apuntes

estructuras de datos.

Un **arbol** es una estructura de datos recursiva en el que cada elemento se organiza de manera jerárquica en forma de nodos conectados entre sí. Esta estructura es considerada **recursiva** porque un arbol esta compuesto de subárboles, es decir a su vez también son árboles, y esta organización puede crecer indefinidamente.

dentro de un árbol, 3 tipos de nodos:

- **nodo raíz**: punto de partida del arbol (no tiene padre)
- **nodos intermedios**: que tiene padre y de menos un hijo.
- **nodos hoja**: son los que no tienen hijos.

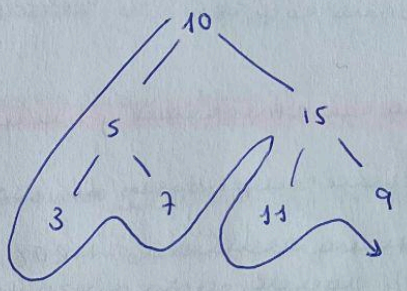
una caracteristica clave de los arboles es que podemos realizar sobre ellos distintos tipos de recorridos, que permiten visitar y procesar los nodos en diferentes órdenes.

los tres recorridos más comunes son:

- **recorrido en preorden**: se procesa primero el nodo actual, luego el subarbol izquierdo y finalmente el derecho.
- **recorrido en orden central**: primero el subarbol izquierdo luego el nodo actual y después el subárbol derecho.
- **recorrido en post orden**: primero se recorren los subárboles (izquierdo y derecho) y se hace el nodo actual.

estos recorridos permiten explorar o mostrar el contenido del arbol de distintas maneras según el objetivo del programa

ejemplo:



→ **recorrido preorden**:

```
void Preorden (N (nodo)){
   if (N!= null){
         System.out.println (N.dato);
         Preorden (N.setMenor ());
         Preorden (N.setMayor ());

   }
}
```

si yo ejecuto este recorrido el resultado de mi arbol saldrá: en preorden voy visitando cada uno de los nodos y según visito hago por pantalla lo que contiene ese nodo

10, 5, 3, 7, 15, 11, 19.

→ Recorrido en orden central.

```
Void Central (N){
   if (N! = null){
        central (N.getMenor())
        System.out.println(N.dato);
        central(N.getMayor());
   }
}
```

entramos a un nodo visitamos a su menor, cuando ya hemos visitado su menor y visitamos sus mayores.

3, 5, 6, 10, 11, 15, 19.

→ recorrido en postorden

```
Void Postorden (N){
   if (N! = null){
      Postorden (N.getMenor());
      Postorden (N.getMayor());
      System.out.println (N.dato);
```

cuando abandonamos un nodo pasa por pantalla cuando ya no hay ni menor ni mayor:

3, 7, 5, 11, 19, 15, 10.

el postorden se da primero los operandos si los nodos padres fueran +, x y las hojas los operandos. Sirve cuando para hacer una expresión matemática de forma lógica.

¿Que tienen en común un coche, una construcción y un arbol?

todas las estructuras jerarquicas están compuestas por partes más pequeñas.
en la industria se utiliza un sistema llamado MRP2 (planificación de recursos de manufactura), donde cada pieza tiene asociado

- un coste
- un tiempo.
- un esfuerzo.

y el objetivo es optimizar todo eso: construir más rápido, más barato, y con mejor logística.

→ ¿para qué sirve el recorrido preorden?

el recorrido preorden primero visita el nodo padre antes que sus hijos.
esto es útil cuando quiero sacar primero los datos importantes, o donde necesito actuar antes de ir a los detalles.

→ ¿cómo se borra un nodo de un arbol binario?

borrar un nodo no es tan simple como eliminarlo y ya. Hay que reacomodar el árbol correctamente. vamos por casos.

CASO 1. el nodo a borrar no tiene hijos (es una hoja)
solo lo desconectas de su padre

```
if (N.padre.getMenor() == N) {
    N.padre.setMenor(null);
} else {
    N.padre.setMayor(null);
}
```

CASO 2. el nodo a borrar tiene un solo hijo.
. se reemplaza el nodo con su único hijo.
. también se actualiza el padre del hijo para mantener la conexión correcta.

```
if (N.getMenor() != null)
    hijo = N.getMenor();
else
    hijo = N.getMayor();

if (N.padre.getMenor() == N)
    N.padre.setMenor(hijo);
else
    N.padre.setMayor(hijo);

hijo.setPadre(N.padre);
```

si no hacemos esto, el arbol puede quedar mal estructurado ("nodo fantasma").

CASO 3 el nodo a borrar tiene dos hijos.
aquí es más complejo, porque necesitas buscar un reemplazo adecuado para el nodo que vas a eliminar.

2 opciones { busca el mayor de los hijos izquierdos.
busca el menor de los hijos derechos.

¿cómo se hace?

vi elijes el camino de la izquierda: vas a la izquierda y todo a la derecha hasta llegar a una hoja.
este nodo será el reemplazo del que vas a borrar.

¿qué pasa si los nodos tienen referencias al padre?

esto te da más flexibilidad, porque puedes subir por el árbol y ubicarte mejor.
para también cumplir con las operaciones: altura al borrar, también debes asegurarte de que el campo padre en los hijos esté actualizado correctamente.

ÁRBOL AVL

es un árbol binario de búsqueda balanceado.
cada vez que haces inserciones o eliminaciones, el árbol se reajusta automáticamente para mantener el equilibrio.
esto es importante porque si el árbol se desequilibra mucho, se comporta como una lista y pierde eficiencia.