Teoría Estructura de datos

Andres David Bonilla Higuera
Daniel Ernesto Gómez Vásquez
Whayner Eduardo Porras Rodríguez

Estructura de datos

Universidad Pontificia Bolivariana

Lenin Javier Serrano Gil

Marzo 2021

1 Estructura de datos

Es una colección de valores, la relación que existe entre estos valores y las operaciones que podemos hacer sobre ellos; en pocas palabras se refiere a cómo los datos están organizados y cómo se pueden administrar. Una estructura de datos describe el formato en que los valores van a ser almacenados, cómo van a ser accedidos y modificados, pudiendo así existir una gran cantidad de estructuras de datos.

¿Por qué son útiles las estructuras de datos?

Las estructuras de datos son útiles porque siempre se manipularan datos, y si los datos están organizados, esta tarea será mucho más fácil.

2 Tipos de estructuras de datos

- Las estructuras contiguamente asignadas están compuestas de bloques de memoria únicos, e incluyen a los arrays, matrices, heaps, y hash tables.
- Las estructuras enlazadas están compuestas de distintos fragmentos de memoria unidos por punteros, e incluyen a los listas, arboles, y grafos.
- Los contenedores son estructuras que permiten almacenar y recuperar datos en un orden determinado sin importar su contenido, en esta se incluyen los pilas y colas.

3 Arrays

Son estructuras de datos de tamaño fijo o dinámico de modo que cada elemento puede ser eficientemente ubicado por su index ó dirección,que inicia desde 0, con el cual se puede localizar de forma rápida un elemento en el arreglo y estos elementos son únicamente de un tipo. Son la base de muchos otros tipos de estructuras de datos.

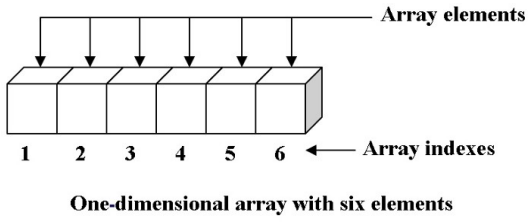


Figure 1: Arrays.

3.1 Ventajas:

- Al tener un espacio contiguo en memoria cada index de cada elemento del array apunta directamente a una dirección de memoria, de esta forma se puede acceder arbitrariamente a los datos de forma instantánea puesto que se sabe la dirección de memoria exacta. Esto deriva en un acceso de tiempo constante dado por los index.

- Los arrays son puramente datos lo que significa que no es necesario desperdiciar espacio en memoria almacenando información extra que ayude a la localización de sus elementos como es el caso de las estructuras enlazadas, los arrays tienen eficiencia de espacio.

- Localidad de memoria, es común que en la programación los datos de una estructura sean iterados (o recorridos) y los arrays son buenos para esto ya que exhiben excelente localización de memoria, permitiendo aprovechar la alta velocidad de la caché en computadoras modernas.

3.2 Desventaja:

Si es un array es fijo no se puede ajustar su tamaño a la mitad de la ejecución de un programa, esto se puede solucionar creando un array dinámico, crear un array lo suficientemente grande para almacenar los datos, pero esto deriva en un desperdicio de memoria totalmente innecesario o también se puede crear un nuevo array, doblar el tamaño de éste cada vez que se necesite crecer y copiar los datos del array anterior al nuevo array

4 Matrices:

Son estructuras de datos que permiten organizar la información en filas y columnas. Cada elemento de una matriz puede ser accedido por un par de índices (la fila y la columna). Al igual que en los arrays, los elementos de una matriz deben ser todos del mismo tipo de datos.

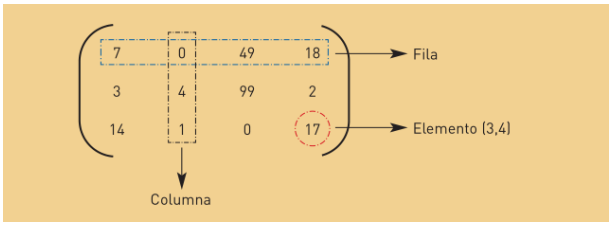


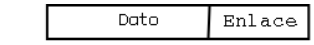
Figure 2: Matriz.

5 Estructuras enlazadas:

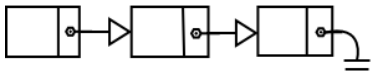
Las estructuras enlazadas son dadas por pointers o punteros, que como su nombre lo indica apuntan a una dirección de memoria donde se encuentra ubicado un valor. Los pointers son los encargados de mantener los "enlaces" entre valores de modo que es posible tener una secuencia de valores todos enlazados por pointers.

5.1 Lista Sencillamente Enlazada:

Es una estructura de datos que nos permite almacenar datos de una forma organizada, al igual que los vectores pero, a diferencia de estos, esta estructura es dinámica, por lo que no tenemos que saber "a priori" los elementos que puede contener.Cada elemento apunta al siguiente excepto el último que no tiene sucesor y el valor del enlace es null. Por ello los elementos son registros que contienen el dato a almacenar y un enlace al siguiente elemento. Los elementos de una lista, suelen recibir también el nombre de nodos de la lista.



Estructura de un nodo



Lista enlazada

Figure 3: Listas Sencillamente Enlazada.

Debe tener unos operadores asociados que permitan la manipulación de los datos que contiene. Los operadores básicos de una lista enlazada son:

- Insertar: inserta un nodo con dato x en la lista, pudiendo realizarse esta inserción al principio o final de la lista o bien en orden.
- Eliminar: elimina un nodo de la lista, puede ser según la posición o por el dato.
- Buscar: busca un elemento en la lista.
- Localizar: obtiene la posición del nodo en la lista.
- Vaciar: borra todos los elementos de la lista

5.2 Lista Doblemente Enlazada:

Es una estructura de datos dinámico lineal pero, a diferencia de la lista sencillamente enlazada, cada nodo de la lista doblemente enlazada contiene dos punteros, de forma que uno apunta al siguiente nodo y el otro al predecesor. Esta característica, permite que se pueda recorrer la lista en ambos sentidos, cosa que no es posible en las listas simples.

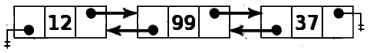


Figure 4: Listas Doblemente Enlazada.

5.3 Lista circular simplemente enlazada:

La lista circular es una especie de lista enlazada simple, pero que posee una característica adicional para el desplazamiento dentro de la lista: esta no tiene fin.

Para que la lista sea sin fin, el puntero siguiente del último elemento apuntará hacia el primer elemento de la lista en lugar de apuntar al valor NULL, como hemos visto en el caso de listas enlazadas simples o doblemente enlazadas.

En las listas circulares, nunca se llega a una posición en la que ya no sea posible desplazarse. Cuando se llegue al último elemento, el desplazamiento volverá a comenzar desde el primer elemento.

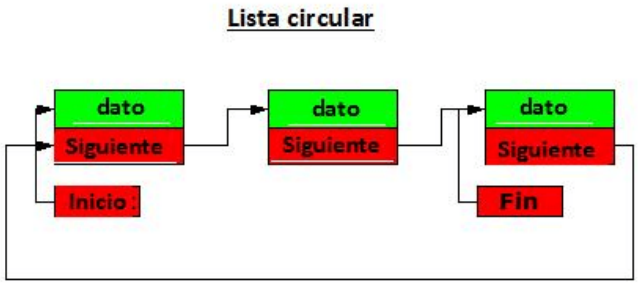


Figure 5: Listas Circular Sencillamente Enlazada.

5.4 Lista circular doblemente enlazada:

En una lista enlazada doblemente circular, cada nodo tiene dos enlaces, similares a los de la lista doblemente enlazada, excepto que el enlace anterior del primer nodo apunta al último y el enlace siguiente del último nodo, apunta al primero.

Como en una lista doblemente enlazada, las inserciones y eliminaciones pueden ser hechas desde cualquier punto con acceso a algún nodo cercano. Aunque estructuralmente una lista circular doblemente enlazada no tiene ni principio ni fin, un puntero de acceso externo puede establecer el nodo apuntado que está en la cabeza o al nodo cola, y así mantener el orden tan bien como en una lista doblemente enlazada.

Una lista doble circular es una estructura donde el último elemento tiene como referencia siguiente al primer elemento y la referencia al anterior del primer elemento de la lista también es el último. Cada elemento esta doblemente enlazado.

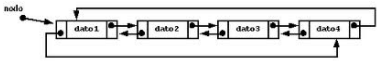


Figure 6: Listas Circular Doblemente Enlazada.

5.5 Árboles:

Es la estructura de datos mas utilizada, pero también una de las mas complejas,se caracterizan por almacenar sus nodos en forma jerárquica y no en forma lineal.

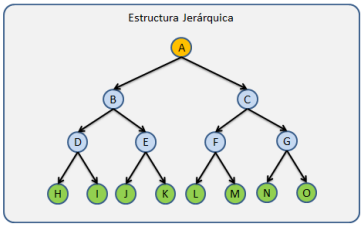


Figure 7: Árboles.

Para comprender mejor que es un árbol se necesita saber los siguiente:

- Nodos: Se le llama Nodo a cada elemento que contiene un Árbol.
- Nodo Raíz: Se refiere al primer nodo de un Árbol, Solo un nodo del Árbol puede ser la Raíz.
- Nodo Hoja: Son todos aquellos nodos que no tienen hijos, los cuales siempre se encuentran en los extremos de la estructura.
- Nodo Rama: Estos son todos aquellos nodos que no son la raíz y que además tiene al menos un hijo.

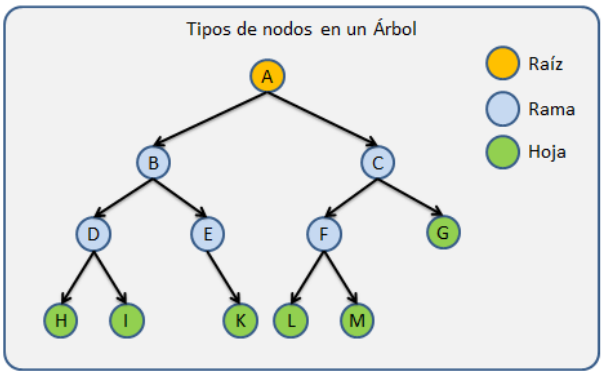


Figure 8: Nodos.

- Nodo Padre: Se utiliza este termino para llamar a todos aquellos nodos que tiene al menos un hijo.
- Nodo Hijo: Los hijos son todos aquellos nodos que tiene un padre.
- Nodo Hermano: Los nodos hermanos son aquellos nodos que comparte a un mismo padre en común dentro de la estructura.

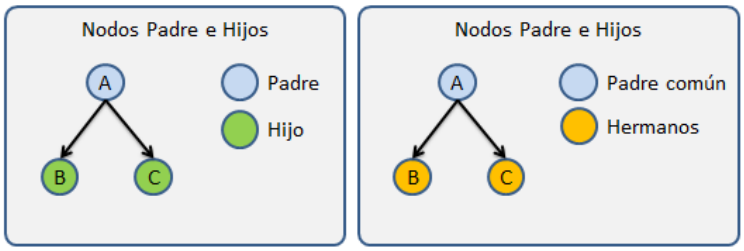


Figure 9: Nodos.

Los arboles además de los nodos también tiene:

Nivel: Se hace referencia a cada generación dentro del árbol. Por ejemplo, cuando a un nodo hoja le agregamos un hijo, el nodo hoja pasa a ser un nodo rama pero además el árbol crece una generación por lo que el Árbol tiene un nivel mas.Cada generación tiene un número de Nivel distinto que las demás generaciones.

- Un árbol vacío tiene 0 niveles
- El nivel de la Raíz es 1
- El nivel de cada nodo se calculado contando cuantos nodos existen sobre el, hasta llegar a la raíz + 1, y de forma inversa también se podría, contar cuantos nodos existes desde la raíz hasta el nodo buscado + 1.

Altura: Se le llama altura al número máximo de niveles de un Árbol.Es calculado mediante recursividad tomando el nivel mas grande de los dos sub-árboles de forma recursiva de la siguiente manera:

$$\textbf{altura} = \textbf{max(altura(hijo1), altura(hijo2),altura(hijoN))} + \textbf{1}$$

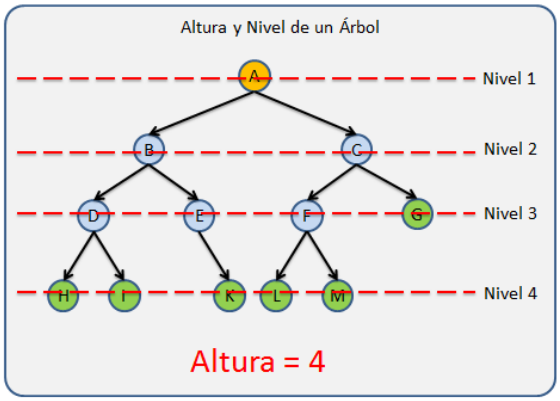


Figure 10: Nivel-Altura.

Peso: Se conoce como peso al número de nodos que tiene un Árbol. Este factor es importante por que da una idea del tamaño del árbol y el tamaño en memoria que nos puede ocupar en tiempo de ejecución (Complejidad Espacial en análisis de algoritmos).

El peso se puede calcular mediante cualquier tipo de recorrido el cual valla contando los nodo a medida que avanza sobre la estructura. El peso es un árbol es igual a la suma del peso de los sub-árboles hijos + 1.

$$\textbf{peso} = \textbf{peso(hijo1)} + \textbf{peso(hijo2)} + \textbf{peso(hijoN)} + \textbf{1}$$

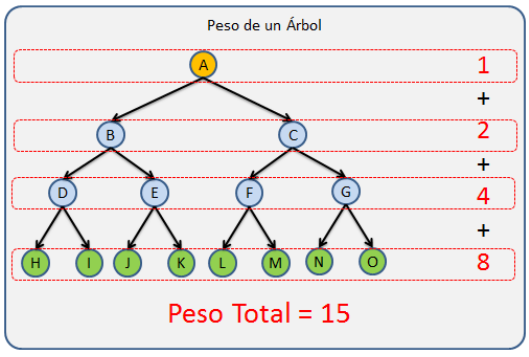


Figure 11: Peso.

Orden: El Orden de un árbol es el número máximo de hijos que puede tener un Nodo.

Un árbol con Orden = 1 no tendría sentido ya que seria una estructura lineal. Este valor no se calcula, si no que ya lo se debe conocer en el momento que se diseña la estructura, ya que si se quiere calcular lo que se obtiene es el grado.

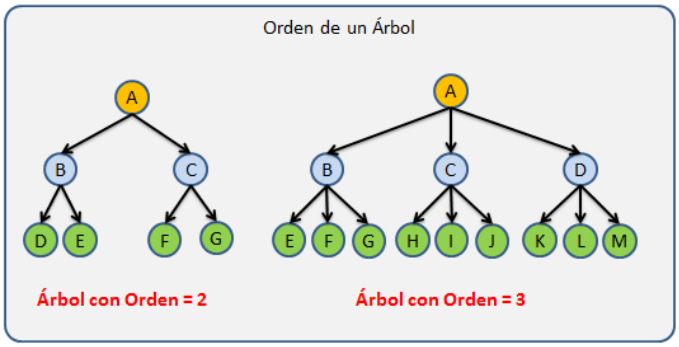


Figure 12: Orden.

Grado: Se refiere al número mayor de hijos que tiene alguno de los nodos del Árbol y esta limitado por el Orden, ya que este indica el número máximo de hijos que puede tener un nodo.

Se calcula contando de forma recursiva el número de hijos de cada sub-árbol hijo y el numero de hijos del nodo actual para tomar el mayor, esta operación se hace de forma recursiva para recorrer todo el árbol.

grado = max(contarHijos(hijo1),contarHijos(hijo2),contarHijos(hijoN), contarHijos(this))

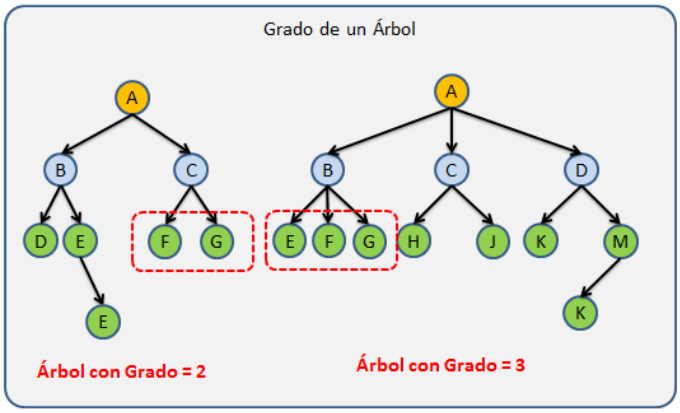


Figure 13: Grado.

Sub-Árbol: Se conoce como Sub-Árbol a todo Árbol generado a partir de una sección determinada del Árbol, Por lo que podemos decir que un Árbol es un nodo Raíz con N Sub-Árboles.

Existen escenarios donde se puede sacar un Sub-Árboles del Árbol para procesarlo de forma separada, de esta forma el Sub-Árboles pasa a ser un Árbol independiente, También se puede eliminar Sub-Árboles completos, agregarlos,entre otras operaciones.

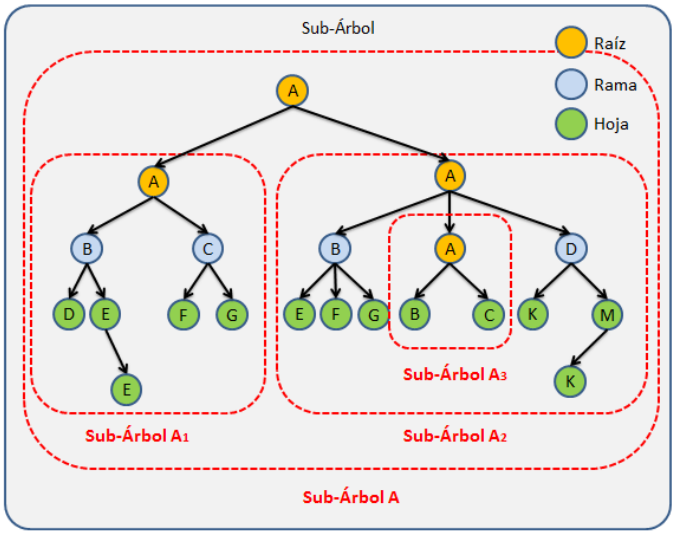


Figure 14: Sub-Árbol.

5.5.1 Recorrido sobre Árboles:

Se puede catalogar las búsqueda en tres tipos, búsquedas no informadas, las búsqueda en profundidad y las búsquedas en amplitud.

Búsquedas no informadas: Son aquellas en que se realiza el viaje por todo el árbol sin tener una pista de donde pueda estar el dato deseado. Este tipo de búsquedas también se conocen como búsquedas a ciegas.

Búsqueda en profundidad:

- Recorrido Pre-orden: El recorrido inicia en la Raíz y luego se recorre en pre-orden cada unos de los sub-árboles de izquierda a derecha.

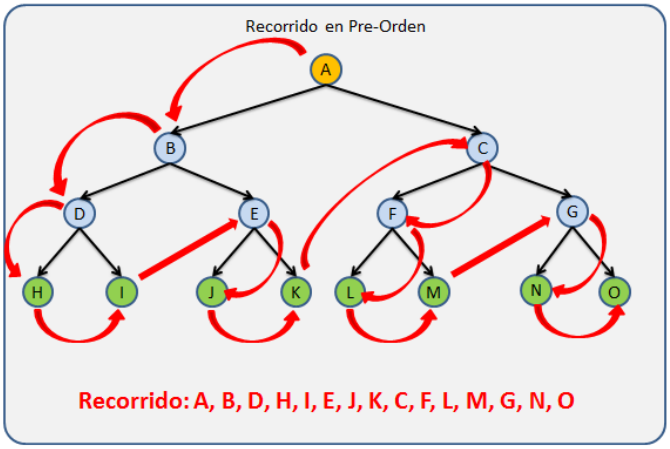


Figure 15: Pre-orden.

- Recorrido Pos-orden: Se recorre el pos-orden cada uno de los sub-árboles y al final se recorre la raíz.

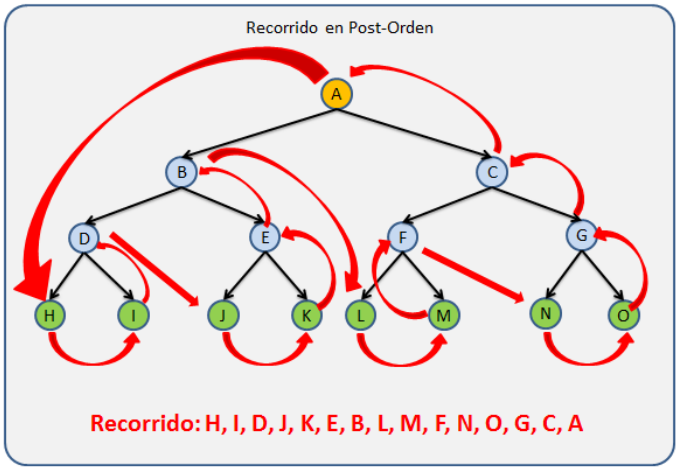


Figure 16: Pos-orden.

- Recorrido in-orden: Se recorre en in-orden el primer sub-árbol, luego se recorre la raíz y al final se recorre en in-orden los demás sub-árboles.

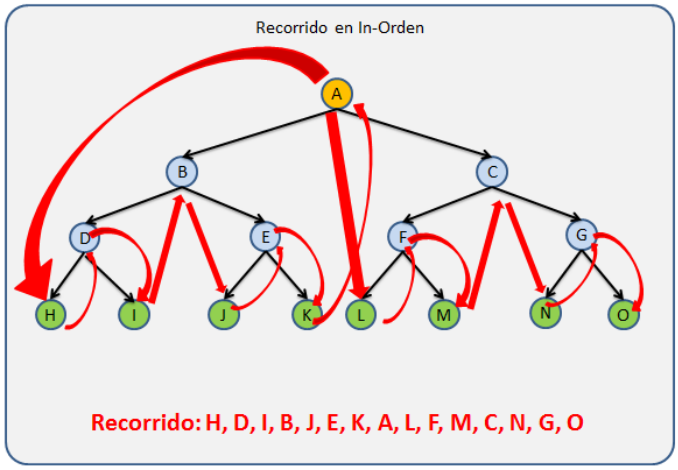


Figure 17: in-orden.

Búsqueda en amplitud: Se recorre primero la raíz, luego se recorren los otros nodos ordenados por el nivel al que pertenecen en orden de Izquierda a derecha. Este tipo de búsqueda se caracteriza por que la búsqueda se hace nivel por nivel y de izquierda a derecha.

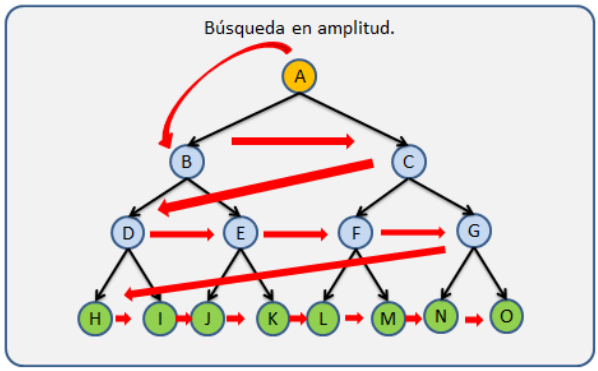


Figure 18: Búsqueda en amplitud.

5.6 Árboles binarios:

Esta estructura se caracteriza por que cada nodo solo puede tener máximo 2 hijo, dicho de otra manera es un Árbol n-ario de Grado 2.

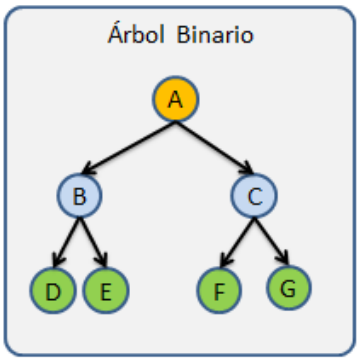


Figure 19: Árbol binario.

Árbol binario lleno: Es aquel que el que todos los nodos tiene cero o 2 hijos con excepción de la Raíz.

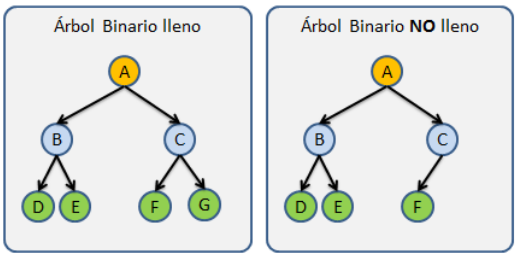


Figure 20: Árbol binario lleno.

Árbol binario perfecto: Es un Árbol lleno en donde todos las Hojas están en el mismo Nivel.

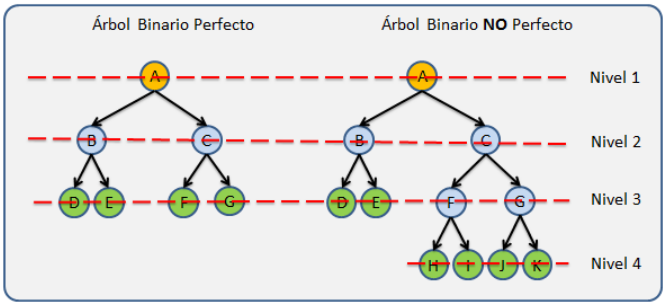


Figure 21: Árbol binario perfecto.

5.7 Árbol n-ario:

Los árboles n-arios son aquellos árboles donde el número máximo de hijos por nodo es de N, en la figura 13 podemos apreciar dos árboles con grado 2 y grado 3, estos dos arboles también los podemos definir como Árbol n-ario con n = 2 y n=3 respectivamente.

5.8 Grafos:

Un grafo es una estructura de datos dinámica muy similar a los árboles formado por un conjunto de nodos y otro conjunto de arcos. Cada arco agrupa a dos nodos que pueden ser el mismo. Cada arco puede tener peso, coste o distancia. Además de arcos pueden estar orientados o no estarlo. En caso de estar orientados se representan mediante una flecha, en caso contrario se representan por un segmento.

Un grafo se puede recorrer igual que un árbol, bien por profundidad o bien por amplitud.

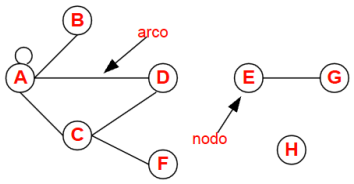


Figure 22: Grafos

6 Contenedores:

Las estructuras de tipo contenedor se caracterizan principalmente por la forma particular de recuperación ordenada de datos que soportan, y en los dos tipos principales de contenedores (pilas y colas) el orden de recuperación depende del orden de inserción.

6.1 Pilas:

Un stack o pila, soporta la recuperación ordenada de datos last-in, first-out (LIFO), es decir, el último dato en entrar es el primer dato en salir. En esta estructura sólo se tiene acceso a la cabeza o cima de la pila.

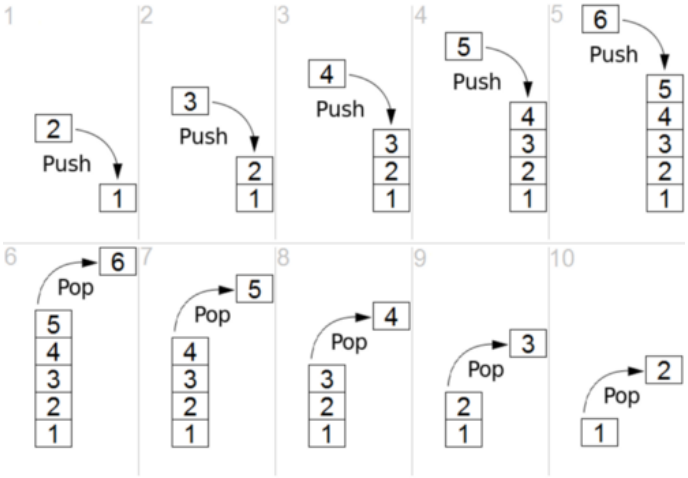


Figure 23: Pilas.

6.2 Colas:

Una queue o cola, soporta la recuperación ordenada de datos first-in, first-out (FIFO), es decir, el primer dato en entrar es el primer dato en salir. Sólo se

tiene acceso al final de la lista para meter elementos y al principio de esta para sacarlos.

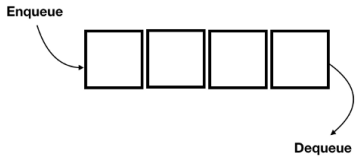


Figure 24: Colas.

6.3 Colas de prioridad

Son una colección de elementos donde cada elemento tiene asociado un valor de ordenación denominado prioridad.Se caracterizan por admitir inserciones de nuevos elementos, además de la consulta y la eliminación de elementos de menor prioridad.

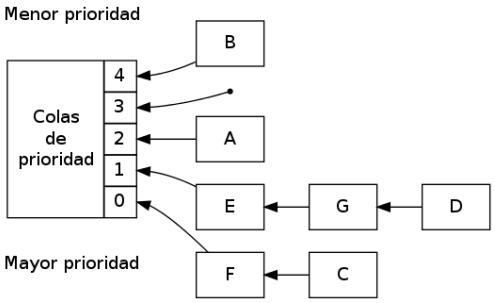


Figure 25: Colas de prioridad.