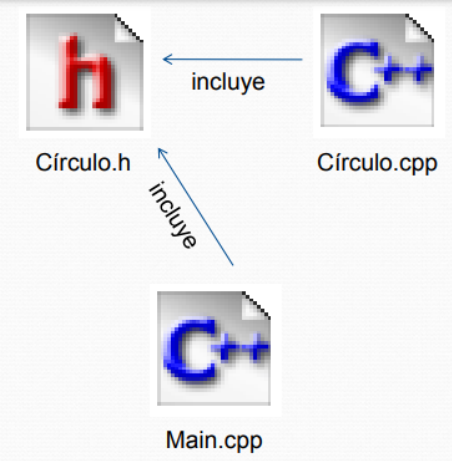
Ayed - (Algoritmos y Estructura de Datos)
Preparación para final - Mayo 2022

PROFESOR: NOMBRE DEL PROFESOR
no_reply@example.com

Declaración

- Interfaz de un módulo
- Definición del QUÉ
- Archivos .h (header)/archivos de cabecera:
    - Posibilita la división del código fuente en varios módulos.



Definición

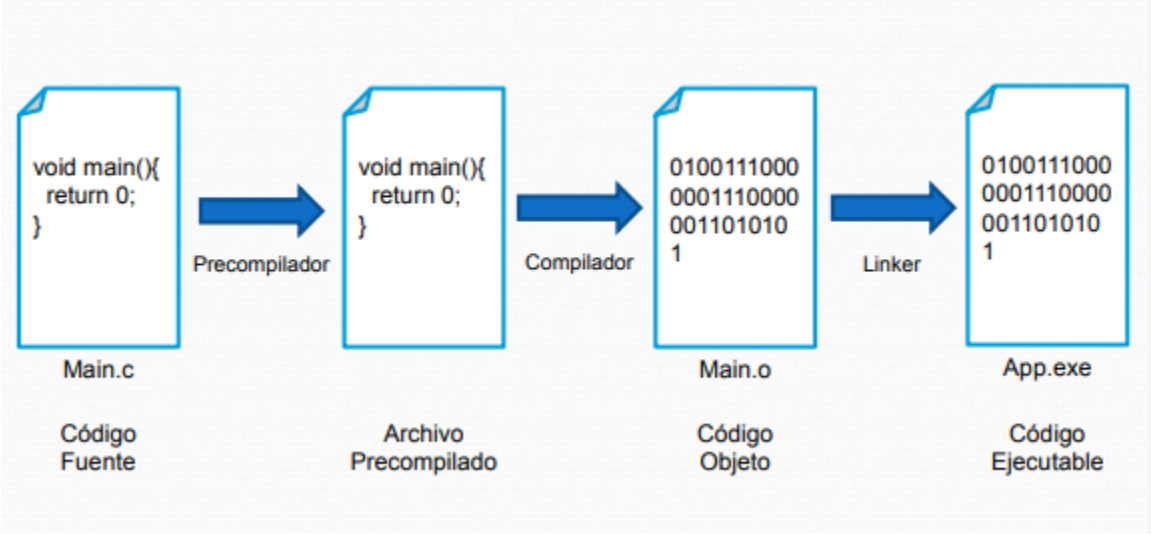
- Implementación de un módulo cumpliendo la interfaz declarada
- Definición del CÓMO
- Archivos .cpp (header)

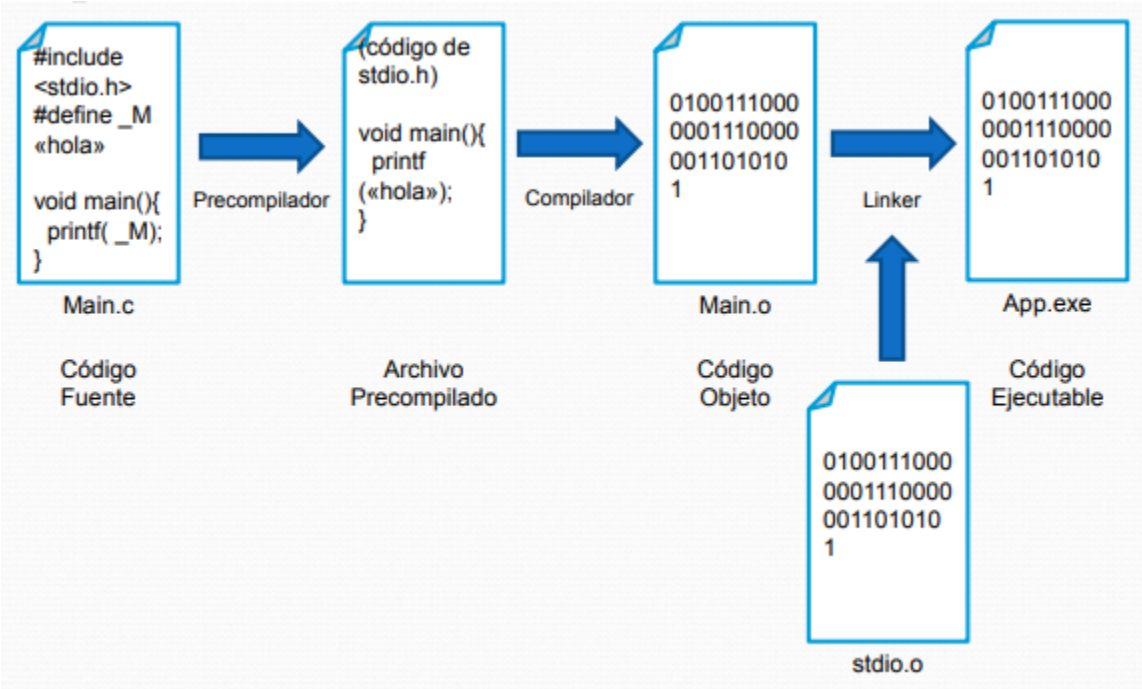
Proceso de compilación

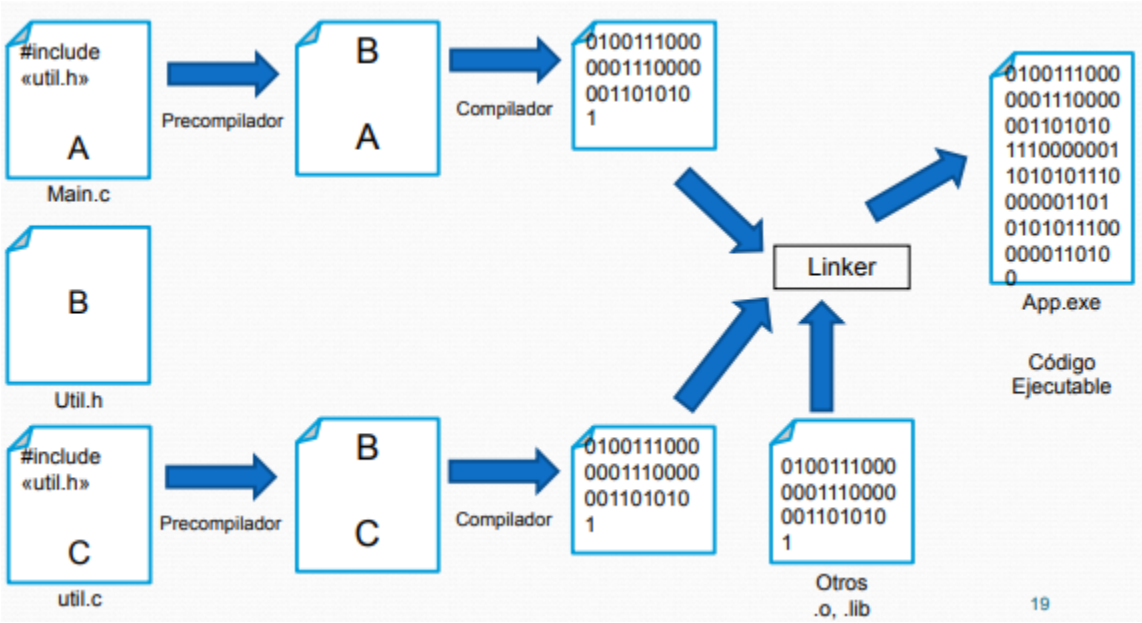
- El proceso de compilación comprende múltiples etapas:

- **<u>Pre procesamiento</u>**:
  - Reemplaza patrones existentes en el código fuente por otros patrones definidos por el programador.
  - Analiza directivas de precompilación:
    - Include
    - Define
    - Ifndef (útil para distintas plataformas, entornos, versiones...)
    - Ifdef
    - Endif
    - ...
- **<u>Compilación:</u>**
  - Un compilador traduce directamente el código fuente en instrucciones de máquina. (código objeto).
- **<u>Linkedicion</u>** :
  - es el responsable de combinar las diversas partes ya compiladas para conformar un programa ejecutable. (archivo .exe).

- La compilación independiente de las diversas partes de una aplicación es una característica fundamental.







Segmentos de Memoria

Code Segment (CD): localizará el código resultante de compilar nuestra aplicación, es decir, la algoritmia en Código Máquina.

Data Segment (DS): almacenará el contenido de las variables definidas en el módulo principal (variables globales).

Stack Segment (SS): almacenará el contenido de las variables locales en cada invocación de las funciones y/o procedimientos (incluyendo las del main en el caso de C/C++).

Extra Segment (ES) / Heap: está reservado para peticiones dinámicas de memoria.

Punteros

- Es un tipo de variable que **almacena direcciones de memoria**.
- Permiten el acceso a cualquier posición de la memoria, para lectura/escritura (en los casos en que esto sea posible).
- Permiten una manera alternativa de pasaje por referencia.
- Permiten solicitar memoria dinámica.
- Son el soporte de enlace que utilizan estructuras de datos dinámicas en memoria.

Operaciones válidas:

Asignación, Comparación, Incremento o decremento.

Referenciar

- Significa tomar la dirección de una variable existente (usando &) para establecer una variable de puntero.
- Para que sea válido, un puntero debe establecerse en la dirección de una variable del mismo tipo que el puntero, sin el asterisco:

```
int  c1;
int* p1;
c1 = 5;
p1 = &c1;
//p1 references c1
```

Desreferenciar

- Significa usar el operador * (carácter de asterisco) para acceder al valor almacenado en un puntero.

```
int n1;
n1 = (*p1);
```

Puntero genérico

- Es un puntero para el cuál no sabemos explícitamente el tipo de dato al que apunta.
- “Puede apuntar a cualquier tipo de dato”.
- Requieren el uso de casteos.
- Se declaran como void*.

Casteo

- Forzamos a ver una zona de la memoria como si fuera de un determinado tipo.
- Hay 3 tipos de casteos:
    - **Static**: verifica que los tipos de datos sean compatibles en tiempo de compilación
    - **Dynamic**: verifica que los tipos de datos sean compatibles en tiempo de ejecución.
    - **Reinterpret**: no verifica que los tipos de datos sean compatibles (es análogo al comportamiento con paréntesis).

Listas

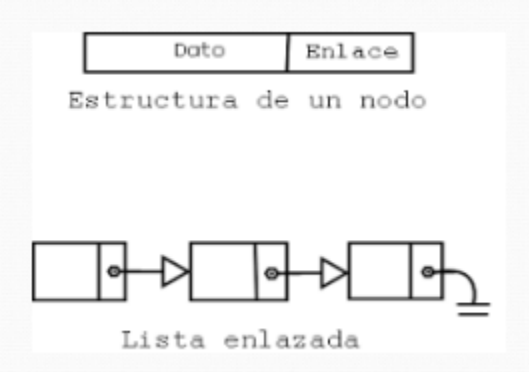
Listas enlazadas

Una lista enlazadas es entonces un grupo de datos organizados secuencialmente, pero a diferencia de los arreglos, la organización no está dada implícitamente por su posición en el arreglo. En una lista enlazada cada elemento es un nodo que contiene el dato y además un enlace al siguiente dato. Estos enlaces ligas son simplemente variables que contienen la(s) dirección(es) de los datos contiguos o relacionados.

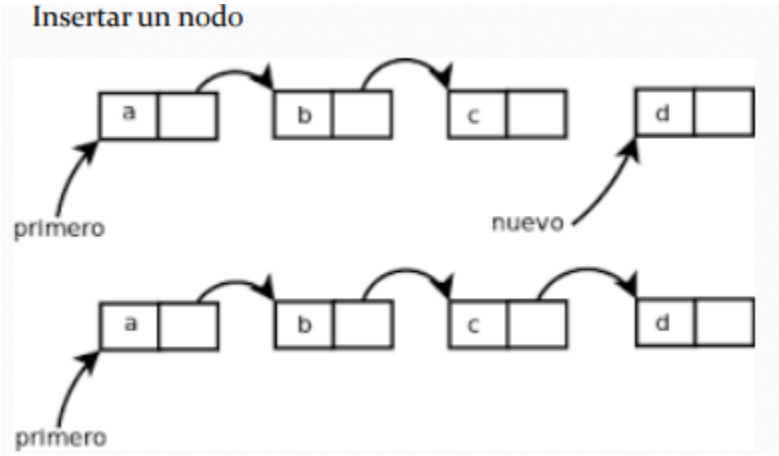
Para manejar una lista es necesario contar con un apuntador al primer elemento de la lista "head" . Las ventajas de las listas enlazadas son que:

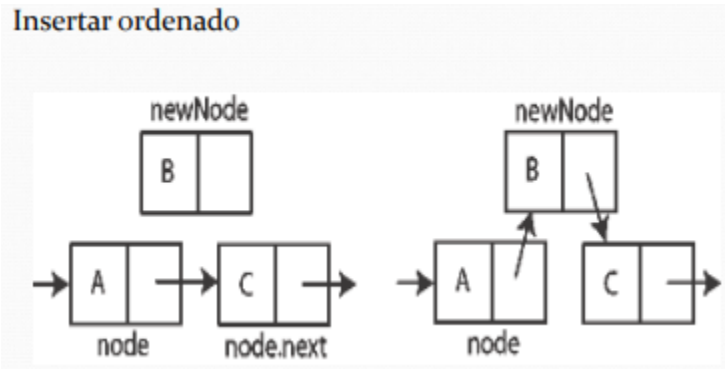
- Permiten que sus tamaños cambien durante la ejecución del programa.
- Proveen una major flexibilidad en el manejo de los datos.

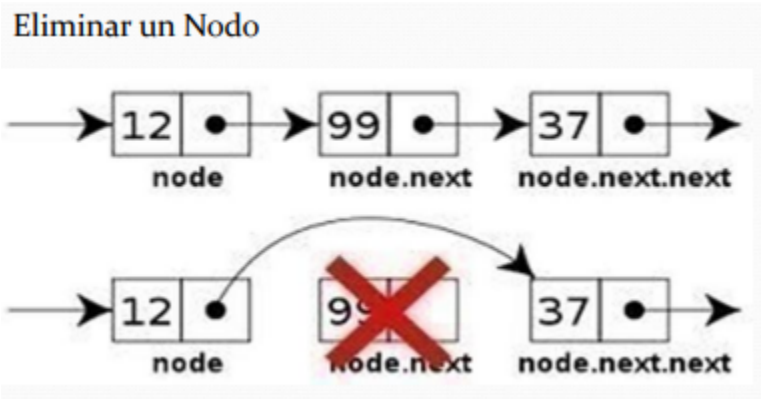
Este principio de listas enlazadas se puede aplicar a cualquiera de los conceptos de estructura de datos vistos anteriormente: arreglos, colas y pilas . Es decir, las operaciones de altas, bajas y cambios, así como búsquedas y ordenamientos se tendrán que adaptar en la cuestión del manejo de ubicaciones únicamente.



Listas simplemente enlazadas

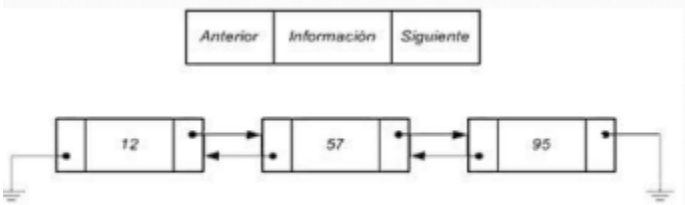




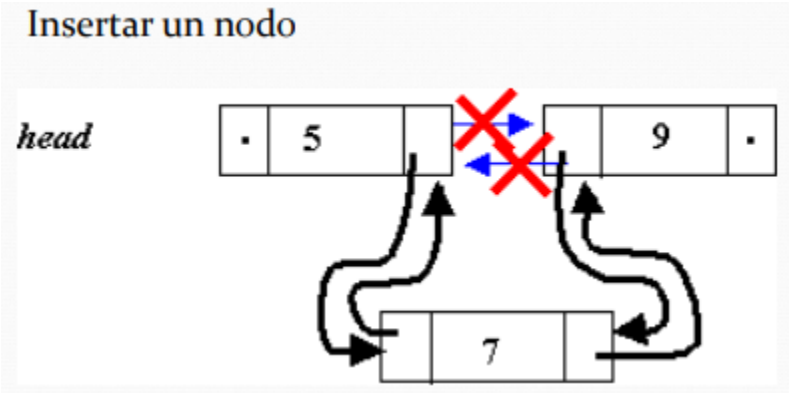


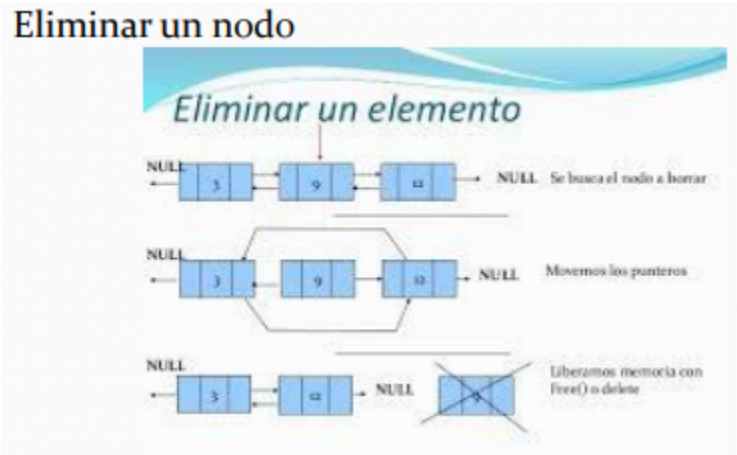
Listas Doblemente enlazadas

poder recorrer una lista en ambas direcciones. Para estos casos se tienen las listas doblemente enlazadas. Esta propiedad implica que cada nodo debe tener dos apuntadores, uno al nodo predecesor y otro al nodo sucesor.



Lista Doblemente Enlazada



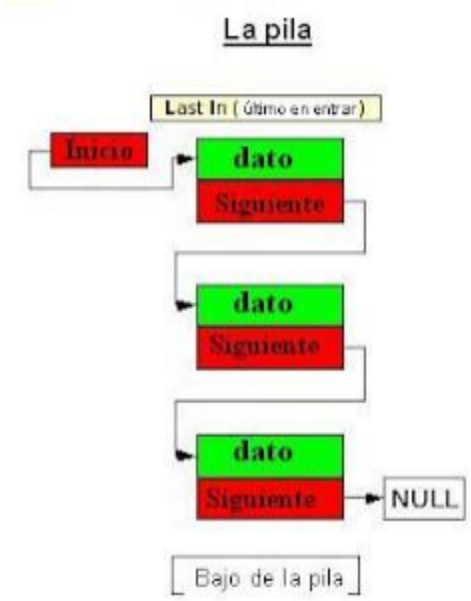


Pilas

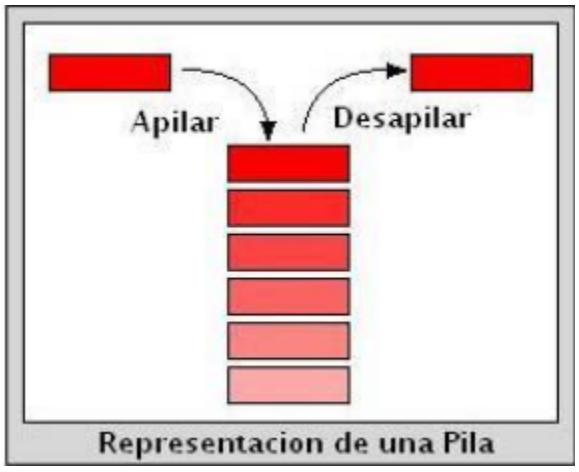
Una pila (stack en inglés) es una **lista ordinal o estructura de datos** en la que el **modo de acceso a sus elementos es de tipo LIFO** (del inglés Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos. Esta estructura se aplica en

multitud de ocasiones en el área de informática debido a su simplicidad y ordenación implícita de la propia estructura.

Para el manejo de los datos se cuenta con dos operaciones básicas: apilar (**push**), que coloca un objeto en la pila, y su operación inversa, retirar (o desapilar, **pop**), que retira el último elemento apilado.



Operaciones: Si hiciéramos analogía con objetos cotidianos, Por una operación apilar (Push) equivaldría a colocar un plato sobre una pila de platos, y una operación desapilar (Pop) a retirarlo.Si hiciéramos analogía con objetos cotidianos, Por una operación apilar (Push) equivaldría a colocar un plato sobre una pila de platos, y una operación desapilar (Pop) a retirarlo.



Cola

Una cola es un tipo especial de **lista abierta** en la que sólo se pueden **insertar nodos en uno de los extremos de la lista y sólo se pueden eliminar nodos en el otro**. Además, como sucede con las pilas, las escrituras de datos siempre son inserciones de nodos, y las lecturas siempre eliminan el nodo leído.

Este tipo de lista es conocido como **lista FIFO** (First In First Out), el primero en entrar es el primero en salir.

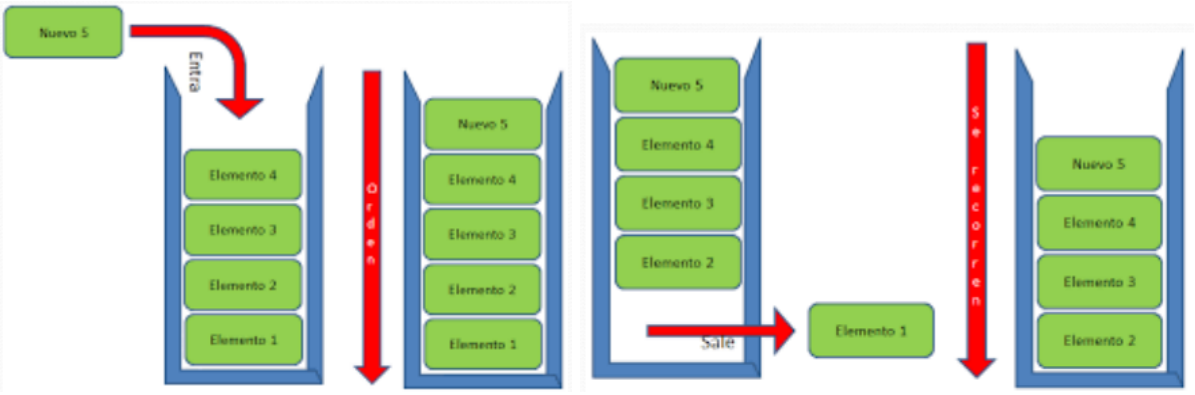
El símil cotidiano es una cola para comprar, por ejemplo, las entradas del cine. Los nuevos compradores sólo pueden colocarse al final de la cola, y sólo el primero de la cola puede comprar la entrada.

El nodo típico para construir pilas es el mismo que vimos en los capítulos anteriores para la construcción de listas y pilas.

Es evidente, que una cola es una lista abierta. Así que sigue siendo muy importante que nuestro programa nunca pierda el valor del puntero al primer elemento, igual que pasa con las listas abiertas. Además, debido al funcionamiento de las colas, también deberemos mantener un puntero para el último elemento de la cola, que será el punto donde insertemos nuevos nodos.

Teniendo en cuenta que las lecturas y escrituras en una cola se hacen siempre en extremos distintos, lo más fácil será insertar nodos por el final, a continuación del nodo

que no tiene nodo siguiente, y leerlos desde el principio, hay que recordar que leer un nodo implica eliminarlo de la cola.



Árboles:

Un árbol (tree) es un T.D.A. que consta de un conjunto finito T de nodos y una relación R (paternidad) entre los nodos tal que:

- Hay un nodo, especialmente designado, llamado la raíz del árbol T.
- Los nodos restantes, excluyendo la raíz, son particionados en m (m 0) conjuntos disjuntos T1, T2, ..., Tm, cada uno de los cuales es, a su vez, un árbol, llamado subárbol de la raíz del árbol T.
- A los nodos que no son raíces de otros subárboles se les denomina hojas del árbol T, o sea, no tienen sucesores o hijos.

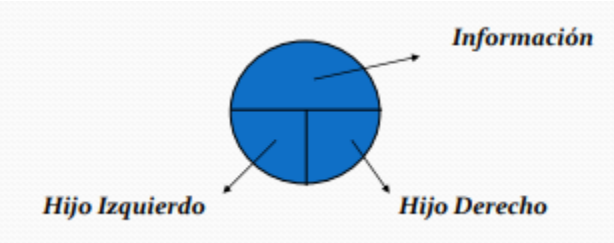
Definiciones:

- **Padre de un nodo:** al nodo raíz del subárbol más pequeño que contiene a dicho nodo y en el cual él no es raíz.
- **Hijo de un nodo:** al (los) nodo(s) raíz(ces) de uno de sus subárboles
- **Predecesor de un nodo:** al nodo que le antecede en un recorrido del árbol.
- **Hermano de un nodo:** a otro nodo hijo de su padre.
- **Grado de un nodo:** cantidad de hijos de un nodo
- **Grado de un árbol:** el mayor de los grados de todos sus nodos.
- **Nodo hoja:** es un nodo sin hijos, es decir, con grado = 0.
- **Nodo rama:** es un nodo que tiene hijos, o sea, a la raíz de un subárbol (grado > 0).

- **Nivel de un nodo:** al nivel de su padre más uno. Por definición, la raíz del árbol tiene nivel 0. Esta definición es recursiva.
- La **profundidad** de un árbol es el máximo nivel de cualquier hoja en el árbol. Esto es igual a la longitud de la trayectoria más larga de la raíz a cualquier hoja.

- **Árbol completo de nivel n:** Un árbol completo es un árbol de profundidad K que tiene todos los nodos posibles hasta el penúltimo nivel (profundidad K-1), y donde los elementos del último nivel están colocados de izquierda a derecha sin dejar huecos entre ellos.
- **Árbol lleno**: es un árbol con todos sus niveles llenos.
- Una **floresta** es una colección de dos o más árboles disjuntos.
    - Disjuntos significa que no hay nodos en común entre dos árboles cualesquiera de la misma.
    - De un árbol se obtiene una floresta al quitarle la raíz, si tiene dos hijos o más.
    - De una floresta se obtiene un árbol al añadir un nodo que sea raíz de todos los árboles que la conforman.
- **Árbol ordenado**: todo árbol para el que se considera el orden relativo de los sucesores o subárboles de cualquier nodo.
    - Es decir, en un árbol ordenado se habla de primero, segundo o último hijo de un nodo en particular. El primer hijo de un nodo de un árbol ordenado es denominado el hijo mayor de ese nodo y el último hijo es denominado el menor.
    - El Árbol es ordenado si al intercambiar el orden relativo de los subárboles de un nodo, representa una situación semánticamente diferente.
- **Árbol orientado** a un árbol para el cual no interesa el orden relativo de los sucesores o subárboles de cualquier nodo, ya que sólo se tiene en cuenta la orientación de los nodos.
    - Ejemplo: La estructura organizativa de una empresa, donde no es importante el orden de los subdirectores a la hora de representarlos en el árbol.
    - En la solución de problemas informáticos, los más utilizados son los árboles ordenados.
- Un **árbol binario** (en inglés binary tree) es un árbol ordenado de grado 2.

- Grado 2 significa que cada nodo tiene como máximo dos hijos, o sea, dos subárboles.
- Al ser ordenado el árbol, importa el orden de los subárboles, es decir, que será necesario especificar de cada nodo cuál es el hijo izquierdo y cuál el hijo derecho.
- Implementación:
  - Cada nodo del árbol binario contiene:
    - Una referencia a su información.
    - Un puntero a su hijo izquierdo.
    - Un puntero a su hijo derecho.



Aclaraciones:

- Si el conjunto finito T de nodos del árbol es vacío, entonces se trata de un árbol **vacío**.
- En esta estructura existe **sólo un nodo** sin padre, que es la **raíz** del árbol.
- Todo nodo, a excepción del nodo raíz, tiene **uno y sólo un padre**.
- Los subárboles de un nodo son llamados **hijos**.

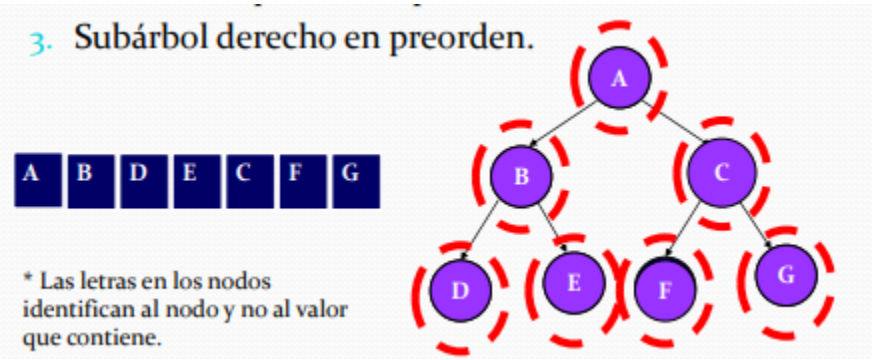
Recorridos de un Árbol

Los recorridos se clasifican de acuerdo al momento en que se visita la raíz del árbol y los subárboles izquierdo y derecho.

Recorrido en PreOrden

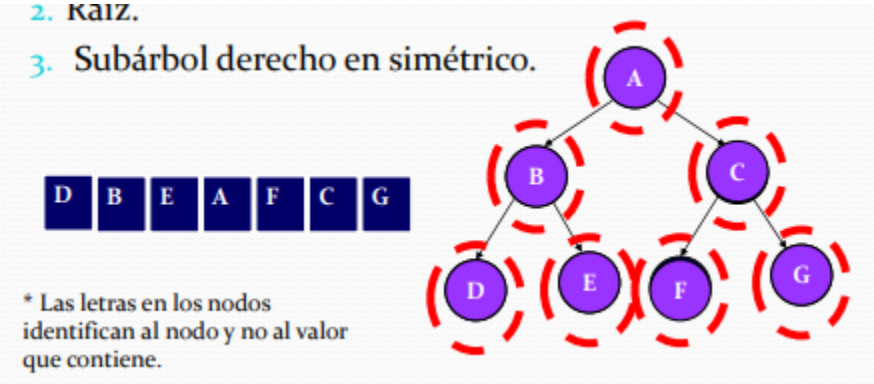
1. Visitar la raíz.
2. Recorrer subárbol izquierdo en preOrden.
3. Recorrer subárbol derecho en preOrden.

- Ejemplo Uso: Se va a utilizar siempre que queramos comprobar alguna propiedad del árbol (p.ej.: localizar elementos).



Recorrido en orden simétrico o inOrden

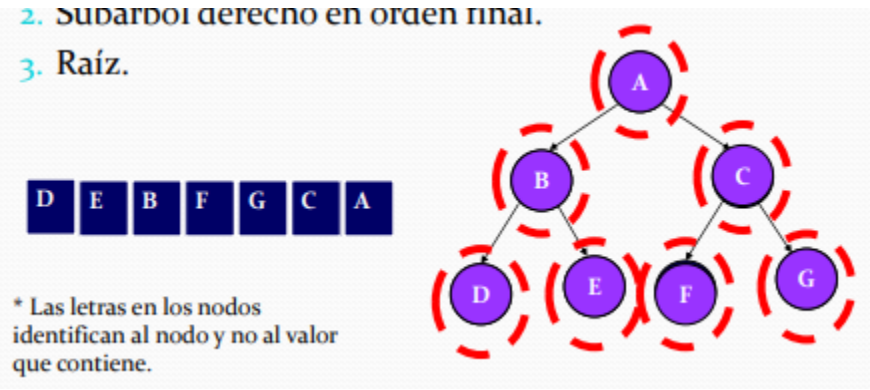
1. Recorrer subárbol izquierdo en simétrico.
2. Visitar la raíz.
3. Recorrer subárbol derecho en simétrico.
    - Ejemplo Uso: Se utiliza siempre que nos pidan algo referido a la posición relativa de las claves o algo que tenga que ver con el orden de las claves (p.ej.: ¿Cuál es la 3ª clave?).



Recorrido en orden final o PostOrden

1. Recorrer subárbol izquierdo en orden final.
2. Recorrer subárbol derecho en orden final.
3. Visitar la raíz.

- Ejemplo Uso: Su principal utilidad consiste en liberar la memoria ocupada por un árbol.



Árbol de búsqueda

Permiten realizar operaciones (recorridos, búsqueda de un elemento, etc) de forma más eficiente.

Hay dos momentos para la manipulación de un árbol:

- La construcción del árbol.
- El recorrido del árbol para realizar las operaciones requeridas según el problema a resolver.

Existen dos tipos especiales de árboles:

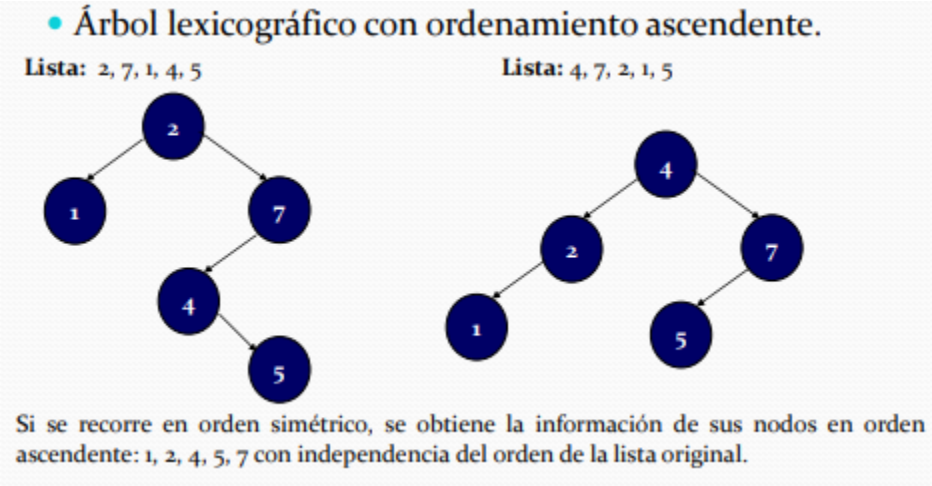
- **Árboles lexicográficos:** es un árbol binario que, recorrido en orden simétrico, permite obtener la información de los nodos en algún criterio de ordenamiento.

  La técnica de construcción de un árbol lexicográfico consiste en un proceso recursivo que va colocando los nodos en el subárbol izquierdo o derecho del nodo raíz, según sea el criterio de ordenamiento deseado (ascendente o descendente).

  Siguiendo un ordenamiento ascendente:

  - Se compara el nodo que se quiere insertar con la raíz del árbol.

- Si es menor, se coloca en el subárbol izquierdo siguiendo el mismo proceso.
- Si es mayor, se coloca en el subárbol derecho siguiendo el mismo proceso.

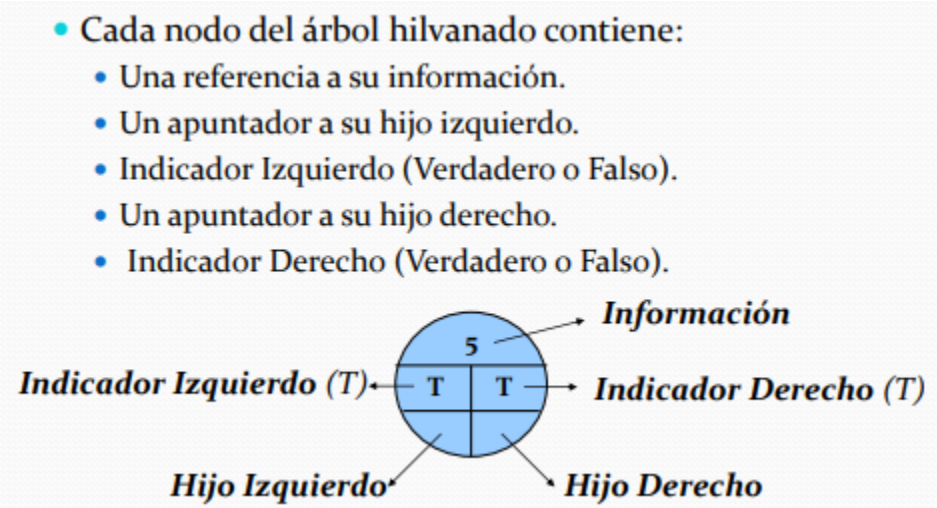


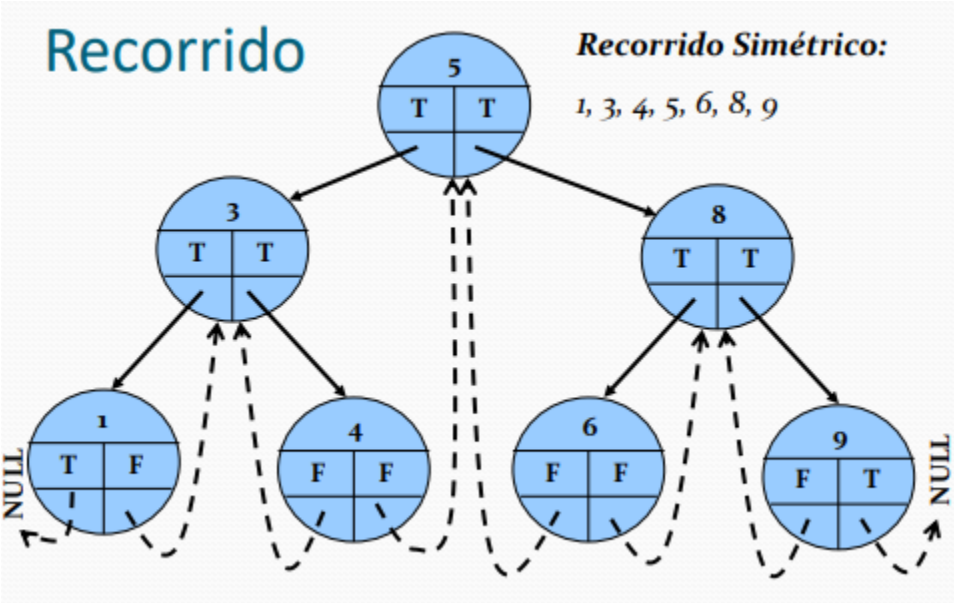
El recorrido de árboles con programas recursivos resulta costoso ya que implica un gasto adicional de memoria y tiempo de ejecución. Para árboles muy grandes se puede desbordar el stack del sistema relativamente pronto.

→ árbol hilvanado

- **Árboles hilvanados:** (o árbol entrelazado) es un árbol binario en el que cada hijo izquierdo de valor nulo es sustituido por un enlace o hilván al nodo que le antecede en orden simétrico (excepto el primer nodo en orden simétrico) y cada hijo derecho de valor nulo es sustituido por un enlace o hilván al nodo que le sigue en el recorrido en orden simétrico (excepto el último nodo en orden simétrico).

  Ahora, un recorrido en orden simétrico se puede implementar sin necesidad de recursión. Sin embargo, se requiere que los nodos tengan en su estructura algún atributo que permita saber cuándo un enlace es real y cuándo se trata de un hilván. En este caso es necesario un atributo para cada hijo.





- **Árbol Balanceado:** La búsqueda en un árbol puede convertirse en una búsqueda secuencial. Esto sucede porque el árbol no está balanceado, es decir los nodos no están distribuidos uniformemente y se han insertado todos los nodos en profundidad.

  Esto podría ser salvado si se utilizara un árbol balanceado que al insertar toma en cuenta la cantidad de niveles del árbol y distribuye los nodos uniformemente.

- **Árbol AVL:** es un árbol binario de búsqueda en el que las alturas de los subárboles izquierdo y derecho de cualquier nodo se diferencian a lo sumo en uno.

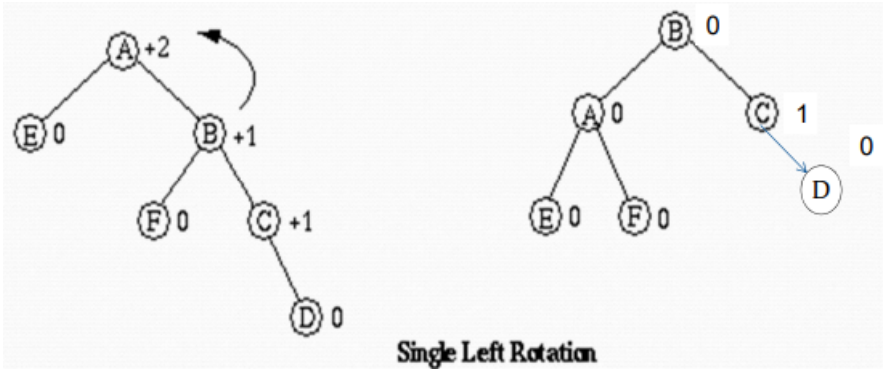
La búsqueda es similar a como se hace en un árbol binario de búsqueda (lexicográficos), pero la inserción y la eliminación deben considerar la propiedad del balance.

Es un Árbol Binario de Búsqueda (ABB) que siempre debe estar equilibrado. Es decir, para todo nodo la altura de sus subárboles izquierdo y derecho pueden diferir a lo sumo en 1.

Cada nodo tiene asignado un peso de acuerdo a las alturas de sus subárboles (1 si su subárbol derecho es más alto, -1 si su subárbol izquierdo es más alto y 0 si las alturas son las mismas)-

Operaciones sobre un AVL

- **Insertar**
  - La técnica es la misma que en un ABB
  - Se inserta el nodo como un nodo hoja en el lugar correspondiente.
  - Se sube hasta la raíz actualizando el equilibrio de cada nodo.
  - Si el equilibrio de un nodo pasa a ser + - 2, se realizan rotaciones para balancear el árbol
- **Balancear**
  - **<u>Caso 1 Rotación simple Izquierda RSI</u>**
    - Si esta desequilibrado a la derecha y su hijo derecho tiene el mismo signo (+) hacemos rotación simple izquierda.



    - Cuando: n->fe = +1+1, n1->fe >= 0

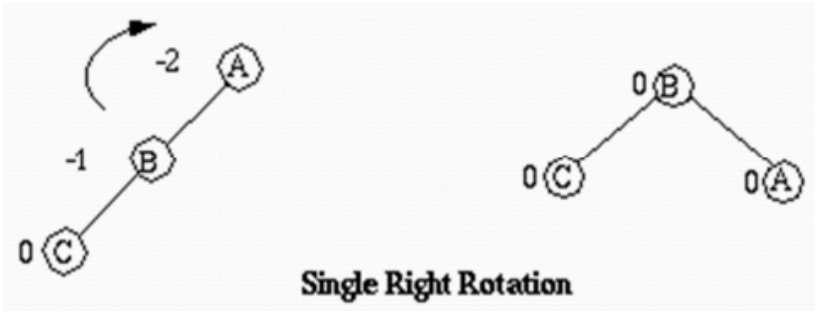
- Código

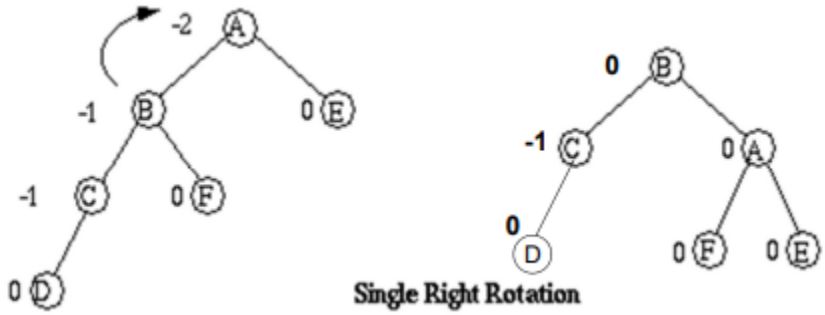
  n->der = n1->izq

  n1->izq = n

  n = n1

- **Caso 2 Rotación simple Derecha RSD :**
  - Si está desequilibrado a la izquierda y su hijo izquierdo tiene el mismo signo (-) hacemos rotación simple derecha.





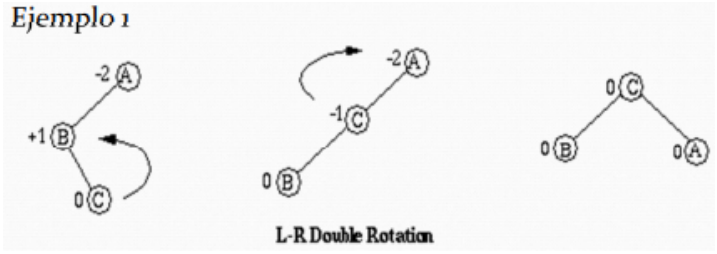
  - Cuando: n->fe = -1-1, n1->fe <= 0
  - Código

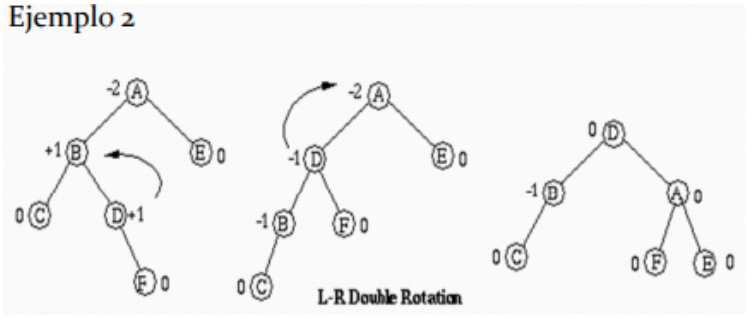
    n->izq = n1->der

    n1->der = n

    n = n1

- **Caso 3 Rotación doble Izquierda Derecha RDID :**
  - Si está desequilibrado a la izquierda (FE < –1), y su hijo izquierdo tiene distinto signo (+) hacemos rotación doble izquierda-derecha.





  - Cuando: n->fe = -1-1, n1->fe > 0
  - Código

    n1->der = n2->izq

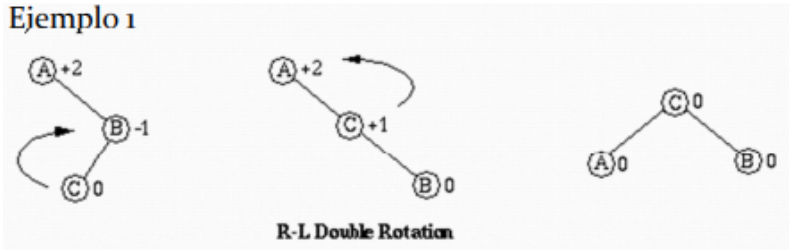
    n2->izq = n1

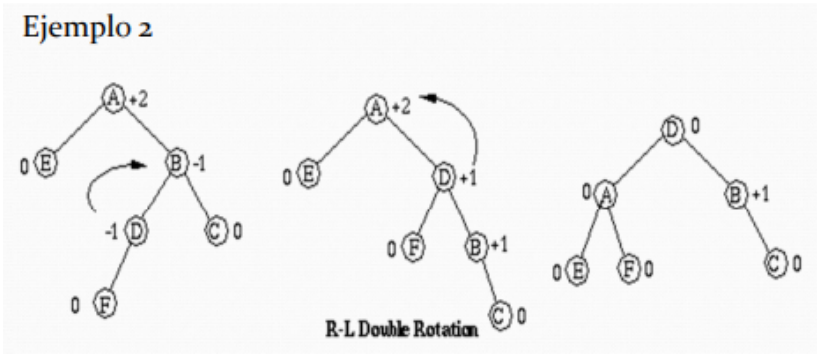
    n->izq = n2->der

    n2->der = n

    n = n2

- **Caso 4 Rotación doble Derecha Izquierda RDDI :**
  - Si está desequilibrado a la derecha y su hijo derecho tiene distinto signo (–) hacemos rotación doble derecha-izquierda.





- Cuando: n->fe = +1+1, n1->fe < 0
- Código:

  n1->izq= n2->der

  n2->der = n1

  n->der = n2->izq

  n2->izq = n

  n = n2

- **Eliminar**
  - Se elimina igual al ABB:
    - Si el nodo es un nodo hoja, simplemente lo eliminamos.
    - Si el nodo sólo tiene un hijo, lo sustituimos con su hijo.
    - Si el nodo eliminado tiene dos hijos, lo sustituimos por el hijo derecho y colocamos el hijo izquierdo en el subárbol izquierdo del hijo derecho.

- Al eliminar un nodo en un árbol AVL puede afectar el equilibrio de sus nodos. Entonces hay que hacer rotaciones simples o dobles.

- **Calcular Altura**

Equilibrio

Equilibrio (n) = altura-der (n) – altura-izq (n)

describe relatividad entre subárbol der y subárbol izq.

- + (positivo) → der mas alto (profundo)
- - (negativo) → izq mas alto (profundo)

Un árbol binario es un AVL si y sólo si cada uno de sus nodos tiene un equilibrio de –1, 0, + 1

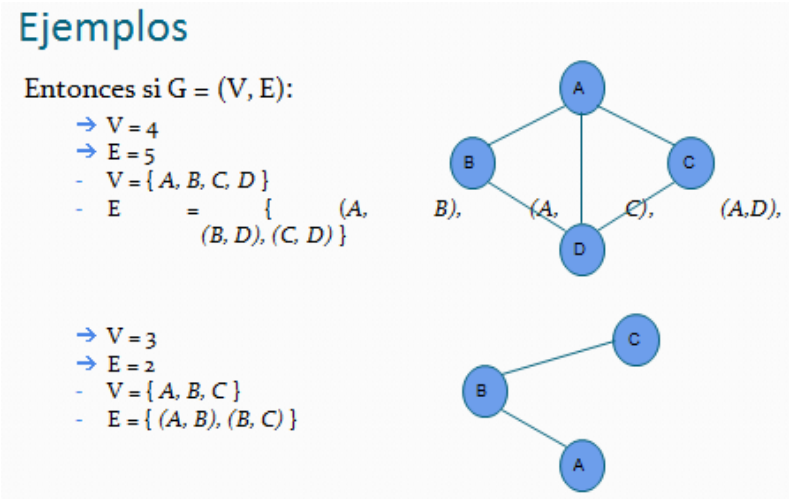
Si alguno de los pesos de los nodos se modifica en un valor no válido (2 o -2) debe seguirse un esquema de rotación.

```
Struct Nodo {
  int fe;  //para almacenar el valor del equilibrio del nodo
  void* ptrDato;
  Nodo* ptrIzq;
  Nodo* ptrDer;
}
```

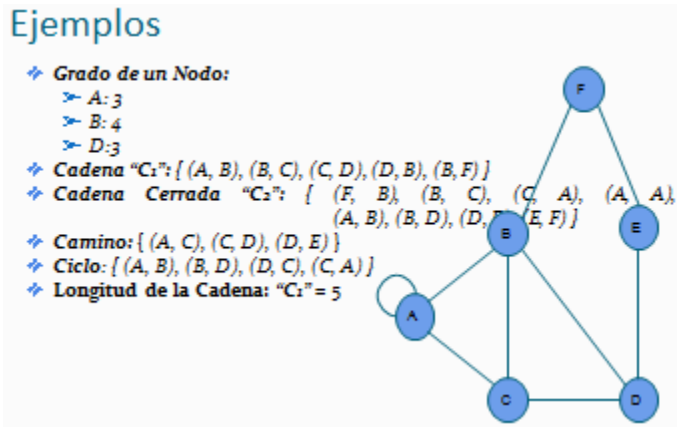
Gráfos

Un grafo es una estructura de datos que consta de un conjunto finito y no vacío de nodos “V” (“vértices”) y unas relaciones “E”(aristas) entre los nodos que lo componen

Sirven para estudiar tanto la estructura de internet, como una red de autopistas, la red de amigos de Facebook o hasta como interactúan las partículas elementales.



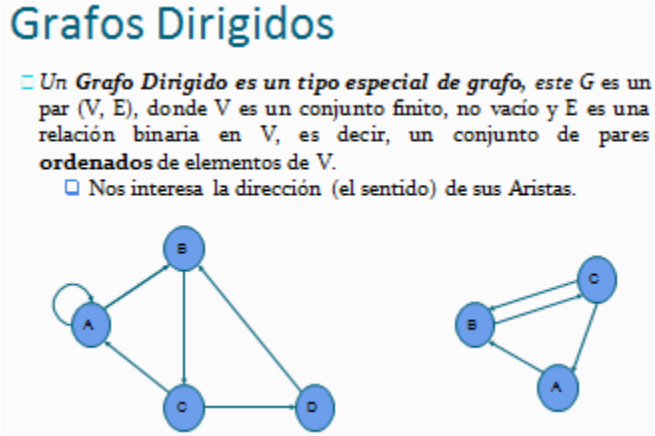
- **Grado de un nodo:** Número de conexiones/relaciones que posee un nodo.
- **Cadena:** a toda la sucesión finita alterna de Vértices (v) y Aristas (E).
- **Cadena Cerrada:** cadena en la que (V) inicial y final coinciden.
- **Camino:** cadena en la que no se repiten ni sus (V) ni sus(E).
- **Ciclo:** cadena en la que no se repiten ni sus (V) ni sus (E), a excepción del (V) inicial y final.
- **Longitud de la cadena:** Número de Aristas(E) que forman una cadena.



- **<u>Grafo Dirigido:</u>**

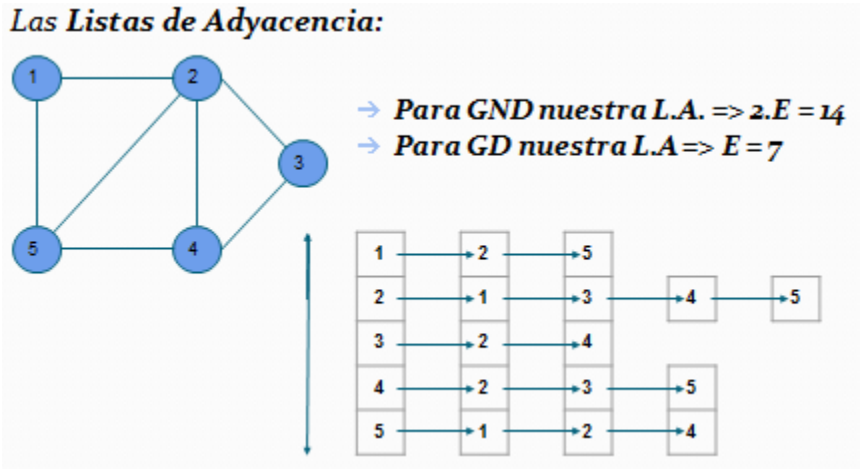
Un grafo dirigido es un tipo especial de grafo, este es un par (V, E), donde V es un conjunto finito, no vacío y E es una relación binaria en V, es decir un conjunto ordenados de elementos de V.

- Nos interesa la dirección de sus aristas.



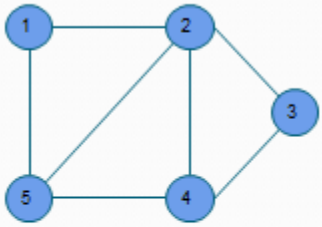
<u>Representaciones:</u>

- **<u>Listas de Adyacencia</u>:** son útiles cuando el número de (V) son muy superior al número de (E).



- **<u>Matriz de Adyacencia</u>:** Van bien cuando buscamos conectividad rápida entre los nodos.

*Una **Matriz de Adyacencia:***



→ 0 si ∄ V(i,j)
→ 1 si ∃ V(i, j)

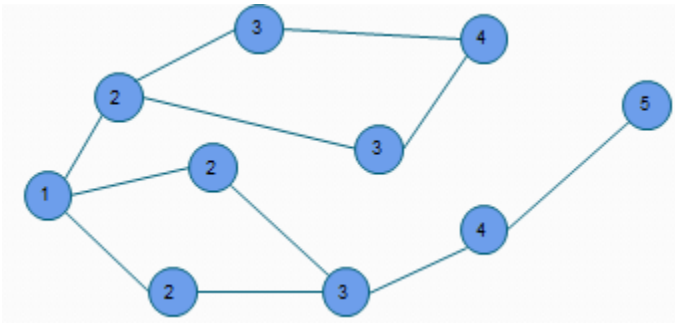
- **N[i][j] == 1**
- **Coste = O(1)**

\	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Recorrido básicos de Grafos

La operación de recorrer una estructura de datos consiste en visitar cada uno de los nodos a partir de uno dado. Así, para recorrer un árbol se parte del nodo raíz y según el orden se visitan todos los nodos. De igual forma, recorrer un grafo consiste en visitar todos los vértices alcanzables a partir de uno dado.

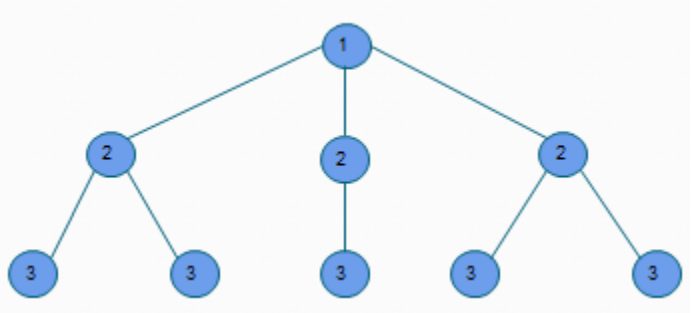
- **<u>Recorrido en Anchura (BFS - Breadth First Search)</u>:**
    - Comienza la búsqueda con los nodos adyacentes o los vecinos directos adhiriéndolos en una cola de Vértices pendientes de Visitar.
    - Es simple y es una de las bases en las que se basa el algoritmo de Prim (para encontrar el árbol mínimo) y Dijkstra (para encontrar los caminos mínimos en un grafo dirigido ponderado).



Breadth-First Search Algorithm:

```
BFS (G, s) //Where G is the graph and s is the source node
let Q be queue
Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked
mark s as visited.
while ( Q is not empty)
     //Removing that vertex from queue,whose neighbour will be visited now
          v = Q.dequeue( )
     //processing all the neighbours of v
          for all neighbours w of v in Graph G
                    if w is not visited
                    Q.enqueue( w ) //Stores w in Q to further visit its neighbour
                    mark w as visited.
```

- **<u>Recorrido en Profundidad (DFS - Depth First Search)</u>:**
    - Admite formularse recursivamente, visitando todos los nodos hasta llegar a un callejón sin salida y, reiniciar el proceso de búsqueda.
    - Es importante marcar los nodos como visitados en el orden que se visitan, y luego continuar con la recursividad de los nodos adyacentes.



Depth-First Search Algorithm:

```
DFS-iterative(G, s): //Where G is graph and s is source vertex
      let S be stack
      S.push(s)  //Inserting s in stack
      mark s as visited
      while(S is not empty):
            //Pop a vertex from stack to visit next
            v  =  S.top( )
            S.pop( )
            //Push all the neighbours of v in stack that are not visited
            for all neighbours w of v in Graph G:
                  if w is not visited :
                        S.push(w)
                              mark w as visited

DFS-recursive(G, s):
                  mark s as visited
                  for all neighbours w of s in Graph G:
                              if w is not visited:
                                          DFS-recursive(G, w)
```

¿Qué es un Algoritmo?

Es un conjunto ordenado de instrucciones bien definidas, no ambiguas y finitas que permite resolver un determinado problema computacional:

- Existen infinitos algoritmos para llegar a la solución pero hay algoritmos más eficientes que otros. Por eso, es necesario no quedarse satisfecho con la1era solución. Siempre pensar si se puede hacer mejor. Porque la 1era solución generalmente no es la mejor.
-