o métodos sin nombre. Por ejemplo la siguiente instrucción es una expresión o función lambda en Java:

```
Function<String,Integer> sizeOf = cadena -> cadena.length();
```

En la expresión o función lambda anterior lo que devuelve es la longitud de una cadena dada, producto de un parámetro string recibido.



El libro de texto desarrolla los aspectos básicos de las expresiones o funciones lambda en las páginas 256 a 258. Se explica someramente su conceptualización y se expone un ejercicio sencillo de implementación.

11.3. Colas y pilas en Java

Las colas y pilas en Java tienen la misma conceptualización en el mecanismo de manejo de cualquier otro lenguaje de programación, por lo que si ya el estudiante domina, conceptual y programáticamente, el manejo de colas y pilas en C++ bastará con conocer un poco la sintaxis de Java.

En el libro de texto se tratan las colas simples y colas de prioridad que son un tipo de cola que dispone de un mecanismo de priorización de salida de los elementos de esta estructura. Es un caso muy común cuando se priorizan trabajos en una cola de impresión, una cola de atención de clientes, una cola de despacho de paquetes por prioridad, entre otros casos.

11.4. Listas enlazadas en Java

Al igual que las colas y pilas estudiadas en la sección dedicada a C++, en este apartado se conceptualiza e implementan listas enlazadas mediante Java. La diferenciación básicamente es el uso de clases especializadas para el manejo de las listas enlazadas como lo es la clase *LinkedList* que implementa una lista doblemente enlazada. También existe la clase *ArrayList* que permite manejar registros en memoria que asemejan la funcionalidad de una lista enlazada. Esta clase, al igual que *LinkedList* posee métodos especializados para el mantenimiento de los registros o nodos de datos que las componen.



En el libro de texto se desarrollan los conceptos de pilas y colas (páginas 258 a 267) y se implementan códigos demostrativos de uso. Acá se utilizan clases *Queue* y *PriorityQueue*, además de la clase *Stack*. En las páginas 267 a 272 también se conceptualizan el uso de listas mediante el uso de las clases *ArrayList* y *LinkedList*

11.5. Interfaz MAP

La interfaz MAP es una estructura de datos especial que maneja elementos a manera de un diccionario, donde convergen dos datos: la llave (*key*) y el valor (*value*). La importancia de esta estructura es que no permite valores duplicados mediante la llave aunque si pueden haber valores duplicados pero con distinta llave.

También se tratan los temas de *HashMap* y *LinkedHashMap*. La estructura *HashMap* trabaja la generación de la llave mediante *hashCode* manejando eficientemente el asunto de las colisiones que se presentan generalmente en las rutinas de hash. *LinkedHashMap*, por su parte, los elementos contenidos se manejan como una lista doblemente enlazada y siempre conservando la lógica de uso de datos estilo llave y valor.

Resumen del capítulo

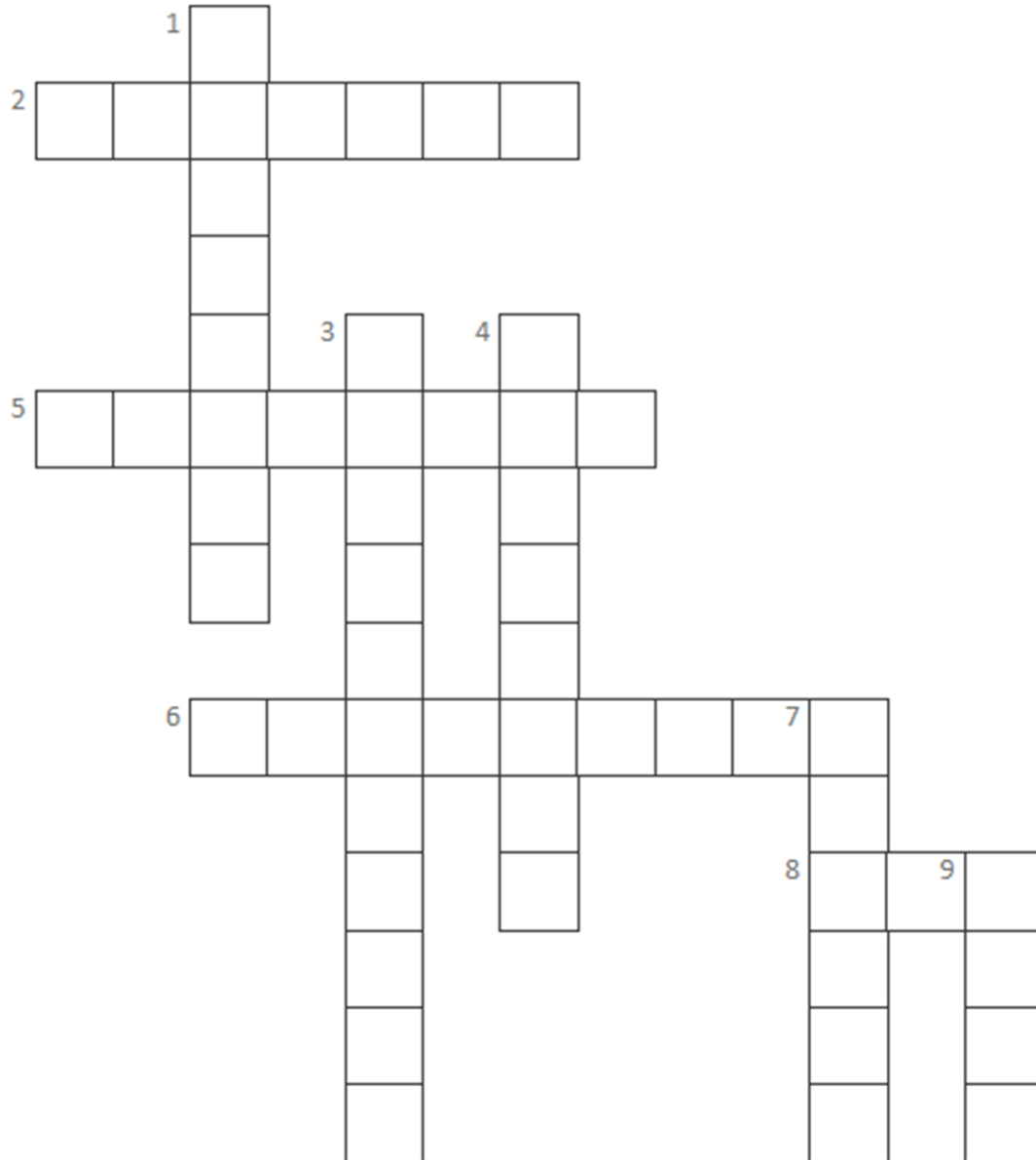
En este capítulo se tratan los temas de colecciones genéricas, expresiones lambda, pilas, colas, listas enlazadas y la interfaz MAP utilizando Java para su implementación. La conceptualización de estas estructuras, a excepción de las colecciones genéricas y la interfaz MAP, fueron desarrolladas en los capítulos anteriores de C++, por lo que no hay mayor desarrollo teórico al respecto.

En cuanto la implementación de estas estructuras, en el libro de texto de

aportan varios ejemplos programados que ayudan al lector a comprender su funcionamiento. Esta implementación no requiere mayor escritura de código por cuanto Java dispone de clases y funciones que ayudan en el manejo o mantenimiento de los datos de estas estructuras.

Ejercicios de autoevaluación del capítulo 11

1. Describa que son colecciones genéricas en Java.
2. Defina que son expresiones o funciones lambda y cree algunos ejemplos de uso.
3. Explique el funcionamiento de la interfaz MAP y todas sus variaciones (*HashMap* y *LinkedHashMap*). Cree ejemplos demostrativos de uso.
4. Desarrolle ejemplos de uso de las estructuras pilas, colas y listas enlazadas en Java.
5. Complete el siguiente crucigrama con las respuestas correctas



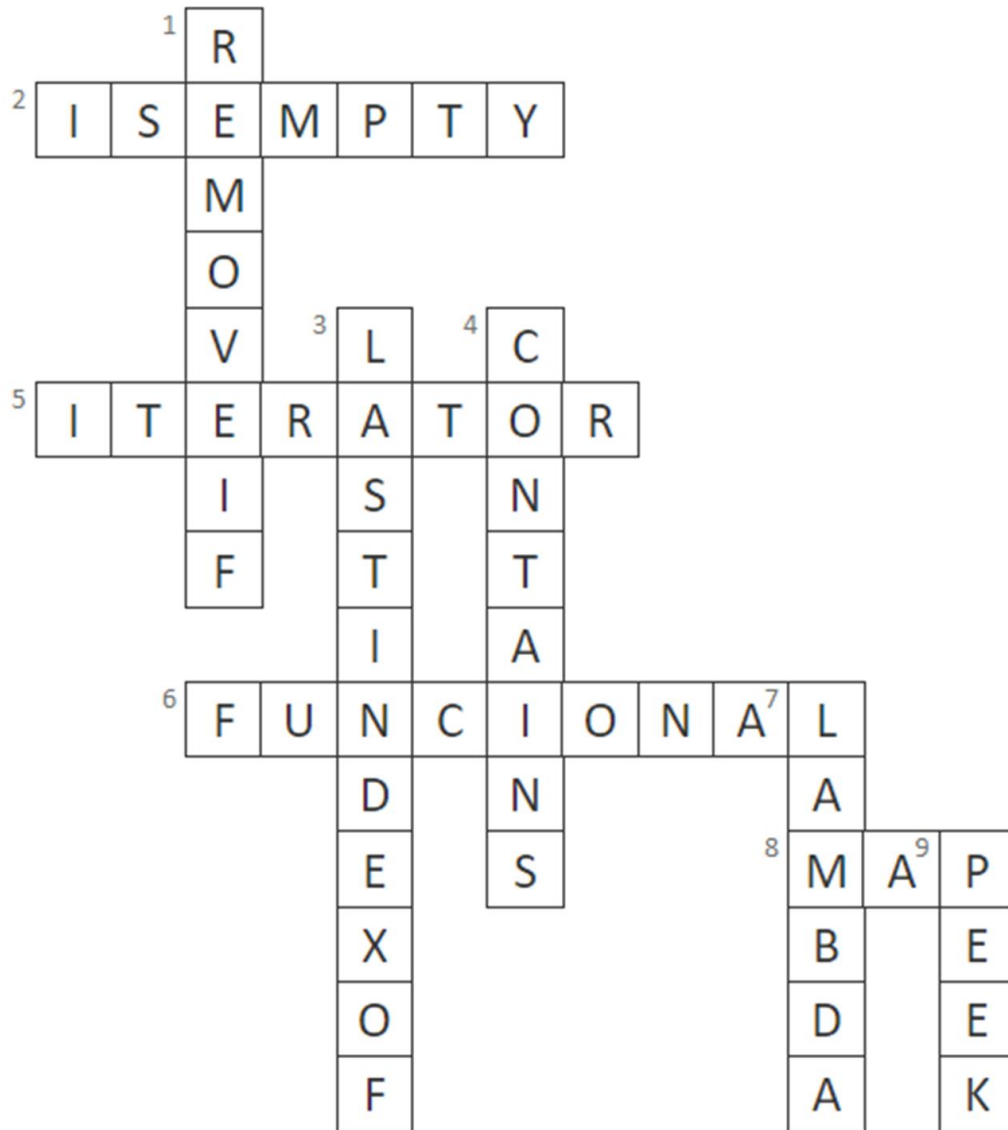
Verticales

1. Elimina todos los elementos de la colección que cumplan con algún criterio.
3. Devuelve la posición de la última aparición de los caracteres especificados en una cadena o de un objeto.
4. Retorna verdadero si el elemento buscado es encontrado en la colección
7. Método sin declaración, es decir, sin modificadores de acceso, que devuelve un valor y un nombre
9. Permite la salida de elementos de una cola

Horizontales

2. Retorna verdadero si la colección referenciada no contiene elementos.
5. Retorna un iterador sobre el elemento referenciado en la colección
6. Tipo de paradigma programático de la que derivan las expresiones Lambda
8. No deriva de la interfaz Collection

Respuesta al crucigrama



Capítulo 12. Fundamentos de estructura de datos / Parte II

12.1. Árboles binarios en Java

El capítulo 7 del libro de texto trata sobre la conceptualización de los árboles binarios. Este capítulo 12 hace una pequeña introducción a este concepto y se enfoca más a la implementación de código mediante el lenguaje de programación Java. Los listados 12.1, 12.2 y 12.3 de implementación de este código se pueden descargar del sitio web de la editorial. Se podrá determinar que la implementación de esta estructura en Java se diferencia de C++ en la sintaxis, por lo que es importante que el estudiante analice este código.

12.2. Árboles AVL en Java

Al igual que los árboles binarios, la conceptualización de árboles AVL o árboles balanceados ya fueron desarrollados en capítulos anteriores por lo que, el esfuerzo de estudio radica en el análisis del código y la diferenciación de la sintaxis entre Java y C++. La implementación del código es la base de estudio propiamente de esta parte del libro. Sin embargo, el libro de texto repite algunos mecanismos ya explicados anteriormente como lo es el balanceo de nodos en un árbol AVL y los métodos de inserción y borrado.



En el libro de texto se desarrollan los conceptos árboles binarios de búsqueda (ABB) y balanceados (AVL) en las páginas 285 a 296. Tanto la conceptualización como los mecanismos de implementación de código se desarrollan en este rango de páginas del libro de texto.

12.3. Grafos en Java

La conceptualización y los mecanismos de uso de los grafos ya fueron discutidos en el capítulo 8 del libro de texto, por lo que este capítulo se enfoca en la implementación de código utilizando el lenguaje de programación Java.

12.4. Recorrido de grafos (BFS y DFS)

Los mecanismos de recorrido de grafos en profundidad (DFS) y en amplitud (BFS) ya fueron explicados en los capítulos anteriores de este libro de texto. En este capítulo lo que enfoca es cómo se implementa utilizando el lenguaje de programación Java, por lo que la comprensión se orienta hacia el código fuente generado en esta implementación.

12.5. Problema de la ruta más corta (algoritmo de Dijkstra)

El algoritmo de Dijkstra se utiliza para encontrar el camino más corto entre un nodo origen y uno destino, pasando a través de sus aristas. Estos nodos o vértices y sus arcos o aristas pueden representar carreteras entre ciudades, conectividad de red entre nodos de computadoras, tareas de un diagrama Pert entre otras.



En el libro de texto se desarrollan los mecanismos de recorrido en profundidad (DFS: Depth-First-Search) y en amplitud o anchura (BFS: Breadth-First-Search) en las páginas 296 a 305. Asimismo, el algoritmo de la ruta más corta de Dijkstra se explica en las páginas 305 a 309. Ambos temas desarrollan un resumen del concepto e implementan código fuente demostrativo.

Resumen del capítulo

Este capítulo se enfoca en el estudio de estructuras ya conocidas de capítulos anteriores: árboles binarios de búsqueda (ABB), árboles binarios balanceados (AVL), grafos, recorridos de grafos y el problema de la ruta más corta mediante el uso del algoritmo de Dijkstra. El enfoque de este capítulo es demostrar como es la implementación del código que maneja estas estructuras mediante el lenguaje Java.

Ejercicios de autoevaluación del capítulo 5

1. Repase la conceptualización de árboles binarios de búsqueda (ABB), balanceados (AVL).
2. Repase la conceptualización de grafos y sus estrategias de recorrido (BFS y DFS).
3. Repase la conceptualización del problema de búsqueda de la ruta más corta mediante el uso del algoritmo de Dijkstra.
4. Analice el código de implementación de árboles binarios de búsqueda (ABB) y árboles binarios balanceados (AVL) mediante el lenguaje Java.
5. Analice el código de implementación del mantenimiento de grafos y sus recorridos BFS y DFS mediante el lenguaje Java.

Guía de lectura

Para el estudio de este tema, lea detalladamente las páginas del libro que se indican a continuación:

Capítulo	Páginas
Capítulo 13. Fundamentos de estructuras de datos en Python/ Parte I.	315 – 343
Capítulo 14. Fundamentos de estructuras de datos en Python/ Parte I.	349 – 363

Comentarios del tema

Capítulo 13. Fundamentos de estructura de datos en Python/Parte I

13.1. Conceptos básicos de Python.

El uso de conceptos básicos de Python abarca la comprensión de como se crean variables, constantes, uso de instrucciones de decisión (if o switch), ciclos (while, for, otros) en otros lenguajes y su representación en este nuevo lenguaje. Por tanto, a este nivel de programación, el estudiante de estructuras de datos no debería tener mayor problema en cómo implementar la sintaxis básica de Python en los programas que se crean en este apartado del libro de texto.

13.2. Sintaxis en Python

Python posee dos tipos de instrucciones, tal como lo indica el libro de texto: simples y compuestas. Las simples son aquellas en las que Python compila una sola línea de código. Las compuestas son aquellas que se indentan de acuerdo con el nivel de profundidad de la instrucción. Por ejemplo, un if puede al final tener dos puntos (:) y luego indentarse lo que se desea sea parte de las instrucciones que se cumplen si el criterio de ese if es verdadero. La indentación es la que caracteriza la subordinación de código en Python.

13.3. Bifurcaciones y ciclos en Python

Las bifurcaciones y ciclos en Python trabajan conceptualmente como cualquier otro lenguaje. Podemos encontrar el `if`, `for`, `do while`, como en otros lenguajes, con la diferencia que otros los separan con un `begin...end`, con llaves, entre otros determinantes. Sin embargo para Python, la característica es que no existen estos determinantes, salvo los dos puntos (`:`) que denotan subordinación, seguido de la correspondiente indentación. Los listados 13.1. y 13.2. del libro de texto, muestran el uso de bifurcaciones y ciclos en Python.

13.4. Funciones en Python

Las funciones en Python cumplen el mismo objetivo que en cualquier otro lenguaje de programación. La diferencia en Python es que las funciones se definen anteponiendo al nombre de la función la palabra reservada `def`. Acá también reviste gran importancia el uso de los dos puntos (`:`) que denotan subordinación del código subsiguiente y la indentación. El listado 13.3. del libro de texto muestra el uso de funciones en Python.

13.5. Recursividad en Python

La recursividad como concepto se maneja de igual manera que en otros lenguajes de programación. Es una lógica en la cual una función se llama a sí misma tantas veces como sea necesario para generar el resultado esperado. Por tanto, acá entra en juego el mecanismo de funcionamiento de funciones indicado en el punto 13.4. Los listados 13.4., 13.5. y 13.6. del libro de texto muestran código que utiliza recursividad.

13.6. Arreglos en Python

El concepto de arreglo ya fueron discutidos en capítulos anteriores del libro de texto. Los mecanismos de manejo de un arreglo en Python se llevan a cabo mediante el

uso de la clase *array*, lo cual facilita la escritura de código sobre estas estructuras. Se pueden realizar inserciones dentro del arreglo solamente utilizando la instrucción *insert*, indicándole en qué posición deseamos hacer dicha inserción. El listado 13.7 del libro de texto demuestra claramente el uso de arreglos en Python.

En Python también es posible realizar operaciones sobre matrices utilizando la librería *numpy*. Con esta librería se pueden realizar agregaciones con *append* o *insert*, cambiar el signo de los elementos de un arreglo con *negative*, raíz cuadra con *sqrt*, entre otras funcionalidades. El listado 13.8. muestra el uso de la librería *numpy* sobre una matriz de elementos numéricos.

13.7. Colas y pilas en Python

Las colas y pilas en Python tienen un mecanismo de manipulación similar que en cualquier otro lenguaje de programación. En el caso de las pilas se utilizan palabras reservadas que ayudan al manejo de éstas: *stack* (para crear una pila), *push(elemento)* para agregar un elemento en la pila, *pop()* (para eliminar el elemento superior de la pila), *top()* (que devuelve el elemento en el tope de la pila), *isEmpty()* o *size()* para determinar si está vacía la pila o el tamaño de la pila, respectivamente. En cuanto las colas, operaciones como *Queue* (crear una cola vacía), *enqueue(elemento)* agrega un elemento al final de la cola, *dequeue()* elimina el elemento al frente de la cola, entre otras. Los listados 13.9. y 13.10. muestran la implementación de pilas y colas en Python.

13.8. Listas en Python.

Las listas en Python son arreglos que permiten almacenar conjuntos de datos de diferente tipo. Son dinámicas y cuentan con una serie de métodos que permiten el manejo eficiente de las operaciones normales sobre un arreglo. Se pueden insertar elementos al final de la lista o en una determinada posición de ésta, se pueden remover sin mucha dificultad, entre otras funcionalidades. El listado 13.11. muestra el código requerido para implementar una lista en Python.



En el libro de texto se desarrollan el mantenimiento de datos en listas simplemente enlazada con Python en las páginas 332 a 336. De la página 336 a 337 el manejo de listas doblemente enlazadas y, finalmente en las páginas 337 a 343 sobre listas circulares simple y doblemente circulares en Python. Es importante analizar este código y recordar el manejo conceptual que se ha venido explicando a través de todo el libro de texto.

Resumen del capítulo

El capítulo 13 del libro de texto es una breve introducción al lenguaje de programación Python. La finalidad es introducir al estudiante en los conceptos básicos de este lenguaje de programación con la idea de ir fundamentando la implementación de estructuras de datos que ya fueron desarrolladas en capítulos anteriores del libro.

Este capítulo también enfoca el uso de conceptos básicos de Python, el uso de arreglos, pilas, colas y listas. Todo utilizando clases con métodos especializados en el mantenimiento de los datos contenidos en estas estructuras. Por ende, si la lógica es la misma que en otros lenguajes de programación, solamente hay que invertir tiempo en el aprendizaje de la sintaxis de este lenguaje de programación.



Ejercicios de autoevaluación del capítulo 13

1. Defina con sus propias palabras los conceptos de arreglo, pila, cola y lista, como estructura de datos.
2. Enumere los métodos provistos en las pilas, colas y listas que provee Python.
3. Realice ejercicios alternativos, replicando el código facilitado en el libro de texto para el manejo de arreglos, pilas, colas y listas en Python.

Capítulo 14. Fundamentos de estructura de datos en Python/Parte II

14.1. Árboles en Python.

Tal como indica el libro de texto, los arboles binarios y AVL pueden ser implementados a través de listas indexadas por punteros o mediante vectores indexados por vértices o nodos. En cuanto los árboles en Python su implementación no representa gran complejidad y el ejemplo del listado 14.1. así lo comprueba. La imagen de ejecución de este listado (figura 14.1) muestra la conformación del árbol. Para el árbol binario de búsqueda (ABB) el listado 14.2. implementa la funcionalidad requerida y que ya, a este nivel, el estudiante debería estar familiarizado.

Para el árbol binario balanceado (AVL) el listado 14.3. implementa la funcionalidad, enfatizando en el factor de balanceado requerido por este tipo de estructura. Las tablas 14.1. y 14.2. muestran los procedimientos requeridos para que el árbol binario se encuentre balanceado en todo momento, luego de realizar inserciones o eliminaciones. Asimismo, la tabla 14.3. muestra los tipos de recorridos del árbol.

14.2. Grafos

Los grafos fueron desarrollados en el capítulo 8 de este libro. Por tanto, la conceptualización de su funcionalidad está más que discutida. Para este capítulo 14, el listado 14.4. muestra la búsqueda más corta entre dos nodos de un grafo. La conceptualización de grafos se desarrolló en capítulos anteriores de este libro, por tanto no se entra en mayores detalles y se enfoca en la implementación de código mediante sintaxis Python.

14.3. Recorridos DFS y BFS.

Las búsquedas en amplitud (BFS: *Breadth First Search*) y profundidad (DFS, *Depth First Search*) explicadas conceptualmente en capítulos anteriores de este libro, se enfatizan su funcionalidad mediante código Python. El listado 14.6. del libro de texto implementa la lógica funcional de estos dos tipos de búsqueda utilizando sintaxis Python. Las figuras 14.10. y 14.11. muestran estos recorridos a través del grafo.

Resumen del capítulo

El capítulo 14 del libro de texto es la continuación básica del uso de Python. La finalidad es implementar estructuras de datos tales como árboles binarios, de búsqueda o ABB y balanceados AVL, y grafos. La implementación de estas estructuras, utilizando Python, demuestran la facilidad y claridad del código de este lenguaje de programación.

Los códigos escritos para implementar la funcionalidad de árboles, grafos y los recorridos en amplitud y profundidad vienen a demostrar la utilidad de estas estructuras de datos para la solución de muchos problemas computacionales.

Ejercicios de autoevaluación del capítulo 14

1. Defina con sus propias palabras los conceptos de árbol binario, árbol binario de búsqueda (ABB), árbol binario balanceado (AVL), grafo y recorridos DFS y BFS.
2. Utilice el sitio web <https://visualgo.net/en> para reforzar los conocimientos conceptuales y de implementación de las estructuras de datos desarrollados en este libro de texto.
3. Realice ejercicios alternativos, replicando el código facilitado en el libro de texto para el manejo de árboles, grafos y recorridos en amplitud y profundidad en Python.