Estructuras de datos

Sitio: Centros - Cádiz
Curso: Programación
Libro: Estructuras de datos

Imprimido por: Barroso López, Carlos
Día: martes, 21 de mayo de 2024, 23:28

Tabla de contenidos

1. Definición y tipos

1. Definición

Hasta el momento hemos trabajado básicamente con datos simples o primitivos, tales como `int`, `float` o `bool`. Una variable de un **tipo de dato simple** almacena un único valor, el cual es referenciado mediante su identificador.

Por el contrario, un **tipo de dato compuesto** o estructurado puede almacenar múltiples datos, los cuales pueden ser referenciados de forma individual mediante su identificador y el uso de operadores específicos.

Estos tipos de datos están construidos a partir de otros tipos de datos (simples o compuestos) y son conocidos como **estructuras de datos**.

2. Tipos

Las estructuras de datos se pueden clasificar, según la forma de acceder y almacenar sus elementos en memoria, en: *estáticas y dinámicas*

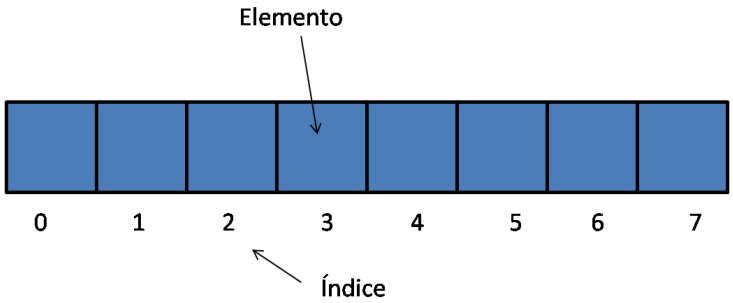
2.1. Estructuras estáticas

Sus elementos ocupan **posiciones de memoria contiguas**. Es posible acceder a cualquier elemento mediante un índice, en un tiempo constante independientemente de su posición (*acceso aleatorio*).

i. **De tamaño fijo** (*Gestión estática de memoria*)

Su tamaño en memoria se define **en tiempo de compilación**, y no varía durante la ejecución. Por tanto, la cantidad de elementos que pueden almacenar es fija.

Ejemplo: El **array**, de una o varias dimensiones.



ii. **De tamaño variable** (*Gestión dinámica de memoria*)

Es posible añadir o eliminar elementos en **tiempo de ejecución**.

Nota: En C++ la gestión dinámica de memoria se realiza de forma transparente al programador, a diferencia de C, que es el programador quien debe llevarla a cabo mediante el uso de punteros y asignación dinámica de memoria.

Ejemplos: *clase string o contenedor vector de C++*

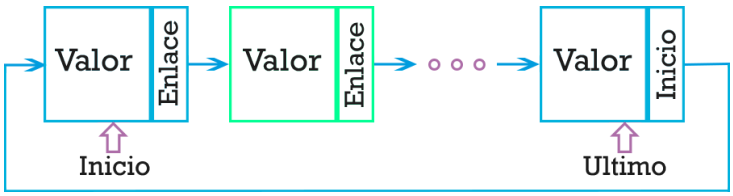
2.2. Estructuras dinámicas

A diferencia de las estructuras estáticas, sus elementos ocupan **posiciones de memoria no contiguas**. Para acceder a un determinado elemento es necesario recorrer la estructura hasta posicionarse en él. Por tanto, el tiempo para acceder a un elemento varía según su posición.

Cada elemento contiene una o varias **referencias** a los elementos adyacentes.

Su principal ventaja es que permiten la inserción y borrado de elementos intermedios de forma muy rápida.

Ejemplo: *lista enlazada*



2. Estructuras dinámicas

La estructuras dinámicas se componen de elementos que ocupan **posiciones de memoria no contiguas**, permitiendo añadir y quitar elementos sin tener que desplazar al resto.

Las estructuras dinámicas se pueden clasificar en:

- **Lineales**: Lista enlazada, Pila (Stack), Cola (Queue)
- **No lineales**: Árbol, Grafo

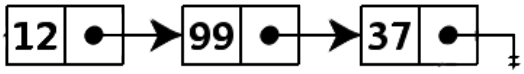
2.1. Lineales: Lista enlazada, Pila y Cola

Estructura lineal: cada elemento tiene un único predecesor y sucesor (salvo el primero y el último).

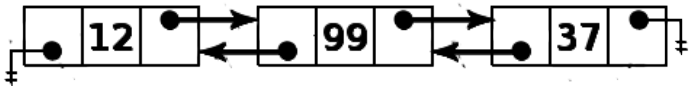
1. Lista enlazada

1.1. Tipos

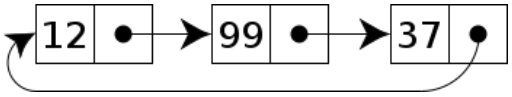
Lista (simplemente) enlazada: se puede recorrer en un sólo sentido.

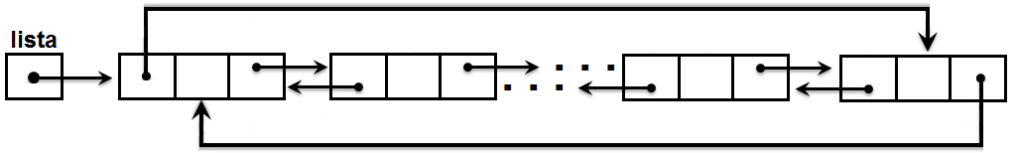


Lista doblemente enlazada: se puede recorrer en ambos sentidos.



Lista enlazada circular: el primer y último elemento están conectados. Es necesario identificar el nodo cabecera a la hora de recorrer la lista.

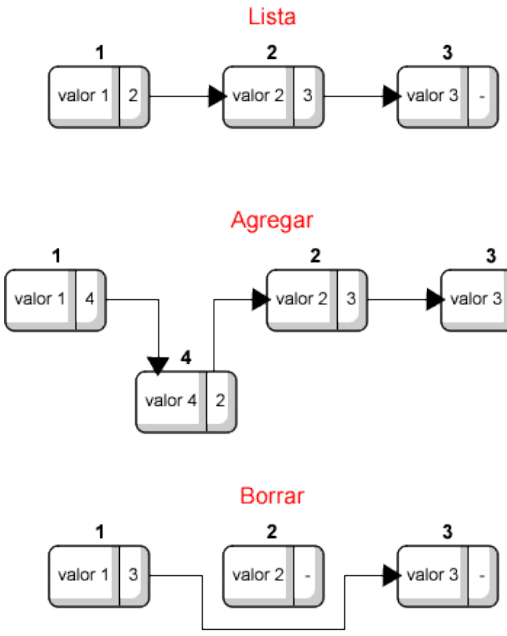




Lista doblemente enlazada circular

1.2. Operaciones

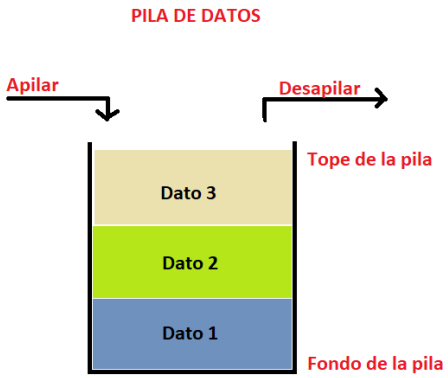
- Insertar elemento
- Borrar elemento
- Buscar elemento
- Recorrer lista



2. Pila (Stack)

Es un tipo de lista donde la inserción (**push**) y el borrado (**pop**) de elementos se realiza sólo por un extremo.

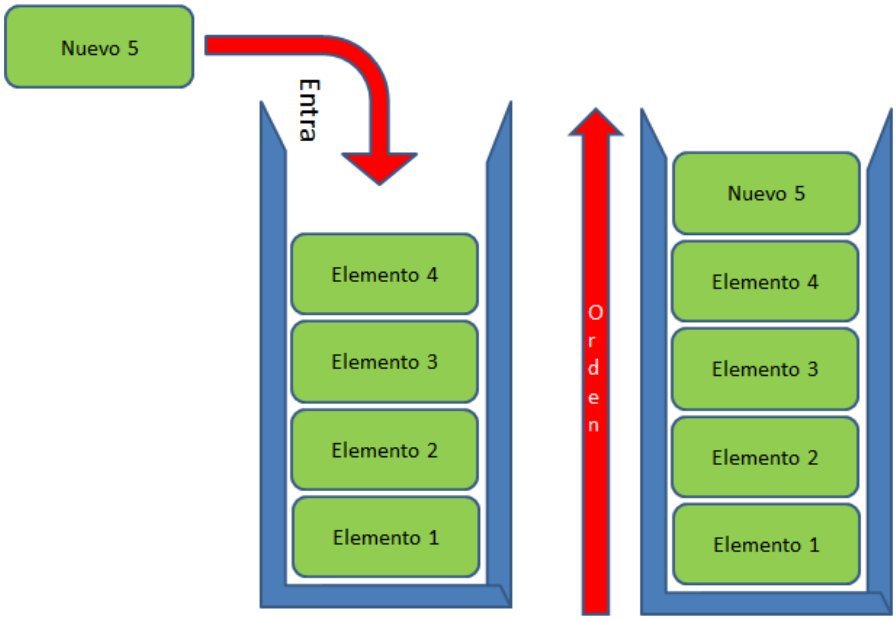
El último elemento insertado en la pila es el primero que se puede sacar. Es una estructura **LIFO** (*last in, first out*).



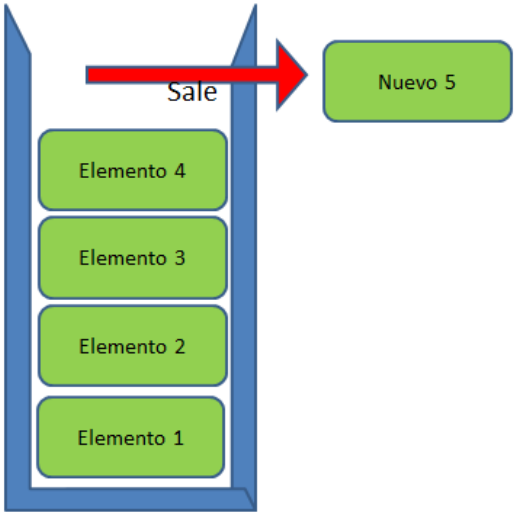
2.1. Operaciones

- **Push** (apilar): inserta un elemento en el tope.
- **Pop** (desapilar): devuelve el elemento que está en el tope, y lo elimina.
- **Tope**: devuelve el elemento que está en el tope, sin eliminarlo.

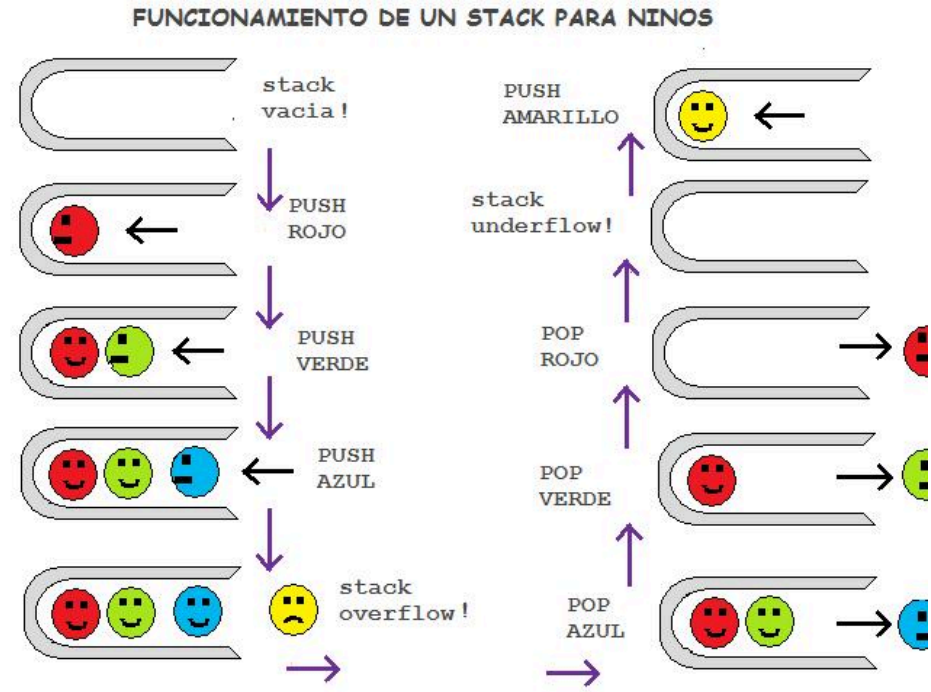
PUSH



POP



Ejemplo de funcionamiento:

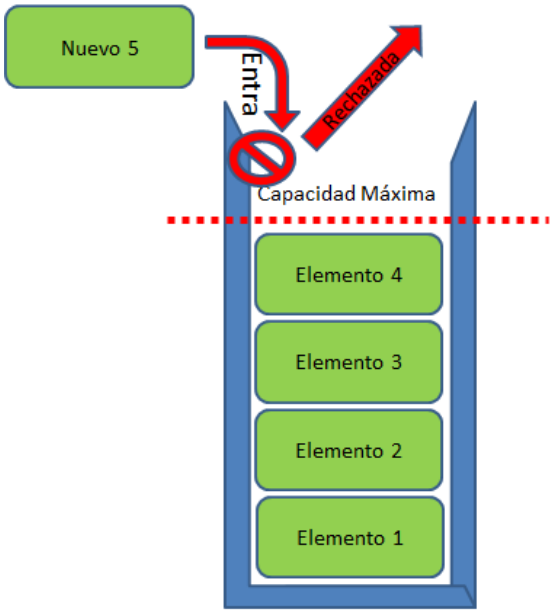


Ejemplo de uso:

Almacenar la información de las diferentes llamadas recursivas de una función.

2.2. Desbordamiento de la pila (Stack overflow)

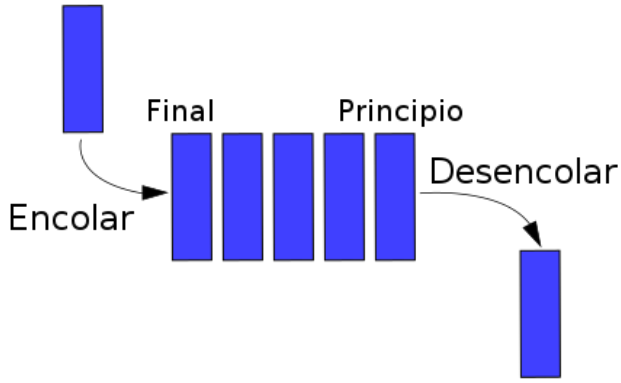
Ocurre cuando se realiza una operación *push* sobre una pila con todos sus elementos ocupados. En el mejor de los casos, el elemento es rechazado.



3. Cola (Queue)

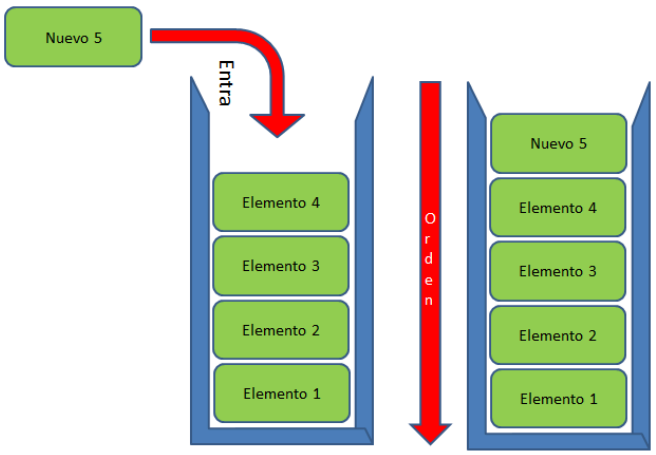
Una Cola o Queue es un tipo de lista donde la inserción de datos se realiza por un extremo y la eliminación por el otro.

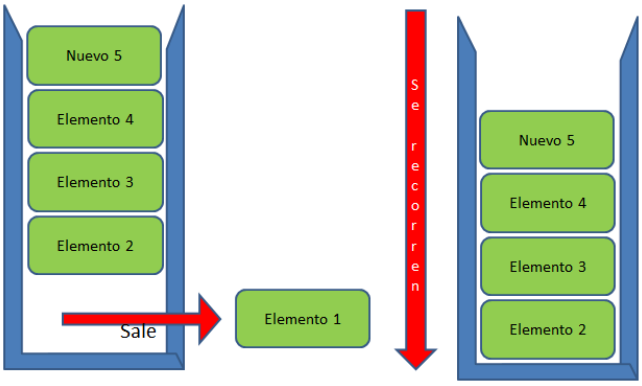
Es una estructura **FIFO** (*first in, first out*). Esto quiere decir que el primer elemento en entrar en la cola será el primero en salir.



3.1. Operaciones

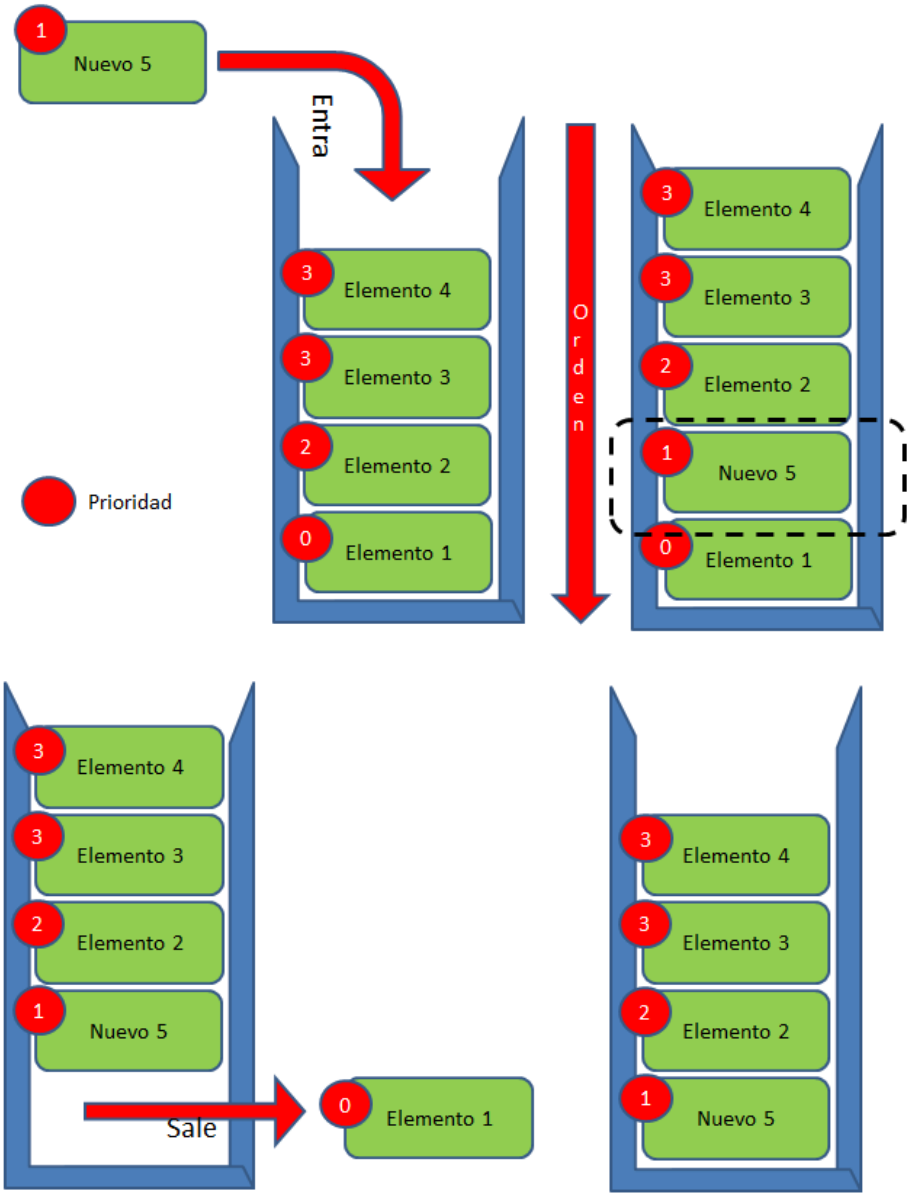
- **Poner en cola**: inserta un elemento en la cola (al final).
- **Quitar de cola**: devuelve el primer elemento, y lo elimina.
- **Frente**: devuelve el primer elemento, sin eliminarlo.





3.2. Cola de prioridad

Es una variante en la cual los elementos de la cola tienen una determinada prioridad. Cuando un elemento nuevo entra en la cola, se coloca en la posición que determine su prioridad.


1
Nuevo 5
Entra
3
Elemento 4
3
Elemento 3
2
Elemento 2
0
Elemento 1
Prioridad
Orden
3
Elemento 4
3
Elemento 3
2
Elemento 2
1
Nuevo 5
0
Elemento 1
3
Elemento 4
3
Elemento 3
2
Elemento 2
1
Nuevo 5
Sale
0
Elemento 1
3
Elemento 4
3
Elemento 3
2
Elemento 2
1
Nuevo 5

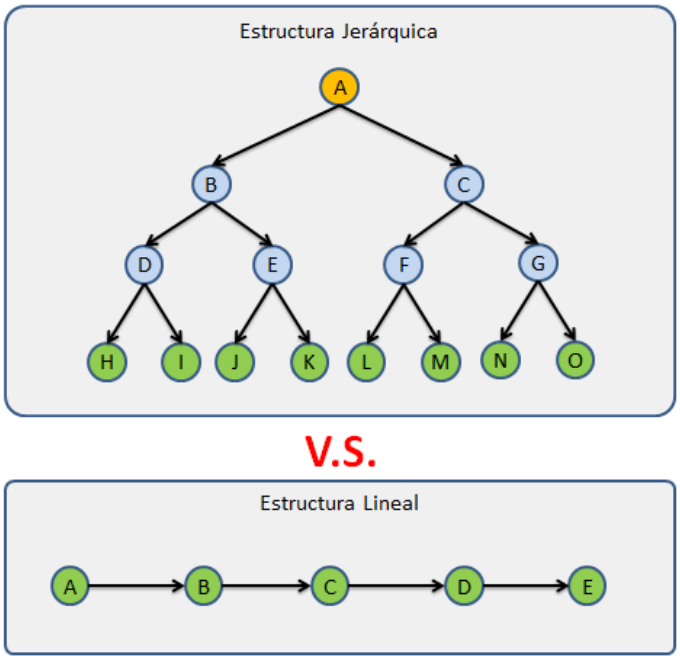
2.2. No lineales: Árbol y Grafo

Estructura no lineal: cada elemento puede tener varios sucesores o varios predecesores.

1. Árbol

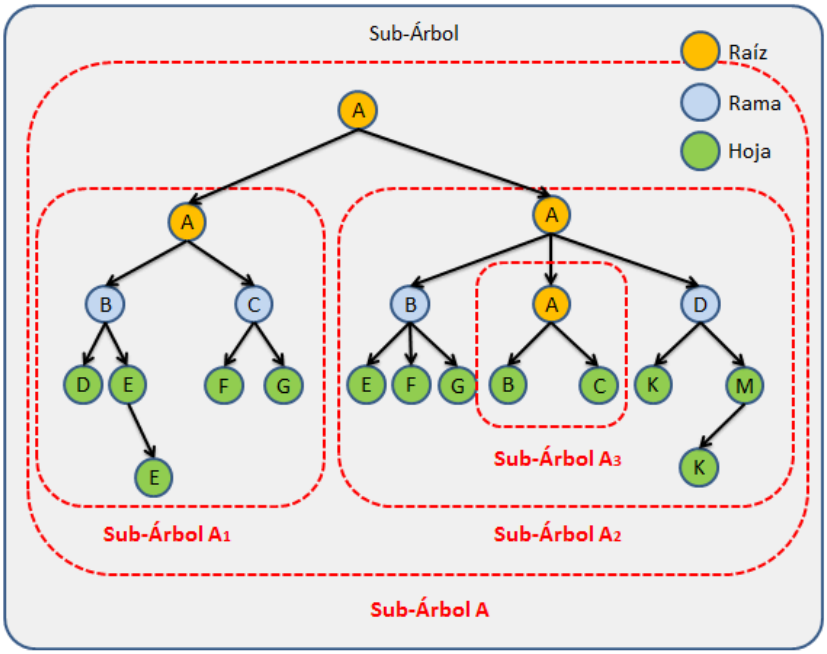
1.1. Definición

Estructura muy utilizada para representar información que mantiene una **relación jerárquica**. Cada elemento tiene un único antecesor y puede tener varios sucesores.

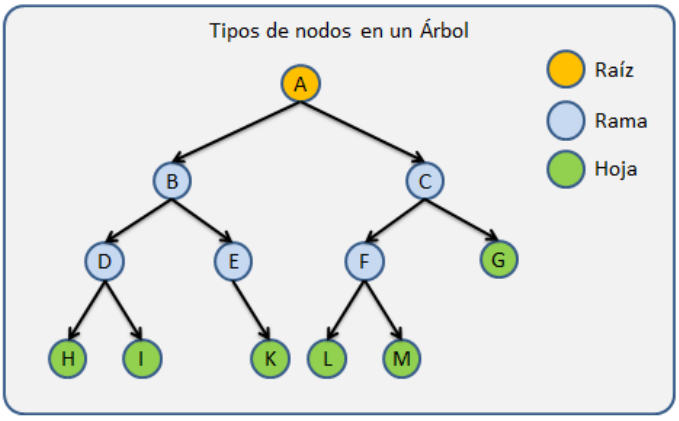


Un árbol es un conjunto de elementos denominados **nodos**, compuesto por:

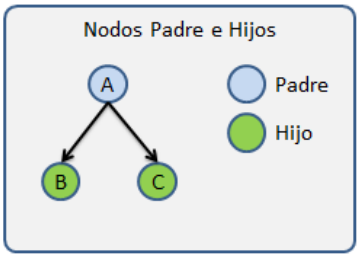
- Un nodo especial denominado **raíz**.
- Un conjunto de nodos que descienden del raíz denominados **hijos**, que a su vez son árboles. Estos nodos son el nodo raíz de su correspondiente árbol.

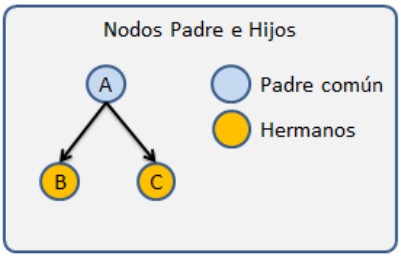


1.2. Tipos de nodos



- **Hijo**: nodo que desciende de forma directa de otro denominado **Padre**. (Relación jerárquica)
- **Hermanos**: nodos con un mismo padre.
- **Raíz**: único nodo que no tiene padre.
- **Terminal u hoja**: nodo que no tiene ningún descendiente.





1.3. Características de un árbol

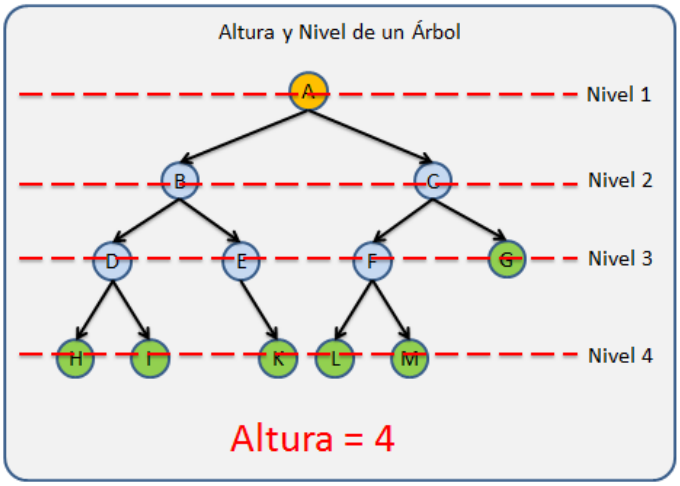
Nivel

Conjunto de nodos de una misma generación, es decir, con la misma distancia al nodo raíz. Habitualmente se asigna el nivel 1* al nodo raíz, incrementándose en cada nueva generación.

(* Hay autores que le asignan el nivel 0)

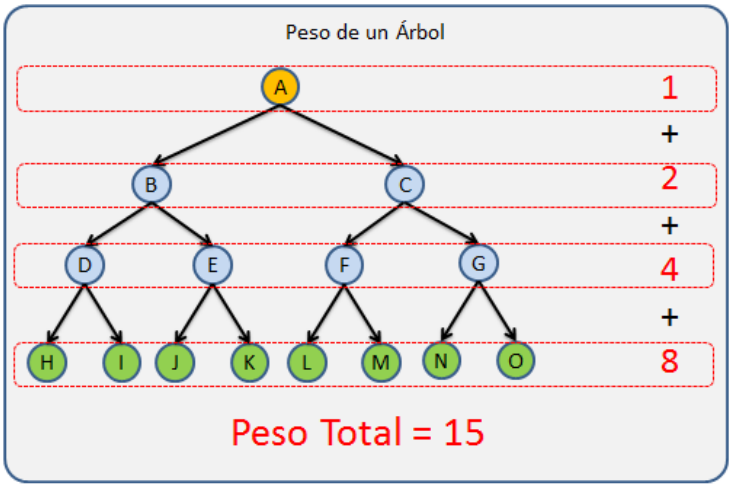
Altura

Número de niveles o generaciones de nodos que tiene un árbol.



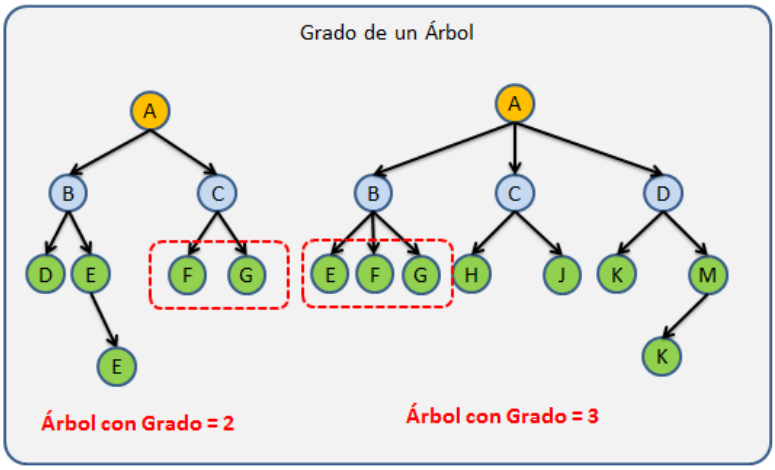
Peso

Número total de nodos que tiene un árbol.



Orden y Grado

Orden es el número máximo de hijos que puede tener un nodo. El **grado** se refiere al número mayor de hijos que tiene alguno de los nodos del árbol y esta limitado por el orden.

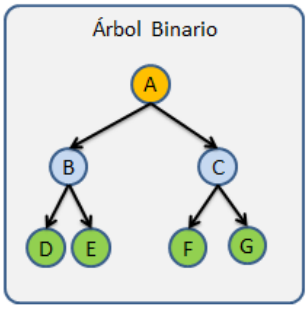


Subárbol

Un subárbol de un árbol T es aquel árbol que se forma a partir de un nodo de T (que actúa como nodo raíz) y todos sus descendientes.

2. Árbol binario

Un árbol binario es un árbol de grado 2, es decir, cada nodo solo puede tener **como máximo 2 hijos**.

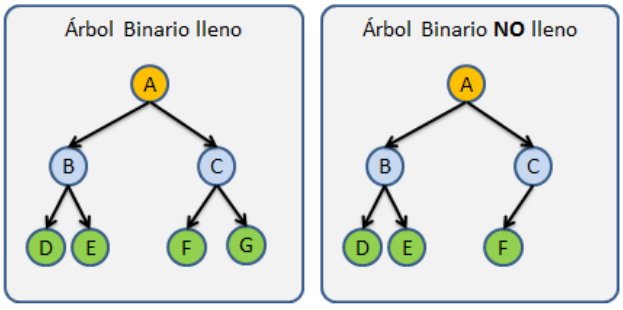


Este tipo de árbol es de especial relevancia, dado que es **más sencillo y eficiente de implementar**. Existen técnicas para convertir un árbol no binario en binario.

2.1. Características de los árboles binarios

AB Lleno

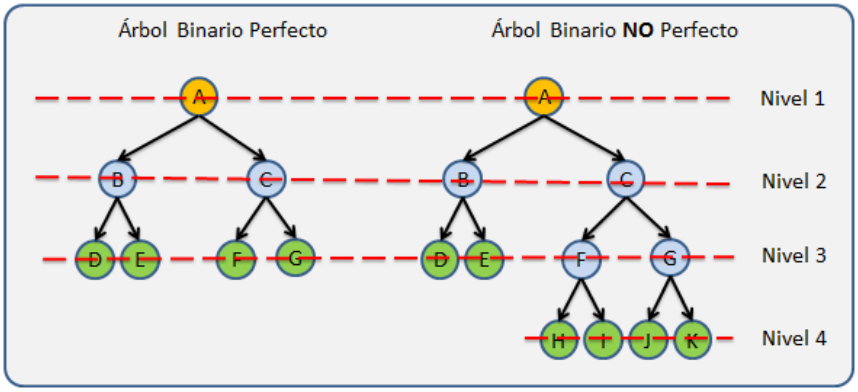
Todos sus nodos tienen 0 o 2 hijos.



C sólo tiene un hijo (árbol derecho)

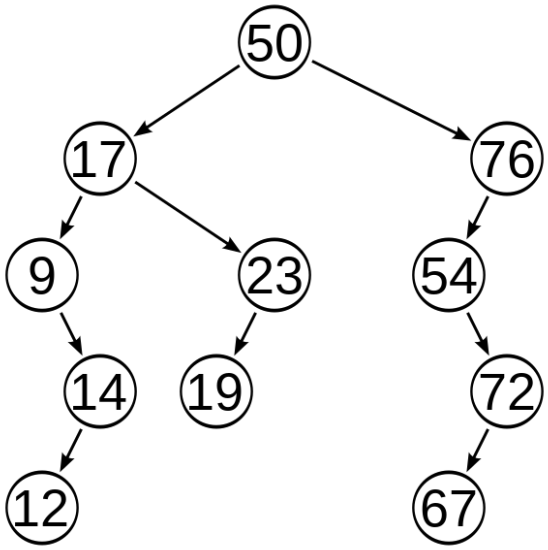
AB Perfecto

Es un AB lleno donde todos sus nodos hojas están en el mismo nivel.

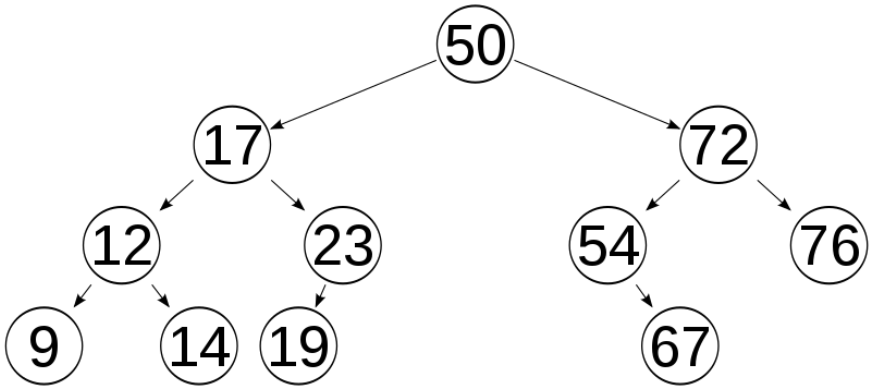


AB Equilibrado

Un AB está equilibrado si las **alturas de los dos subárboles** de cada nodo se diferencian como máximo en una unidad.



AB no equilibrado



AB equilibrado

2.2. Recorridos

Recorridos en profundidad

Existen 3 maneras distintas de recorrer un árbol en profundidad:

1. **Preorden**

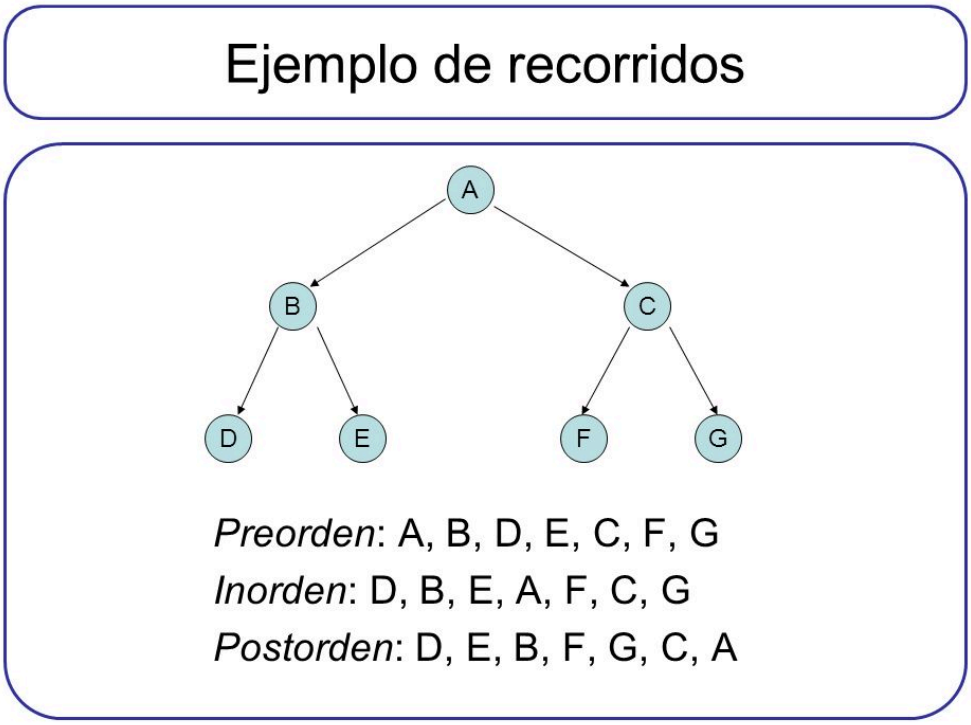
RID (*Raíz, Subárbol Izdo, Subárbol Dcho*)

2. **Inorden**

IRD (*Subárbol Izdo, Raíz, Subárbol Dcho*)

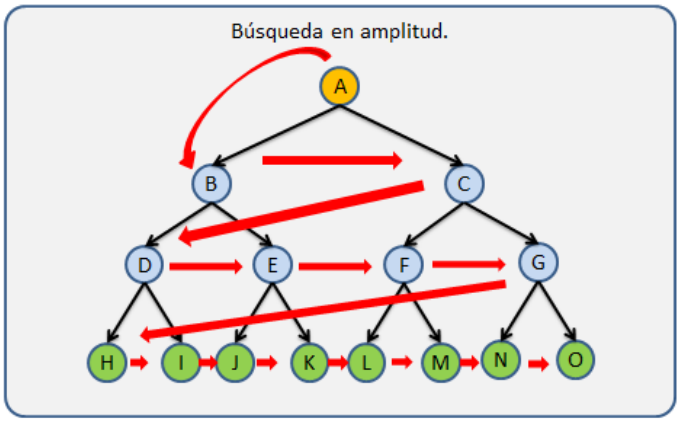
3. **Postorden**

IDR (*Subárbol Izdo, Subárbol Dcho, Raíz*)



Recorrido en amplitud

Se recorre primero la raíz, luego se recorren los demás nodos por niveles y dentro de un nivel de izquierda a derecha.



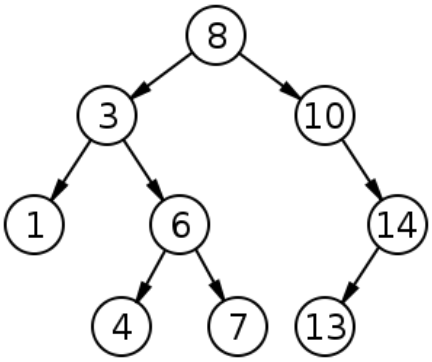
2.3. Usos de los AB

Árbol AVL

Los árboles AVL (siglas de sus autores) son árboles que **están siempre equilibrados**, para ello, si al realizar una operación de inserción o borrado se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

ABB o AB de búsqueda (BST *Binary Search Tree*)

Árbol binario que cumple que el **subárbol izquierdo** de cualquier nodo (si no está vacío) contiene valores menores que el que contiene dicho nodo, y el **subárbol derecho** (si no está vacío) contiene valores mayores.

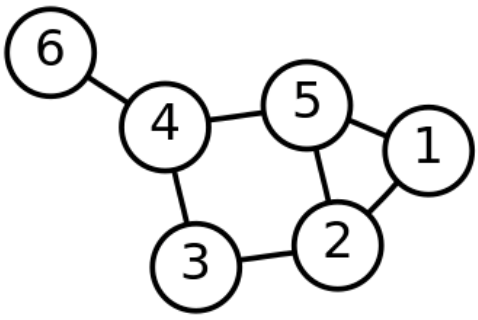


Una propiedad de los ABB es que al hacer un **recorrido** en profundidad **inorden** se obtienen los elementos ordenados de forma ascendente.

3. Grafo

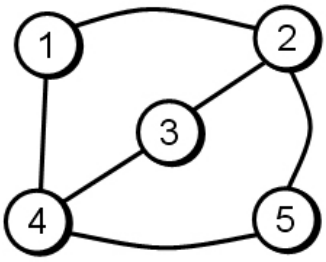
Estructura que representa un conjunto de elementos y las relaciones entre ellos. Los elementos se representan por **nodos** y las conexiones por **arcos**.

A diferencia de los árboles, no existe relación de jerarquía entre los nodos.



3.1. Representación

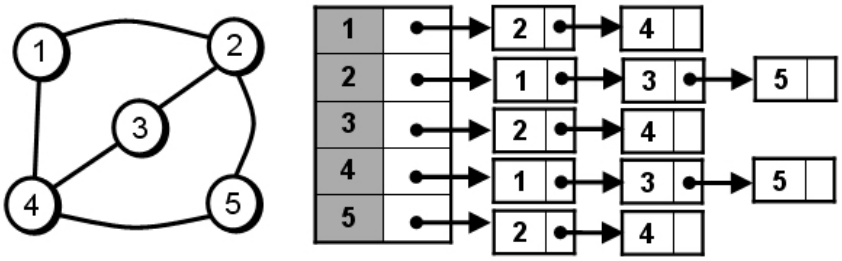
Matriz de adyacencias



M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Se asocia cada fila y cada columna a cada nodo del grafo, siendo los elementos de la matriz la relación entre los mismos, tomando los valores de 1 si existe la arista y 0 en caso contrario.

Lista de adyacencias



Se asocia a cada nodo del grafo una lista que contenga todos aquellos nodos que sean adyacentes a él.

3.2. Variantes

Grafo dirigido

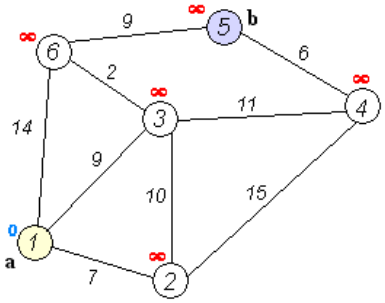
Los arcos (conexiones) tienen un único sentido, y se representan habitualmente con una **flecha**.

A --> B: existe conexión de A a B, pero no de B a A.

Grafo etiquetado o con pesos

Los arcos tienen asociado un peso o coste. Muy utilizado para encontrar los caminos de menor coste entre dos nodos.

*Ej.: Algoritmo de caminos mínimos o **algoritmo de Dijkstra***



Reiniciar tour para usuario en esta página