**Árboles**

**Def. no recursiva ->** Un árbol está compuesto por un conjunto de nodos y un conjunto de aristas dirigidas que conectan parejas de nodos.

**Propiedades de un árbol con raíz:**

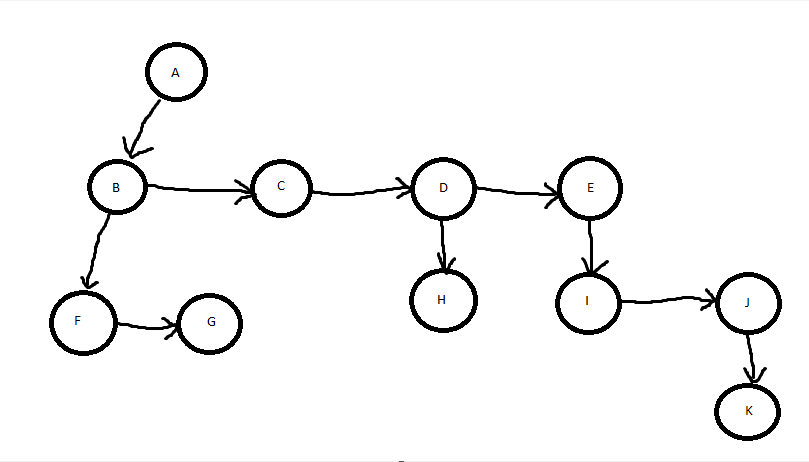
* Uno de los nodos se distingue al ser raíz,
* Todo nodo **c** (excepto raíz) está conectado mediante una arista a un único nodo **p**. Nodo **p** padre de **c** y **c** el único hijo de **p**,
* Existe un solo camino de la raíz a cada nodo, el número de aristas que hay que recorrer es la longitud del camino,
* Un árbol con **n** nodos tiene **n – 1** aristas,
* La profundidad de un nodo es la longitud del camino de la raíz hasta ese nodo,
* La altura de un nodo es la longitud del camino que va desde la hoja más profunda hasta el nodo. La altura del árbol es la de la raíz,
* Los nodos del mismo padre son hermanos.

Si existe un camino desde el nodo **u** al **v**, entonces **u** es ancestro de **v** y **v** descendiente de **u**, si **u != v** entonces **u** es ancestro propio de **v** y **v** descendiente propio de **u**.

El tamaño de un nodo es el número de descendientes, incluyendo el propio nodo.

**Def. recursiva ->** Un árbol está vacío o está compuesto de una raíz y cero o más subárboles no vacíos T1, T2, …, Tk cuyas raíces están conectadas mediante una arista.

Método del primer hijo / siguiente hermano: Se trata de que cada nodo mantiene una referencia del hermano a su derecha y de su primer hijo a la izquierda.



Nodo interno -> tiene por lo menos un hijo.

Nivel = profundidad.

**Árbol binario**

Es un árbol en el que ningún nodo puede tener más de 2 hijos. Recursivamente o está vacío o está compuesto de una raíz, árbol izquierdo y derecho.

Una implementación es el árbol de expresión, las hojas son operandos (constantes o nombres de variables), los restantes nodos contienen operadores.

Diagrama

Descripción generada automáticamente

Otra implementación es el árbol de codificación de Huffman, tiene el alfabeto en sus hojas y los demás nodos tienen izquierdo 0 y derecho 1. (b = 100).

Otra implementación son los árboles de búsqueda binaria, permitiendo insertar y acceder a elementos en tiempo logarítmico.

Diagrama

Descripción generada automáticamente

**Aplicaciones:**

* Expresiones aritméticas,
* Procesos de decisión,
* Búsqueda.

**Métodos de recorrido:**

* Recorrido en preorden -> primero se coloca el nodo y luego se procesan recursivamente sus hijos. (Ejemplo Duplicate).
* Recorrido en postorden -> el nodo se procesa después de haber procesado ambos hijos. (Ejemplo size y height)
* Recorrido en orden -> se procesa recursivamente el nodo izquierdo, el nodo actual y después el derecho.

Tiempo de ejecución lineal para cada algoritmo.

No hay diferencia de velocidad entre algoritmo recursivo y no recursivo.

**Recorrido Postorden ->** Se implementa utilizando una pila para almacenar el estado actual (1- llamada recursiva árbol izquierdo, 2 – llamada recursiva árbol derecho, 3 – al nodo actual). Cada nodo será insertado 3 veces en la pila en consecuencia.

El nodo será extraído por primera o segunda vez, no estará listo para ser visitado, así que se vuelve a introducir en la pila. La tercera vez extraído el nodo se declara como visitado.

Diagrama

Descripción generada automáticamente

**Árbol binario completo -> Todos los nodos tienen al menos 2 hijos.**

**Los hijos de un nodo son un par ordenado.**

**Recorrido en orden ->** Se declara igual que postorden, salvo porque un nodo se declara como visitado después de ser extraído por segunda vez. Después repite el proceso con el hijo derecho.

Diagrama

Descripción generada automáticamente con confianza media

**Recorrido Preorden ->** Es igual al en orden salvo porque un nodo se declara como visitado al ser extraído por primera vez. Se introduce el hijo derecho y luego el izquierdo.

Diagrama

Descripción generada automáticamente

**Árbol búsqueda binaria**

Se dice que un árbol binario es de búsqueda si está organizado de forma que:

* Para cada nodo ti todas las claves del subárbol izquierdo de ti son menores que la clave de ti y todas las claves del árbol derecho son mayores.

Si el árbol tiene n nodos y está balanceado, su altura será de log(n).

Una búsqueda en este caso puede tomar log(n) comparaciones o menos.

**Operaciones con un árbol de búsqueda binaria:**

find -> comenzando por la raíz y bifurcándose a la izquierda o derecha, dependiendo el resultado de una operación.

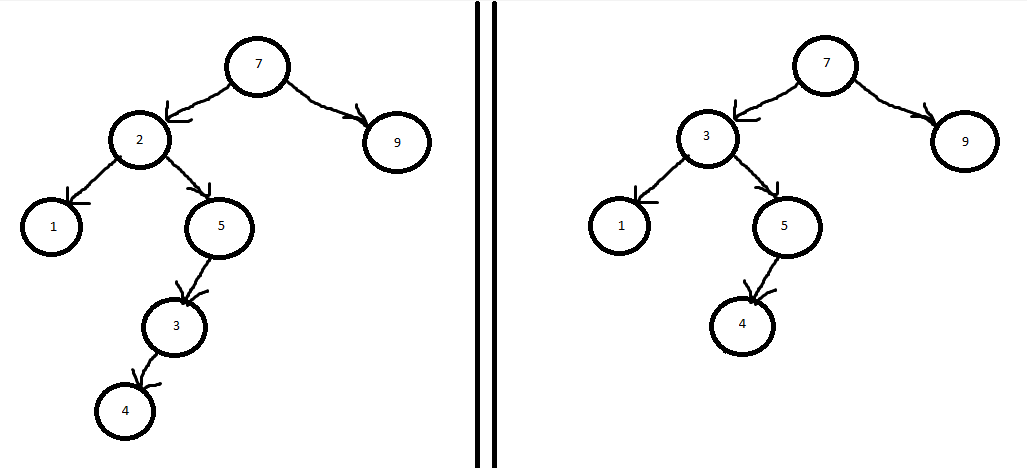
findMin -> comenzamos por la raíz y vamos bifurcándonos a la izquierda hasta que deje de haber un nodo hijo izquierdo.

finMax -> igual a finMin solo que se bifurca hacia a la derecha.

Coste logarítmico pero lineal en el peor caso (la profundidad del árbol afecta el tiempo de ejecución de los algoritmos).

remove -> si el nodo es una hoja, se elimina sin problemas. Si el nodo tiene un hijo, se elimina después de ajustar el enlace entre el padre y el hijo. (Por esto removeMin y removeMax son fáciles de implementar al igual que las raíces que no tienen padre).

Si el nodo tiene dos hijos, reemplazar el nodo por el elemento más pequeño del subárbol derecho (te aseguras que no tenga hijo izquierdo).



**Árbol AVL (**primer árbol de búsqueda binaria equilibrado)

Se trata de árbol de búsqueda binaria que tiene condición adicional de equilibrio (la cual es fácil de mantener y garantiza que la profundidad del árbol sea O(log(n)).

La condición dice que para cualquier nodo del árbol, las alturas de los subárboles izquierdo y derecho se diferencian de 1. (Altura subárbol vacío: -1). La condición indica que el árbol tiene solo una profundidad logarítmica. Todas las operaciones de búsqueda de un árbol AVL son logarítmicas.

Reequilibrar un árbol para satisfacer propiedad AVL:

Cuando el hijo izquierdo / derecho, del subárbol izquierdo / derecho de un nodo X se inserta rompiendo la propiedad se hace una rotación simple (intercambia al padre con el hijo) y en caso que sea intercalado (en el interior) se hace una rotación doble (rotación entre hijo y nieto de X, rotación entre X y su nuevo hijo).

El método más simple de insertar un elemento es el algoritmo recursivo. Al insertar el elemento si la altura del subárbol no cambia, se termina ahí. En cambio si varía, se hace la rotación simple o doble.

**Búsqueda binaria**

Usado para ubicar un elemento dentro de una secuencia ordenada de n elementos en un array. Según un low (0) y high (n – 1) toma el medio = (low + high) / 2. Si el valor buscado es igual a medio, lo devuelve. Si es menor a medio, setea high = medio – 1; si es mayor a medio, setea low = medio + 1.

busquedaBinaria(arr, valor, low, high)

COM

Si low > high entonces

retornar falso

Fin si

medio <- (low + high) / 2

Si valor = arr[medio] entonces

retornar valor

Sino si valor > arr[medio]

retornar busquedaBinaria(arr, valor, medio + 1, high)

Sino si valor < arr[medio]

retornar busquedaBinaria(arr, valor, low, medio – 1)

Fin si

Fin

**Test**

@Test -> se coloca antes del método,

@BeforeEach -> método que se ejecuta antes de cada test,

@DisplayName(“Titulo”) -> se usa para mostrar el nombre del test al usuario,

@RepeatedTest(5) -> define las veces que se va a repetir el test,

@Disable -> desactiva los tests,

@TempDir -> crea archivos y path temporales,

@TestMethodOrder -> según el parámetro, es para ordenar los tests.

Forma, Polígono

Descripción generada automáticamente

Forma, Polígono

Descripción generada automáticamente

Imagen que contiene Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente con confianza baja

Texto

Descripción generada automáticamente

Gráfico

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente