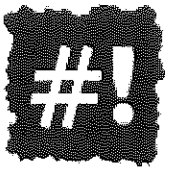


Preparador Informática

www.preparadorinformatica.com

TEMA 12. INFORMÁTICA / S.A.I.

ORGANIZACIÓN LÓGICA DE LOS DATOS. ESTRUCTURAS DINÁMICAS.

TEMA 12 INF / SAI: ORGANIZACIÓN LÓGICA DE LOS DATOS. ESTRUCTURAS DINÁMICAS

1. INTRODUCCIÓN

En el estudio de la informática es fundamental el conocimiento de las estructuras de datos, ya que éstas se consideran la base de todo el proceso de información. Para procesar la información es necesario hacer una abstracción de los datos que tomamos del mundo real, estructurándolos de una determinada forma y formato (organización lógica de los datos) para obtener un rendimiento adecuado en su tratamiento.

Por tanto, la eficiencia y velocidad de ejecución de un programa en un ordenador además de depender del tipo de ordenador en el que se ejecute, de los algoritmos empleados y del compilador, dependerá en gran medida de la organización lógica de los datos.

Actualmente existen sistemas de análisis de datos y big data que trabajan no solamente con datos estructurados sino también con datos no estructurados. Esto ha supuesto un desafío para poder organizar y homogeneizar datos de diferentes fuentes como ayuda a una toma de decisiones más eficiente por parte de las organizaciones.

El presente tema está dedicado a estudiar la organización lógica de los datos, y concretamente los tipos y características de las estructuras dinámicas.

2. ORGANIZACIÓN LÓGICA DE LOS DATOS

2.1. CONCEPTOS

2.1.1. DATO

Se denomina dato a cualquier objeto manipulable por el ordenador. Es decir, un dato puede ser un carácter leído de un teclado, información almacenada en un disco, un número que se encuentra en memoria principal, etc.

2.1.2. ESTRUCTURAS DE DATOS

Una estructura de datos es una forma de organizar un conjunto de datos elementales con objeto de facilitar la manipulación de estos datos.

Las estructuras de datos se caracterizan por:

- La eficiencia en el uso de la memoria
- La velocidad de acceso a los datos

2.2. TIPOS

2.2.1. TIPOS DE DATOS

Podemos clasificar los tipos de datos que se emplean en los lenguajes de programación en dos tipos:

- **Datos simples**: tienen como característica común que cada variable representa a un elemento. Son aquellos cuyos valores son atómicos y, por tanto, no pueden ser descompuestos en valores simples. Ejemplos: entero, real, carácter, Etc.
- **Datos compuestos**: están construidos basándose en tipos de datos simples. Ejemplos: matriz, cadena de caracteres, Etc.

2.2.2. TIPOS DE ESTRUCTURAS DE DATOS

Los tipos de datos se pueden organizar en diferentes estructuras atendiendo a la forma en que ocupan el espacio en memoria:

- **Estáticas**: El tamaño ocupado en la memoria se define antes de que el programa se ejecute y no puede modificarse durante la ejecución. Ejemplos: arrays.
- **Dinámicas**: El tamaño ocupado en memoria puede cambiar en tiempo de ejecución. Ejemplos: listas (enlazadas, pilas, colas), árboles y grafos.

3. ESTRUCTURAS DINÁMICAS

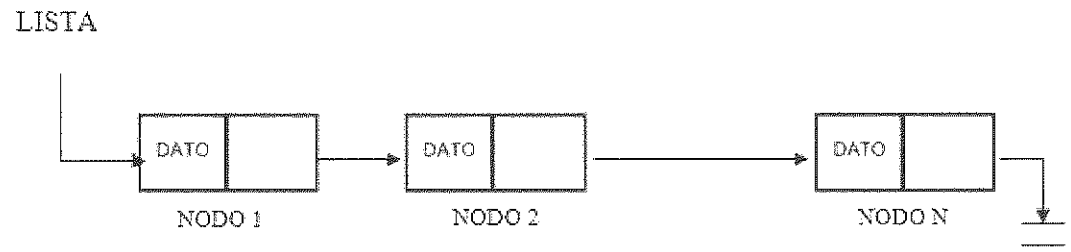
El estudio de las estructuras dinámicas se divide en:

- Estructuras de datos lineales:
  - Listas
  - Pilas
  - Colas
- Estructuras de datos no lineales:
  - Árboles
  - Grafos

3.1. ESTRUCTURAS DE DATOS LINEALES

3.1.1. LISTAS ENLAZADAS

Una lista enlazada es una secuencia de nodos donde cada nodo almacena sus propios datos y un puntero (dirección) a la ubicación del siguiente nodo. De modo que una lista enlazada es una sucesión de nodos o elementos entre los que existe una relación lineal. El último elemento de la lista tiene un enlace a NULL, indicando el final de la lista.



Además de la lista enlazada sencilla existen otros tipos de listas:

- ***Listas circulares***: el último nodo enlaza con el primero.
- ***Listas doblemente enlazadas***: cada nodo tiene un doble enlace para apuntar al nodo siguiente y anterior.
- ***Multilistas***: cada uno de los nodos de la lista puede apuntar a otras listas.

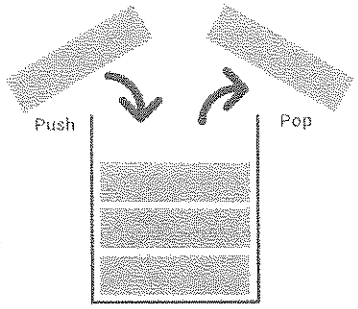
Ejemplo: Implementación de una lista enlazada en Java junto con sus principales operaciones:

```
public class ListaEnlazada {

	static class Nodo {
		int valor;
		Nodo sig;

		public Nodo(int valor) {
			this.valor = valor;
		}
	}

	private Nodo primero = null;

	//Inserta un elemento al principio de la lista
	public void insertarAlPrincipio(Nodo nodo) {
		nodo.sig = primero;
		primero = nodo;
	}

	//Inserta un elemento al final de la lista
	public void insertarAlFinal(Nodo nodo) {
		if (primero == null)
			primero = nodo;
		else {
			Nodo ptr = primero;
			while(ptr.sig != null)
				ptr = ptr.sig;
			ptr.sig = nodo;
		}
	}

	//Elimina el primer elemento de la lista
	public void eliminarPrimero() {
		primero = primero.sig;
	}

	//Muestra los elementos de la lista
	public void mostrarLista() {
		Nodo ptr = primero;
		while(ptr != null) {
			System.out.print(ptr.valor+" -> ");
			ptr = ptr.sig;
		}
	}

	public static void main(String[] args) {
		ListaEnlazada L = new ListaEnlazada();
		L.insertarAlPrincipio(new Nodo(1));
		L.insertarAlPrincipio(new Nodo(8));
		L.insertarAlFinal(new Nodo(5));
		L.mostrarLista();
	}
}
```

3.1.2. PILAS

Una pila es una sucesión de nodos o elementos en la que las inserciones y eliminaciones se producen siempre por un mismo extremo llamado cima.

De esta forma, siempre que se agrega un elemento a la pila se sitúa en la cima de la misma y solo puede eliminarse el elemento situado en la cima. Por esta razón se trata de una estructura LIFO (Last In, Primero Out).



Operaciones

- **Push**: añadir un nuevo elemento a la cima de la pila
- **Pop**: eliminar el elemento situado en la cima de la pila

Aplicaciones

Las pilas se pueden usar para crear las funcionalidades de deshacer y rehacer, como analizadores de expresiones, etc.

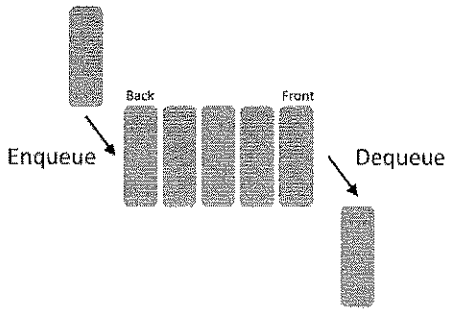
Ejemplo: Implementación de una pila en Java junto con sus principales operaciones:

```
public class Pila {

	static class Nodo {
		int valor;
		Nodo sig;

		public Nodo(int valor) {
			this.valor = valor;
		}
	}

	private Nodo primero = null;

	//Añade un nuevo elemento a la cima
```

```
	public void push(Nodo nodo) {
		nodo.sig = primero;
		primero = nodo;
	}

	//Elimina el elemento situado en la cima
	public void pop() {
		primero = primero.sig;
	}

	public void mostrarPila() {
		Nodo nodo = primero;
		while(nodo != null) {
			System.out.println("|"+nodo.valor+"|");
			nodo = nodo.sig;
		}
	}

	public static void main(String[] args) {
		Pila s = new Pila();

		s.push(new Nodo(1));
		s.push(new Nodo(2));
		s.push(new Nodo(4));
		s.push(new Nodo(8));

		s.mostrarPila();
	}
}
```

3.1.3. COLAS

Una cola es una sucesión de nodos o elementos en la que sólo pueden añadirse elementos por un extremo (fin) y sólo pueden retirarse por el otro (cabeza). Es una estructura de tipo FIFO (First In First Out), ya que el primer elemento en entrar es también el primero en salir.



Operaciones

- **Enqueue** (encolar): añade un nuevo elemento a la cola
- **Dequeue** (desencolar): elimina un elemento de la cola

Aplicaciones

Las colas, entre otras aplicaciones, se usan en sistemas operativos para la gestión de trabajos no interactivos (procesamiento por lotes o "batch").

Ejemplo: Implementación de una cola en Java junto con sus principales operaciones:

```
public class Cola {

	static class Nodo {
		int valor;
		Nodo sig;

		public Nodo(int valor) {
			this.valor = valor;
		}
	}

	private Nodo primero = null;
	private Nodo ultimo = null;

	//Añade un elemento al final de la cola
	public void enqueue(Nodo nodo) {
		if(ultimo == null)
			primero = ultimo = nodo;
		else {
			ultimo.sig = nodo;
			ultimo = nodo;
		}
	}

	//Elimina el elemento situado al comienzo de la cola
	public void dequeue() {
		primero = primero.sig;
		if(primero == null)
			ultimo = null;
	}

	public void mostrarCola() {
		Nodo nodo = primero;
		while(nodo != null) {
			System.out.mostrarCola(nodo.valor+" ");
			nodo = nodo.sig;
		}
	}

	public static void main(String[] args) {
		Cola q = new Cola();

		q.enqueue(new Nodo(1));
		q.enqueue(new Nodo(2));
		q.enqueue(new Nodo(8));
		q.mostrarCola();

	}
}
```

3.2. ESTRUCTURAS DE DATOS NO LINEALES

3.2.1. ÁRBOLES

Un árbol es una estructura de datos jerárquica formada por una serie de nodos conectados por una serie de aristas que verifican que:

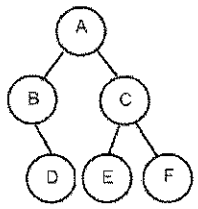
- Hay un único nodo raíz.
- Cada nodo, excepto la raíz, tiene un único nodo padre.
- Hay un único camino desde la raíz hasta cada nodo.

A. TERMINOLOGÍA

- **Nodo padre y nodo hijo**: Existe una relación entre nodos en la que el nodo que está por encima se llama padre y el que está por debajo hijo.
- **Nodo hoja**: es aquel que no tiene nodos hijos.
- **Profundidad** de un árbol: es el número de relaciones padre que hay que seguir para llegar desde su descendiente hoja más lejano hasta la raíz.
- **Altura** de un árbol: es el número de relaciones hijo que desde la raíz hay que seguir hasta alcanzar su descendiente hoja más lejano.
- **Nivel**: conjunto de nodos que se encuentran a la misma profundidad.
- **Subárbol**: árbol formado por un nodo y todos sus descendientes.
- **Grado de un nodo**: es el número de nodos hijos que tiene.
- **Grado de un árbol**: es el mayor grado de todos sus nodos.

B. RECORRIDOS

Recorrer un árbol es el proceso por el cual se accede a todos los nodos.



Los recorridos de un árbol se pueden clasificar en:

- **Recorridos en profundidad**: pueden ser de 3 tipos:
  - *Recorrido en preorden*: Visita primero la raíz, después el subárbol izquierdo y por último el subárbol derecho. (A, B, D, C, E, F)

- *Recorrido en inorden*: Visita primero el subárbol izquierdo, después la raíz, y por último el subárbol derecho.(B, D, A, E, C, F)
- *Recorrido en postorden*: Visita primero el subárbol izquierdo, después el subárbol derecho, y por último la raíz. (D, B, E, F, C, A)

– **Recorrido en amplitud**: Consiste en ir visitando el árbol por niveles. Primero se visita el nodo del nivel 1, después los nodos del nivel 2, y así sucesivamente. (A, B, C, D, E, F)

C. OTRAS OPERACIONES SOBRE ÁRBOLES

- Crear árbol
- Insertar/eliminar nodo
- Buscar nodo

D. TIPOS DE ÁRBOLES

En función del grado que tenga un árbol se pueden clasificar en **árboles binarios** y **árboles multicamino**.

1. Árboles binarios

Un árbol binario es un árbol de grado 2. Es decir, cada nodo tiene como máximo dos nodos hijos (llamados izquierdo y derecho).

Los principales tipos de árboles binarios son:

a) Árboles binarios de búsqueda (ABB)

Un árbol binario de búsqueda es aquel en que la clave de cada nodo es mayor que todas las claves de su subárbol izquierdo, y menor que todas las claves de su subárbol derecho.

b) Árboles AVL

Un árbol AVL (Adelson-Velskii y Landis) es un árbol binario de búsqueda bien equilibrado. Un árbol está bien equilibrado cuando para cada nodo la altura de su subárbol izquierdo y su subárbol derecho difiere a lo sumo en una unidad.

c) Árboles Rojo-Negro

Un árbol rojo-negro es un ABB que verifica las siguientes propiedades orden:

- Los nodos tienen un atributo de color (rojo o negro).
- La raíz es siempre negra.
- Si un nodo es rojo los nodos hijos son negros.
- Cada camino desde la raíz hasta las hojas tiene el mismo número de nodos negros

2. Árboles multicamino

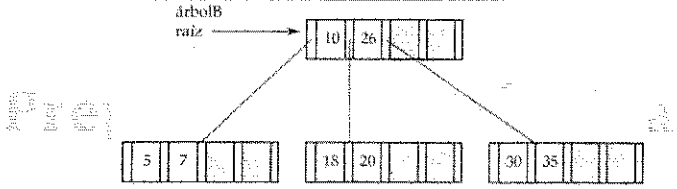
Un árbol multicamino es un árbol de grado mayor que 2.

Los principales tipos de árboles multicamino son:

a) Árboles B

Un árbol-B de orden M es un árbol que satisface las siguientes propiedades:

- Cada nodo tiene como máximo M hijos.
- Cada nodo (excepto raíz) tiene como mínimo (M)/2 claves.
- Todos los nodos hoja aparecen al mismo nivel.
- Un nodo no hoja con k hijos contiene k-1 claves.



b) Árboles B+

Los árboles B+ son aquellos en los que todas las claves están en los nodos hoja mientras que los nodos interiores contienen los índices para las claves. Cada nodo hoja tiene un apuntador adicional para apuntar al siguiente nodo hoja, de modo que están conectados secuencialmente.

3.2.2. GRAFOS

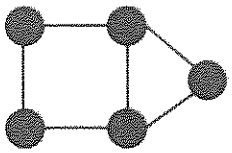
Un grafo es un conjunto de nodos conectados donde cada nodo se denomina vértice y la conexión entre dos de ellos se denomina arista o arco.

Los grafos se utilizan para representar muchas aplicaciones de la vida cotidiana, como redes, rutas de transporte y redes sociales como Facebook, donde cada

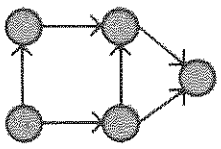
persona representa un nodo. De igual modo, los grafos son utilizados en muchas aplicaciones de la informática también.

Se pueden distinguir dos tipos de grafos, **no dirigidos** y **dirigidos**:

- **Grafo no dirigido**: las aristas del grafo son bidireccionales.
  Ejemplo: En Facebook, si un usuario A es amigo de un usuario B, entonces significa que B es amigo de A.



- **Grafo dirigido**: las aristas del grafo son unidireccionales (indican su dirección con una flecha).
  Ejemplo: En Twitter, si un usuario A sigue al usuario B, no necesariamente el usuario B sigue al usuario A.



Cuando el grafo tiene un número fijo de nodos se puede implementar como una **matriz de adyacencia**. Sin embargo, lo más habitual es que el número de nodos del grafo pueda variar por lo que se implementa el grafo mediante **listas de adyacencia** (estructura dinámica).

4. RECURSOS Y HERRAMIENTAS EDUCATIVAS DE INTERÉS

Como herramienta educativa de interés relacionada con este tema podemos citar a **Sololearn**, que se trata de una aplicación web para iOS y Android con una amplia comunidad de usuarios para el aprendizaje de lenguajes de programación, estructuras de datos y fundamentos de algoritmia. Fue galardonada por la FbStart (Facebook) como mejor aplicación del año 2017.

Por otra parte, también podemos citar el sitio web https://visualgo.net/es. Se trata de una web que permite mediante animaciones interactivas ver el

funcionamiento de las estructuras de datos dinámicas y como realizan sus operaciones paso a paso. Es un proyecto dirigido por el Dr. Steven Halim, profesor de la Universidad Nacional de Singapur.

5. CONCLUSIÓN

La elección de la estructura de datos idónea dependerá de la naturaleza del problema a resolver y del lenguaje de programación a utilizar. Por ejemplo, los grafos se usan para representar conexiones en redes sociales (como Twitter, Facebook), los árboles se usan para representar datos jerárquicos, como archivos y directorios en un sistema de archivos, etc. En general, cuando la cantidad de datos a manejar sea grande o se desconozca el tamaño a priori, resultará beneficioso utilizar estructuras dinámicas, ya que presentan la ventaja de ser muy flexibles permitiendo reservar y liberar espacio según se va necesitando. Elegir la estructura de datos correcta es clave para diseñar un algoritmo eficiente.

6. BIBLIOGRAFÍA

- Joyanes Aguilar, L. **Fundamentos de programación. Algoritmos, estructuras de datos y objetos**. Editorial McGraw-Hill
- Prieto A., Lloris A. y Torres J.C. **Introducción a la informática**. Editorial McGraw-Hill
- De Miguel Anasagasti, P. Fundamentos de los computadores. Editorial Paraninfo
- Langsam, Augenstein y Tenenbaum: **Estructuras de datos en C y C++.** Editorial Prentice Hall.
- www.xataka.com (Web sobre actualidad TIC e informática)
- www.sololearn.com
- https://visualgo.net/es