

Organización Lógica de los  
Datos. Estructuras  
Dinámicas.

## TEMA 12

---

ABACUS NT

Oposiciones 2021

## Índice

---

- 1. Introducción**
- 2. Organización Lógica de los Datos**
  - 2.1. Descripción de datos**
  - 2.2. Tipos de datos**
    - 2.2.1. Tipos simples y compuestos
- 3. Estructuras Dinámicas**
  - 3.1. Listas**
    - 3.1.1. Definición
    - 3.1.2. Declaración
    - 3.1.3. Estructura
    - 3.1.4. Operaciones
  - 3.2. Árboles**
    - 3.2.1. Definición
    - 3.2.2. Estructura
    - 3.2.3. Tipos de árboles
    - 3.2.4. Almacenamiento en memoria
    - 3.2.5. Operaciones
  - 3.3. Grafos**
    - 3.3.1. Definición
    - 3.3.2. Tipos de grafos
    - 3.3.3. Representación de los grafos
    - 3.3.4. Operaciones
- 4. Conclusiones**
  - 4.1. Relación con el currículo**
- 5. Bibliografía**

## 1. Introducción

La información en la sociedad actual es un aspecto relevante en múltiples ramas del conocimiento; incluyendo la biología genética, la física cuántica, la divulgación científica y por supuesto la informática, Internet y el tratamiento de la información.

Cada día, se intercambian unos 2,5 billones de bytes de datos. Se estima que el 90% de los datos existentes en el mundo actual se han generado en el curso de los últimos dos años. Este diluvio se conoce como **Big Data**.

Sin embargo, este tremendo gigante en el que se apoya la sociedad moderna, requiere de estructuras de datos y programas que los manipulen de forma eficiente.

Según el autor **Y. Daniel Liang**, en su libro **Introduction to Java Programming and Data Structures, Comprehensive Version (2017)**, “*elegir la estructura de datos más adecuada es, junto con el diseño del algoritmo, la clave para desarrollar software con un buen rendimiento*”.

## 2. Organización Lógica de los Datos

### 2.1. Descripción de datos

Un dato puede identificarse de forma directa dando su valor, o bien asignándole un nombre con el que vamos a reconocer ese dato.

Hay que ver también si ese valor va a mantener fijo su valor a lo largo del problema o bien se va a mantener constante durante todo el proceso, podemos por tanto encontrar datos literales y asociados a un identificador.

- **Literales:** Un literal es la expresión de un valor de un tipo de dato. Por ejemplo, son literales “nombre”, 5, ‘a’. Son, por tanto, valores fijos.
- **Variables:** Una variable representa un conjunto de **posiciones de memoria** que tienen asociado un nombre (**identificador**) y un **valor** (contenido) de un determinado tipo. Para utilizar una variable normalmente hay que declararla primero, asignándole un nombre y un tipo de dato.
- **Constantes:** Una constante es un dato cuyo valor se declara al principio del programa y no variará a lo largo del algoritmo. Igual que las variables se asocia un nombre al valor de la constante.

### 2.2. Tipos de datos

#### 2.2.1. Tipos simples y compuestos

Los tipos de datos utilizados en los diferentes lenguajes de programación se pueden clasificar en **tipos simples y en tipos compuestos**.

Los tipos de datos simples o **primitivos no están compuestos de otras estructuras** de datos. Los usados más frecuentemente en casi todos los lenguajes son: **enteros, reales o**

carácter. Los datos simples tienen como característica común que cada variable representa a un elemento.

Los tipos de datos **compuestos o estructurados** de datos están construidos **basándose en tipos de datos simples**. El ejemplo más representativo es la **cadena** de caracteres. Los datos estructurados tienen como característica común que **un identificador puede representar a múltiples datos individuales**, pudiendo estos ser referenciados individualmente. Las estructuras de datos se pueden dividir según su modo de almacenamiento en:

- **Internas**: se almacenan en memoria interna, estas a su vez pueden dividirse en.
- **Externas**: Se almacenan en la memoria externa del ordenador: ficheros y bases de datos.

Las estructuras de datos internas, pueden a su vez, clasificarse en:

- **Estáticas**: El tamaño ocupado en la memoria se define antes de que el programa se ejecute y no puede modificarse durante la ejecución del programa. Están implementadas en casi todos los lenguajes y son los registros, ficheros y matrices. Dependiendo del tipo de dato pueden ser:
  - Homogéneas: todos los subelementos son del mismo tipo: arrays o matrices
  - Heterogéneas: los subelementos contienen datos de distintos tipos: registros
- **Dinámicas**: El tamaño de la estructura se va modificando conforme avanza el programa; es decir, su tamaño cambia en tiempo de ejecución.
  - Lineales: listas, pilas, colas
  - No lineales: árboles y grafos
- **Punteros**: Los tipos de datos puntero a apuntadores almacenan direcciones de memoria que a su vez indican la posición de otras estructuras de datos o funciones del programa.

Muy relacionado está también el concepto de **tipo de dato abstracto** que es un **modelo matemático** compuesto por una colección de **operaciones** definidas sobre un conjunto de datos para el modelo (Por ejemplo, el TDA “Cadena”).

El siguiente esquema muestra una clasificación de los tipos de datos más comunes:

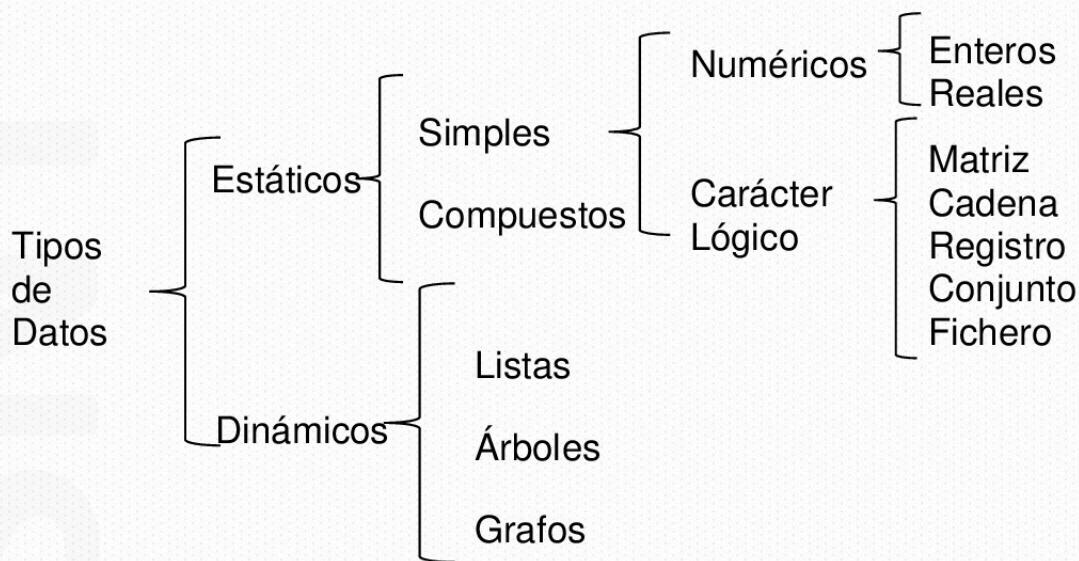


Fig. 1. Tipos de datos.

## 3. Estructuras Dinámicas

### 3.1. Listas

#### 3.1.1. Definición

Las listas son secuencias de cero o más elementos del mismo tipo abstracto de dato. Son estructuras lineales donde cada elemento de la lista tiene un único predecesor (excepto el primero) y un único sucesor (excepto el último).

Al número de elementos de la lista se le llama longitud y decimos que la lista está vacía si tiene cero elementos.

Al ser un tipo de dato dinámico, se pueden añadir nuevos elementos o suprimirlos en cualquier posición.

#### 3.1.2. Declaración

En lenguaje mnemotécnico o pseudocódigo sería algo así:

```

<identificador>: tnodo{
    contenido: <tipo de objeto>;
    siguiente: *tnodo;
}tnodo
  
```

Nótese que únicamente definimos un nodo con capacidad de apuntar a otro; dinámicamente iremos creando la lista reservando memoria para los nodos a medida que los necesitemos crear.

### 3.1.3. Estructura

Dependiendo del tipo de lista, los elementos se enlazan mediante al menos un apuntador a la dirección de memoria del siguiente elemento.

Para simplificar las operaciones sobre la lista se pueden añadir un nodo vacío (sólo apuntadores, sin contenido) como cabecera de la lista y otro como cola de la lista; de esta forma las operaciones de inserción y borrado no tienen que estar comprobando estas posiciones en particular.

### 3.1.4. Operaciones

Las operaciones que se pueden realizar sobre una lista son:

- Inserción, de un nuevo nodo
- Borrado, de un nodo existente
- Búsqueda de un valor concreto
- Recorrido completo de la lista
- Borrado completo de la lista

## Tipos de listas

### Listas contiguas

Se implementan mediante vectores, y la inserción y borrado de un elemento necesitará una translación de los elementos de la lista.

Los elementos son adyacentes en memoria y tendrán un límite inferior y superior.

### Listas enlazadas

Los elementos se colocan en posiciones de memoria que no son adyacentes, por lo que cada elemento necesitará almacenar la posición del siguiente en la lista.

Esto permite una gran versatilidad en la inserción y en el borrado de nodos, dotándolas de un gran potencial.

Normalmente se implementan de forma dinámica si el lenguaje lo permite.

### Listas circulares

Consisten en una lista enlazada a la que se le añade la funcionalidad de que el último elemento apunta al primero de la lista

## Listas doblemente enlazadas

En este tipo de listas cada nodo contiene dos apuntadores, uno al elemento siguiente y otro al elemento anterior. Esta implementación permite recorrer la lista en ambos sentidos.

## Listas doblemente enlazadas circulares

Es una lista doblemente enlazada cuyo nodo final apunta al primero de la lista (además de al anterior por el doble enlace).

## Listas de Listas

También se conocen como multilista. Formado por una lista enlazada, en la que cada nodo puede apuntar además de al siguiente nodo puede apuntar a una nueva lista enlazada.

## Pilas

Las pilas son listas con estructura LIFO (Last Input, First Output) y están diseñadas para recuperar sus elementos en orden inverso a como se introdujeron.

Tanto la extracción como la inserción de elementos en la pila se realizan por la parte superior, por lo tanto, sólo se puede acceder al último elemento insertado.

Las operaciones que se pueden realizar con una pila son:

- Crear.
- Destruir.
- Top: devuelve el tope de la pila sin borrarlo.
- Pop: elimina el tope de la pila.
- Push: inserta un nuevo elemento en la pila.

## Colas

Una cola también es una estructura de datos lineal de tipo FIFO (First Input, First Output) donde las inserciones se realizan siempre por un extremo y el borrado siempre por el otro.

Las operaciones que se pueden realizar con una Cola son:

- Crear la cola
- Destruir la cola
- Poner en cola, inserta un elemento
- Frente, devuelve el primer elemento de la cola sin eliminarlo
- Quitar de la cola, elimina el primer elemento de la cola.
- Vacía, pregunta si la cola está vacía.

## 3.2. Árboles

### 3.2.1. Definición

Un árbol es un tipo abstracto de datos (TDA) ampliamente usado que imita la estructura jerárquica de las ramas un árbol (se representa invertido) de forma que existe una relación jerárquica, y un único nodo principal, denominado raíz, precede a todos los demás a los que se denomina hijos.

Más formalmente, un árbol es un grafo conexo, acíclico y no dirigido.

De forma recursiva:

Existe un único nodo especial denominado raíz del árbol. Dicho nodo se define como el único que no tiene padre.

Los nodos restantes se dividen en  $m \geq 0$  subconjuntos distintos: A<sub>1</sub>, A<sub>2</sub>...A<sub>m</sub>, cada uno de los cuales es a su vez, un árbol y son denominados subárboles de la raíz.

### 3.2.2. Estructura

La relación jerárquica existente en un árbol, se establece gráficamente mediante un arco del nodo padre a cada uno de sus hijos.

De esta forma se distinguen los siguientes tipos de nodos:

- Nodo raíz: El único nodo del árbol sin padre (sin ascendente)
- Nodo hoja: No tiene descendientes, también se conocen como nodos terminales.
- Nodos intermedios: Aquellos que no son ni raíz ni hoja.

En un árbol además tendremos las siguientes definiciones:

- Denominamos grado de un nodo al número de hijos que tiene un nodo.
- Longitud de camino de un nodo: número de arcos que deben ser recorridos hasta llegar a él, pasando por la raíz.
- Nivel: número de ramas que hay que recorrer para llegar desde la raíz a un nodo.

Además, un árbol se caracteriza por:

- Profundidad o altura: Es el camino más largo alcanzado por alguno de los nodos del árbol.
- Peso: número de hojas del árbol.

### 3.2.3. Tipos de árboles

#### Árboles no binarios

En este tipo de árboles cada nodo puede tener 0, 1, 2 o más hijos por nodo.

## Árboles binarios

Estos árboles pueden tener como máximo dos hijos por nodo.

Los árboles binarios son más fáciles de programar que los generales. En estos es imprescindible deducir cuántas ramas o caminos se desprenden de un nodo en un momento dado. Por eso, dado que de cada subárbol binario cuelgan como máximo 2 subárboles, su programación será más sencilla.

Un árbol binario se dice equilibrado si las diferencias de las alturas de los dos subárboles se diferencian en una unidad como máximo.

Un árbol binario de nivel  $h$  puede tener como máximo  $2^h - 1$  nodos. La altura del árbol binario lleno, de  $n$  nodos es  $\log_2(n+1)$ ; a la inversa, el número máximo de nodos es de  $(2^h) - 1$ .

## Árboles binarios de búsqueda

Un árbol binario de búsqueda es un árbol binario que se construye teniendo en cuenta tres premisas:

- El primer elemento se utiliza para crear el nodo raíz.
- Los valores del árbol deben ser tales que pueda existir un orden.
- En cualquier nodo, todos los valores del subárbol izquierdo del nodo son menores o iguales al valor del nodo. De modo similar, todos los valores del subárbol derecho deben ser mayores que el valor del nodo.

Los árboles binarios de búsqueda cumplen la propiedad de que al realizar un recorrido en in-orden los valores de los nodos aparecen siempre en orden creciente.

Mantener el orden en el árbol recae en las operaciones de borrado e inserción.

## Árboles B

Un árbol B es un árbol de búsqueda balanceado, muy utilizado en bases de datos, que no tiene por qué ser binario. Un árbol B de orden M puede tener hasta M hijos en sus nodos.

Un árbol-B se mantiene balanceado porque requiere que todos los nodos hoja se encuentren a la misma altura.

### 3.2.4. Almacenamiento en memoria

Un árbol es una estructura dinámica. Su representación en el interior de una computadora se realiza utilizando punteros. Cada nodo puede incluir varios punteros: uno para dirigirse al padre y cada uno de los restantes para dirigirse a cada uno de los hijos. Esto permite moverse con gran facilidad dentro del árbol en cualquier dirección, pero representa el inconveniente de que el número de punteros para cada nodo no es fijo y puede ser excesivamente grande.

Normalmente se utiliza otra estructura con sólo tres punteros por nodo: uno para el padre, otro para el primer hijo y el tercero para el siguiente hermano.

### 3.2.5. Operaciones

#### Recorridos sobre un árbol

Una vez creado un árbol, la mayoría de las operaciones que podamos realizar sobre él, requieren conocer qué elementos tiene, por lo que hay que recorrerlos. Se denomina recorrido de un árbol al proceso que permite acceder una sola vez a cada uno de los nodos del árbol.

Existen métodos que realizan un recorrido exhaustivo del árbol y proporcionan como resultado una lista de nodos del mismo. Vamos a ver los tres más importantes:

- Recorrido en **Preorden**: listamos la raíz del árbol y si existe, listamos el subárbol izquierdo recursivamente en preorden y a continuación si existe, el subárbol derecho recursivamente en preorden.
- Recorrido en **Inorden**: listamos si existe el subárbol izquierdo en Inorden de forma recursiva, a continuación, listamos la raíz del árbol y seguidamente listamos en Inorden si existe el subárbol derecho recursivamente.
- Recorrido en **Postorden**: Listamos subárbol izquierdo, raíz y subárbol derecho, de forma recursiva, igual que en los recorridos anteriores.

#### Otras operaciones sobre árboles

- Creación del árbol: Inicializa el árbol al estado vacío. Dicho árbol estará definido a través del puntero que apunta al nodo superior del árbol, que inicialmente se encuentra vacío (null)
- Construcción del árbol: Este procedimiento irá introduciendo nodos el árbol de forma equilibrada o no según el tipo de árbol a partir de una información dada (por ejemplo, una lista)
- Búsqueda de un nodo: localiza el nodo con el recorrido indicado (inorden, preorden, postorden)
- Inserción de un nodo: primero localiza dónde insertarlo dependiendo del tipo de árbol.
- Borrado de un nodo: Dependiendo del tipo de nodo a insertar, si es una hoja o no, y del tipo de árbol, utilizará algoritmos diferentes, ya que en caso de no ser hoja será necesario reequilibrar el árbol.

## 3.3. Grafos

### 3.3.1. Definición

Un grafo es una estructura lineal mucho más compleja que un árbol. En un grafo no existe una relación de jerarquía entre los nodos, por lo que se utiliza para modelizar relaciones más generales que una jerarquía entre objetos.

Un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones binarias entre elementos de un conjunto. Típicamente, un grafo se representa gráficamente como un conjunto de puntos (vértices o nodos) unidos por líneas (aristas).

Un camino es una secuencia de uno o más arcos que conectan dos nodos. La longitud de un camino es el número de arcos que comprende.

### 3.3.2. Tipos de grafos

#### Grafos sencillos

Un grafo se denomina sencillo si no tiene lazos, es decir, si no existen arcos entre el mismo elemento y no existe más que un arco para unir dos nodos. Cuando un grafo no es sencillo es múltiple.

#### Grafos dirigidos

En un grafo dirigido, un arco es un par ordenado de vértices  $(v,w)$  en el cual  $v$  es el extremo inicial u origen y  $w$  es el extremo final o destino del arco. El arco dirigido  $(v,w)$  se representa con una flecha que sale de  $v$  y llega a  $w$ . Se dice que  $w$  es adyacente a  $v$ , pero no al revés. La existencia del arco  $(v,w)$  no implica la existencia del arco  $(w,v)$ .

En los grafos no dirigidos, los vértices están relacionados, pero no se apuntan unos a otros. La dirección no es importante. Dos vértices son adyacentes si hay un arco común que los une.

Por ejemplo, la red de aguas de una población puede ser considerada como un grafo dirigido, ya que cada tubería sólo admite que el agua la recorra en un único sentido. La red de carreteras interurbanas representa un grafo no dirigido, puesto que una misma carretera puede ser recorrida en ambos sentidos.

#### Grafos conectados

En los grafos conectados siempre existe un camino entre cualquier conjunto de vértices. En los desconectados hay vértices que no están unidos por ningún camino.

#### Grafos ponderados

En muchas aplicaciones es útil asociar información a los vértices y arcos de un grafo dirigido. Para este propósito es posible usar un grafo dirigido etiquetado, en el cual, cada nodo, cada arco o ambos pueden tener una etiqueta asociada. Una etiqueta puede ser un nombre, un valor, un costo o cualquier tipo de datos dado. Estos grafos también se llaman grafos ponderados o con peso.

### 3.3.3. Representación de los grafos

Una decisión que implica el uso de grafos es la forma en la que se almacenaran en memoria. Existen varias posibilidades:

#### Matriz de adyacencia

Si  $G$  es un grafo dirigido de  $n$  nodos, el grafo  $G$  lo representaremos por una matriz  $N \times N$  tal que  $G[i,j]=0$  si no existe un arco entre el nodo  $i$  y el nodo  $j$ , y  $1$  (o el peso del arco en caso de tenerlo) si existe el arco.

Esta representación es tremadamente sencilla, pero tiene la limitación del uso de una estructura estática de datos, por lo que el número de nodos va a estar limitado.

### Listas de adyacencia

Otras representaciones más generales están basadas en la utilización de punteros. Una estructura que representa un grafo adecuadamente es una lista de listas.

En la lista principal se encuentran todos los nodos de G. Cada nodo de esta lista tiene asociada una lista auxiliar donde se indican los nodos que están conectados a él (que son adyacentes)

#### 3.3.4. Operaciones

Las operaciones que puede incluir un grafo son muy variadas.

Además de las clásicas de búsqueda de un elemento, inserción o eliminación de un nodo o arco, también podemos encontrarnos con ejecución de algoritmos que busquen caminos más cortos entre dos vértices, o recorridos del grafo que ejecuten alguna operación sobre todos los vértices visitados, por citar algunas operaciones de las más usuales. Siempre va a depender del problema que queramos abordar.

## 4. Conclusiones

Según el autor **J. Glenn Brookshear**, en su libro “**Introducción a la computación**” (2012), una ventaja de las estructuras estáticas es que son más **fáciles** de manejar, ya que las estructuras dinámicas tienen que llevar a cabo varias tareas adicionales, como, por ejemplo, buscar bloques de memoria libres, lo que afecta al rendimiento. Por el contrario, las estructuras dinámicas son más **versátiles**, ya que no siempre se conoce previamente el número de elementos que se van a procesar.

Programar **eficazmente implica elegir un tipo de estructura adecuada** de datos a la tarea que se desea programar.

Es necesario indicar que la distinción entre estructuras estáticas y dinámicas es sólo aplicable a los lenguajes de programación **compilados**, mientras que, en los lenguajes **interpretados**, como por ejemplo **PHP**, los vectores pueden crecer de forma dinámica, o como en **Java**, el tamaño de los vectores se puede determinar en tiempo de ejecución, ya que en estos lenguajes se implementa un método de reserva dinámica de memoria, aplicable a cualquier tipo de dato compuesto.

### 4.1. Relación con el currículo

Este tema es aplicado en el aula en los módulos profesionales siguientes, con las atribuciones docentes indicadas (PES/SAI):

#### Grado Superior

- Lenguajes de Marcas y Sistemas de Gestión de la Información (DAW - DAM –ASIR) (PES)
- Programación (DAW / DAM) (PES)
- Entornos de desarrollo: (DAW / DAM) (PES)

#### **ESO y Bachiller**

- Tecnologías de la Información y la Comunicación I (PES)
- Tecnologías de la Información y la Comunicación II (PES)
- Programación y computación (libre configuración autonómica) (PES)

## 5. Bibliografía

- Y. Daniel Liang. , Introduction to Java Programming and Data Structures, Comprehensive Version. Pearson, 11<sup>a</sup> edición. 2017.
- Francisco Javier Moldes Teo. , Java 9. Anaya. 2017.
- J, Glenn Brookshear, Introducción a la computación. Edt. Pearson, 11<sup>º</sup> edición. 2012
- Luis Joyanes Aguilar , Fundamentos de programación. Algoritmos, estructuras de datos y objetos., McGraw-Hill, 4<sup>º</sup> edición. 2008.
- Langsam, Augenstein y Tanembaum: “Estructuras de Datos con C y C++”, Prentice-Hall 1997
- Prieto A., Lloris A. y Torres J.C.: Introducción a la Informática, McGraw-Hill, 4<sup>a</sup> ed 2010

