UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100

Árboles

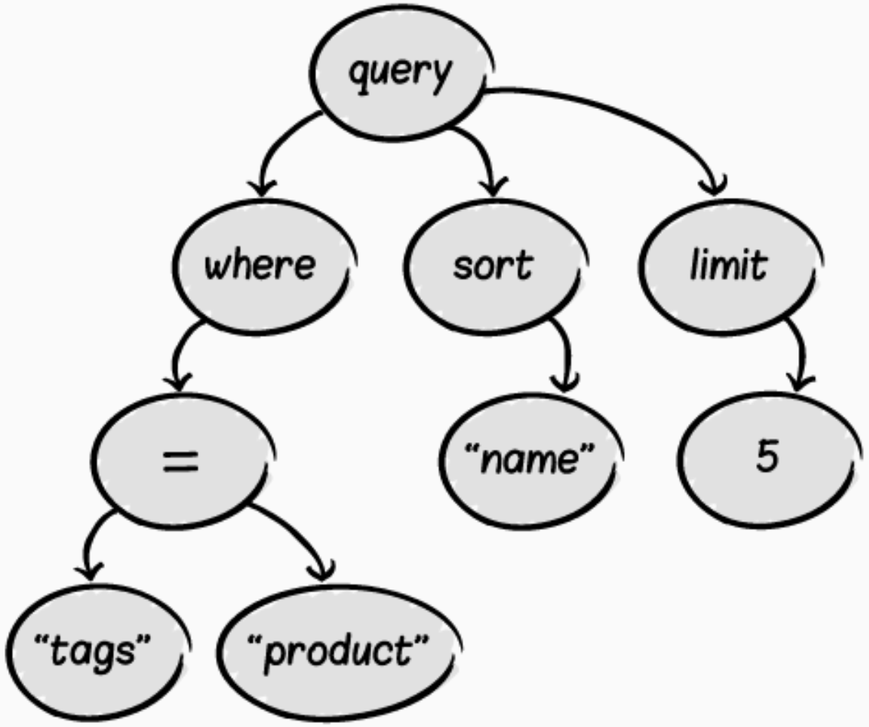
¿Cómo decidimos qué estructura de datos utilizar?

- Relación entre datos. Qué se necesita almacenar?
- Costo de las operaciones
- Uso de memoria

Los árboles usualmente se usan para representar jerarquías, donde la vemos de arriba a abajo

Entonces, un árbol es es una estructura de 0 o más nodos que intenta simular una jerarquía. Donde existe un nodo raíz, y cada nodo tiene hijos (0 o más)

También se puede expresar como una estructura con un nodo root que posee links a “n” sub-árboles



Árboles

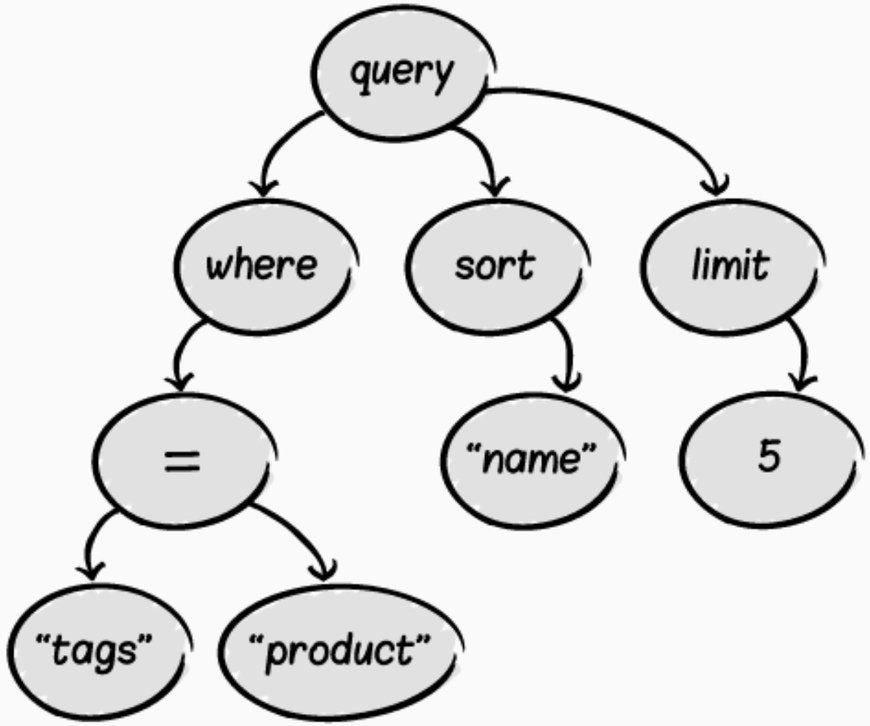
En un árbol existen N nodos y N - 1 uniones

Un **árbol binario** es un árbol donde cada nodo sólo puede tener 2 hijos como máximo.

Un **árbol binario de búsqueda** como se verá más adelante nos permite operaciones rápidas sobre data

Los árboles también pueden ser usados para almacenar diccionarios o para algoritmos de routing en la red.

Hay muchas más aplicaciones, cuáles se les ocurren?



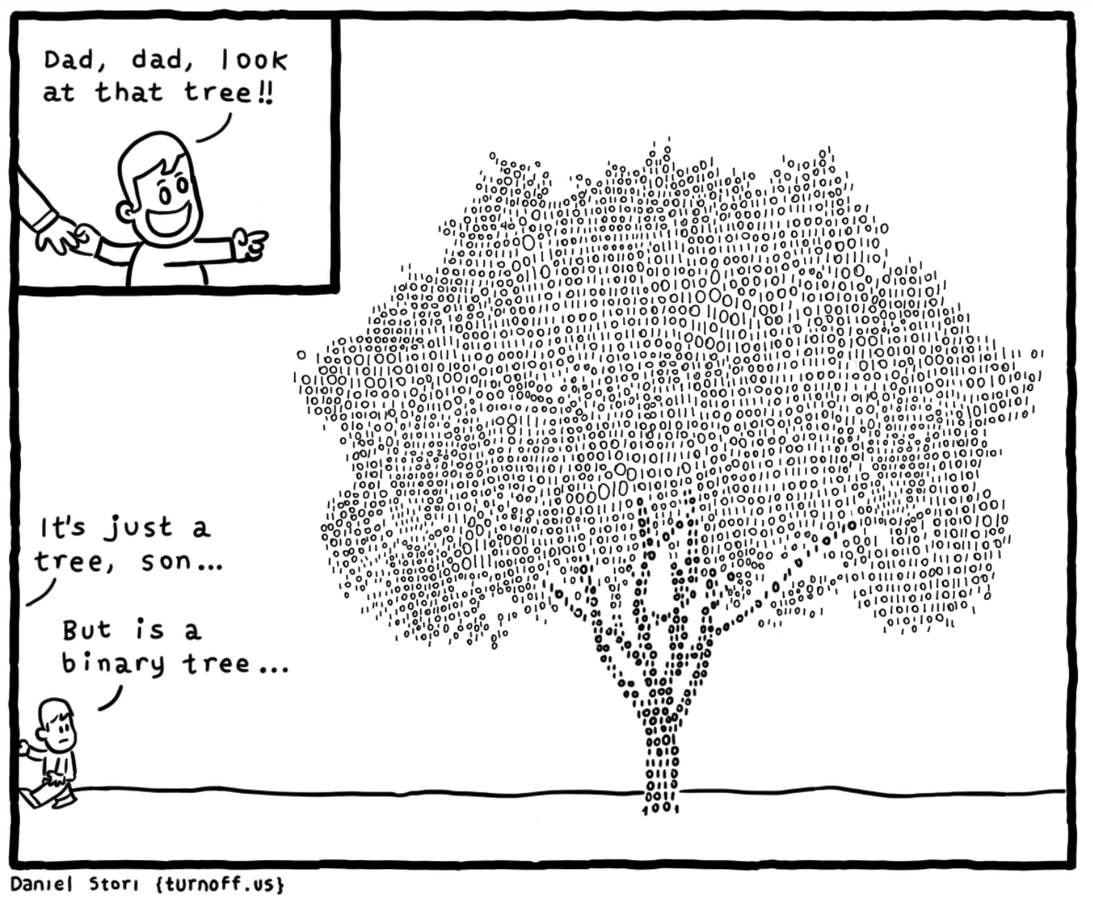
Recordando

Las estructuras que hemos visto tienen un acceso rápido o una actualización rápida.

Las inserciones (front y back) en listas enlazadas toman O(1), mientras que su búsqueda O(n).

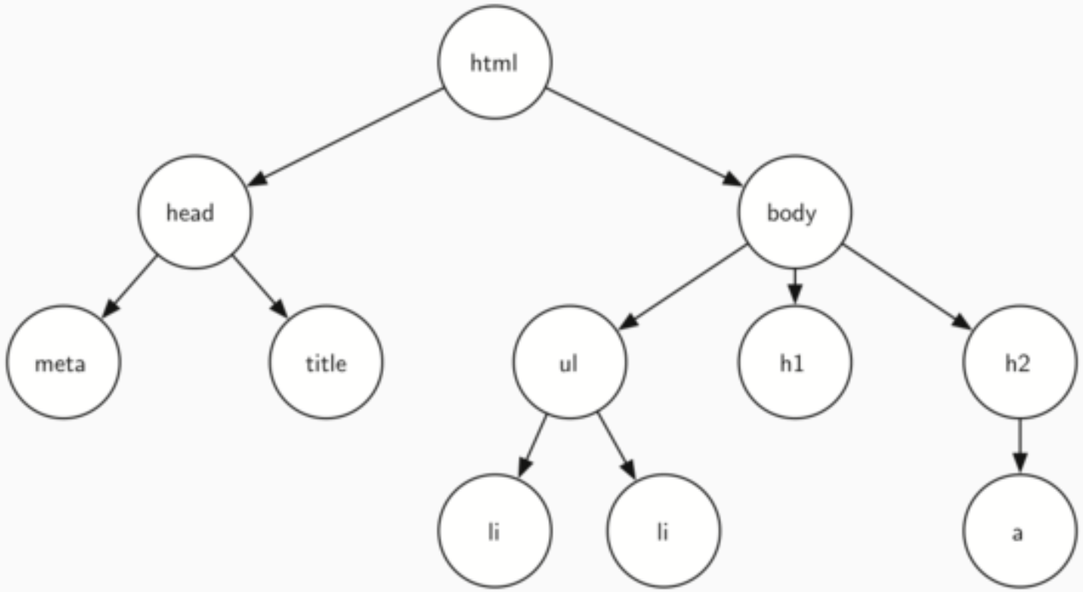
Los arreglos ordenados pueden utilizar búsqueda binaria, pero las actualizaciones toman O(n).

Ahora, la idea básica detrás de un árbol binario de búsqueda es: poder realizar búsqueda binaria, para ello se necesita tener los elementos ordenados y un acceso rápido a cualquier elemento. Además tener un nodo con dos punteros para ir a uno y otro lado.



Tipos de Nodo

- **Raíz:** El primer nodo de un árbol.
- **Padre:** Todo nodo que tiene al menos un hijo.
- **Hijo:** Todo nodo que tiene un padre.
- **Hermano:** Nodos con padre común dentro de la estructura
- **Hoja:** No tienen hijos y se encuentran en los extremos de la estructura.



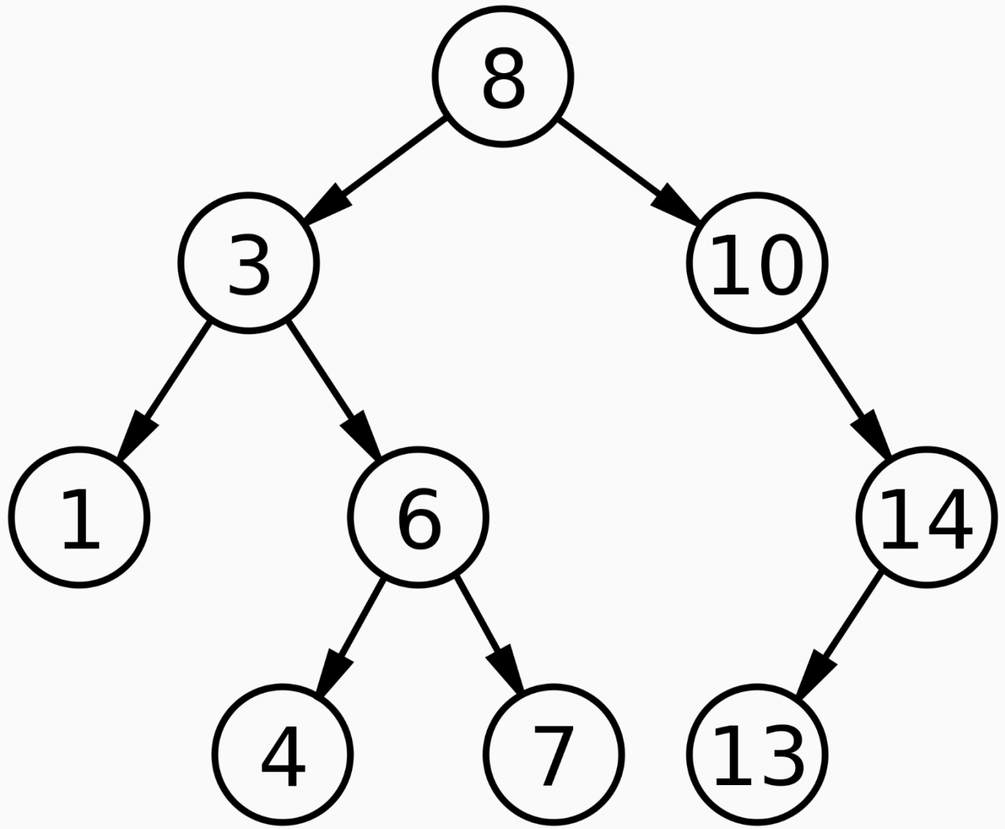
¿Qué es un árbol binario de búsqueda (BST)?

Es una estructura de datos de tipo árbol de orden 2 con nodos finitos, donde se define un nodo raíz y cada nodo tiene máximo dos hijos (hijos izquierdo y derecho).

Para cada nodo, los valores de todos los nodos de los subárboles a la izquierda y a la derecha, son menores y mayores respectivamente (o igual).

Cuando el árbol solo tiene 1 elemento, el nodo raíz es hoja al mismo tiempo.

Orden: El orden de un árbol es el número máximo de hijos que puede tener un nodo x.



¿Qué es un árbol binario de búsqueda?

El **diámetro** de un árbol es el número de nodos en el camino más largo entre dos nodos hojas

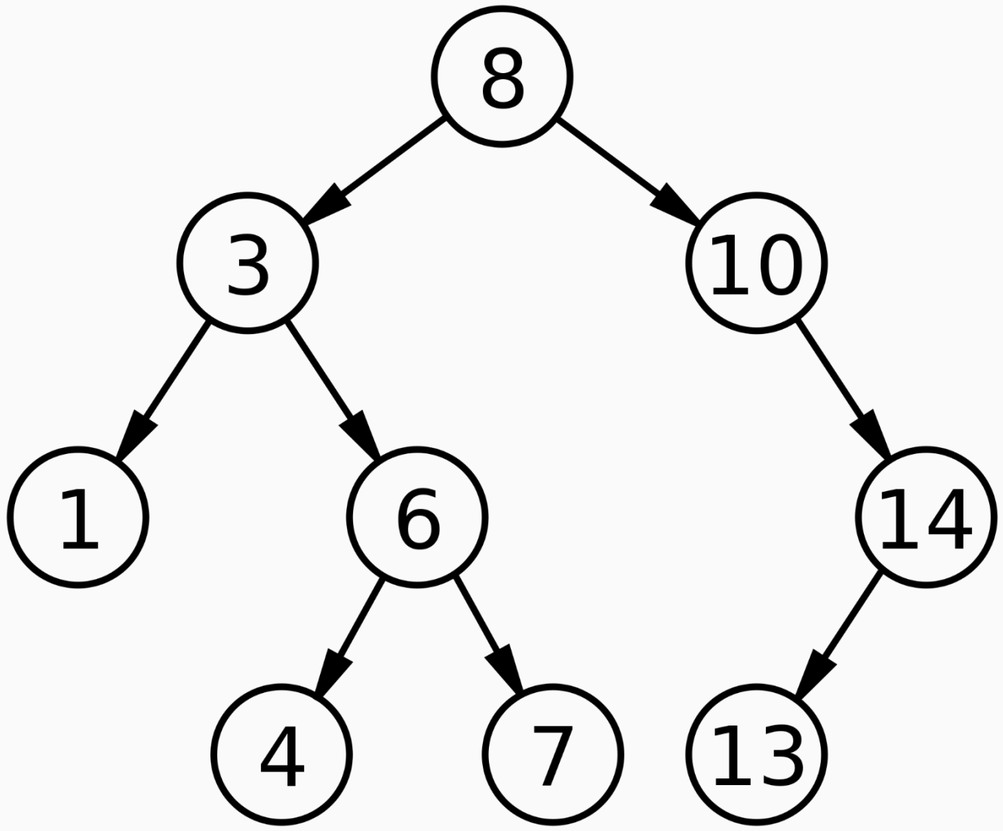
Peso la cantidad de nodos

Nivel de un nodo, es la longitud del camino desde la raíz hasta el nodo (la raíz tiene nivel 0)

La **profundidad** de un nodo es el nivel del nodo

La **altura** de un nodo (como sub árbol) es la distancia más larga hacia un nodo hoja (un nodo hoja tendrá altura 0). Por tanto la altura de un árbol vacío es -1

La **altura de un árbol** será la altura del nodo raíz ¿Cuál sería la altura del árbol de la derecha? Cómo la calcularían en código?



Tipos de árboles binarios

Un árbol binario **completo (o casi completo)** es aquel en que todos sus niveles están completamente llenos, excepto probablemente el último. Y las hojas del último están hacia la izquierda.

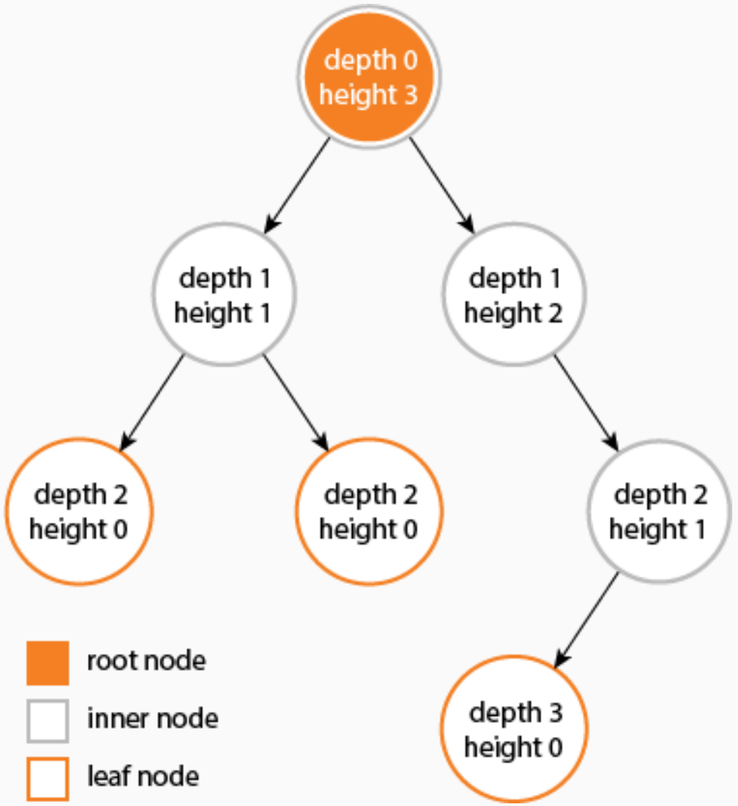
Un árbol binario **lleno** es aquel en el que todos los nodos tienen 0 o 2 hijos.

Un árbol binario **perfecto** es aquel que todas las hojas están al mismo nivel.

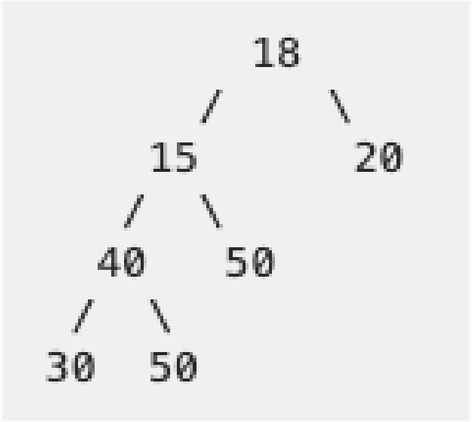
Un árbol binario **balanceado** es aquel en el cual la diferencia de la altura de sus hijos derecho e izquierdo no puede ser más que uno. **Para todo nodo.**

¿Él árbol de la derecha es balanceado? ¿Por qué?

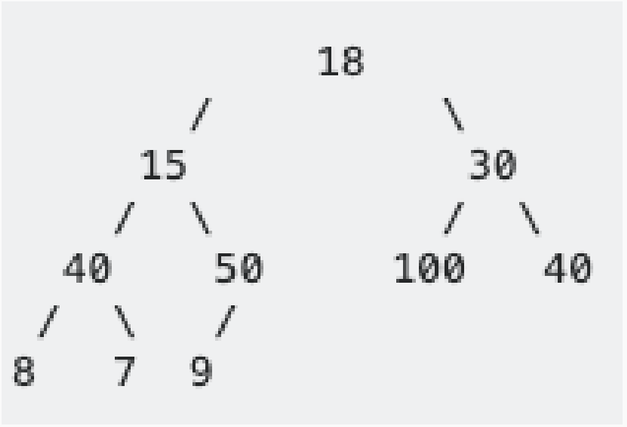
Dibujen 1 un árbol lleno, uno perfecto y un completo



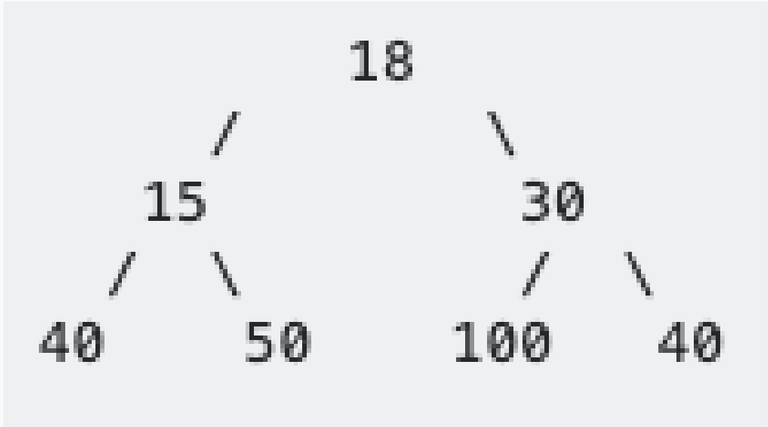
Tipos de árboles binarios



Full Binary Tree



Complete Binary Tree



Perfect Binary Tree

Árbol binario de búsqueda

¿Cómo lo implementarían? Empecemos con el Nodo

Necesitan punteros a la izquierda (menores) y derecha (mayores) y la data. Opcionalmente puede haber un puntero al padre.

La clase que implementa el árbol solo tiene un puntero al root.

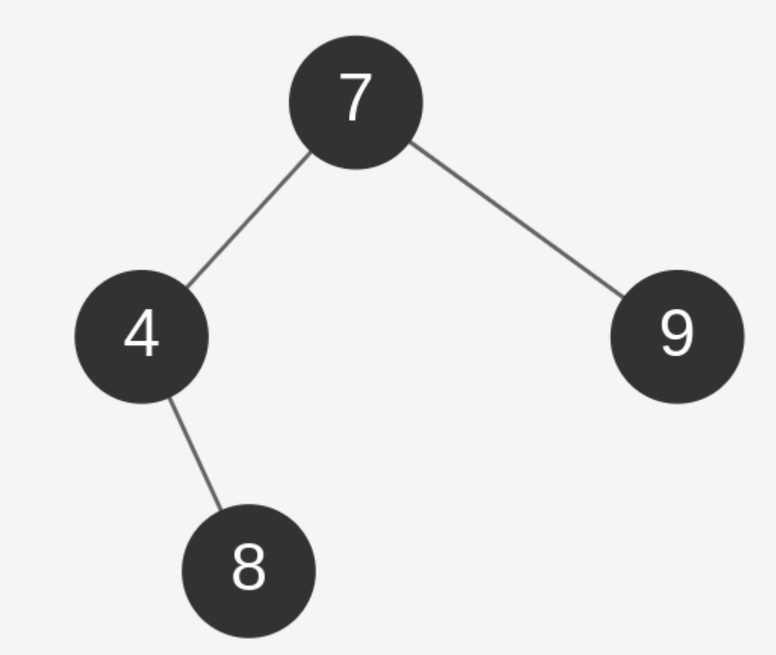
¿Cuáles serían las operaciones básicas a implementar?

Buscar, insertar y borrar. Además, con un iterador se debería poder recorrer.

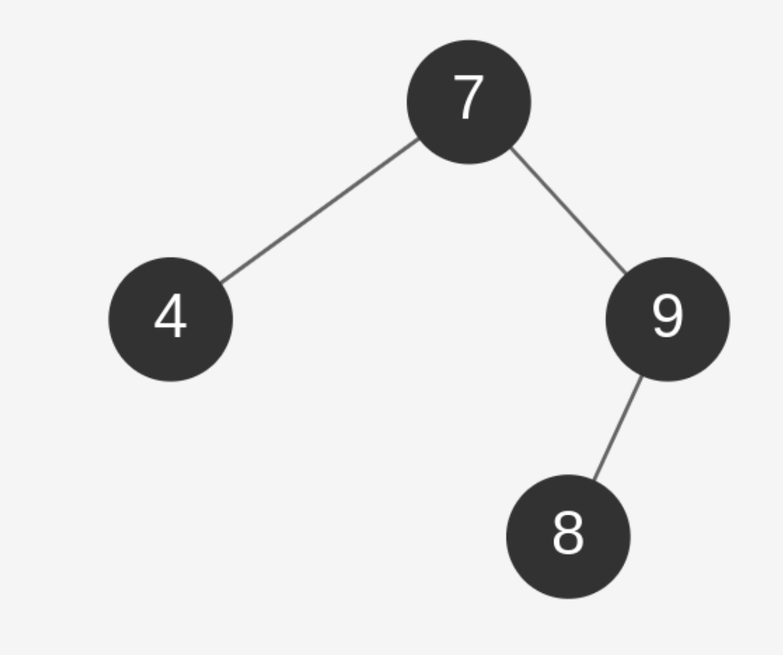
```
template <typename T>
class BSTree {
    struct NodeBT {
        T data;
        NodeBT *left;
        NodeBT *right;
    };

    NodeBT *root;
};
```

Árbol binario de búsqueda



¿Cuál es un árbol binario de búsqueda?



Árbol binario de búsqueda

¿Cómo es la búsqueda?

Se empieza del root y se verifica si el valor es el deseado.

Si lo és, se termina. De otra forma se continúa buscando por la derecha (si es mayor) o izquierda (si es menor).

El algoritmo funciona porque los hijos derecha e izquierda son árboles binarios de búsqueda.

¿Dónde estará el menor elemento de un árbol? ¿Y el mayor?

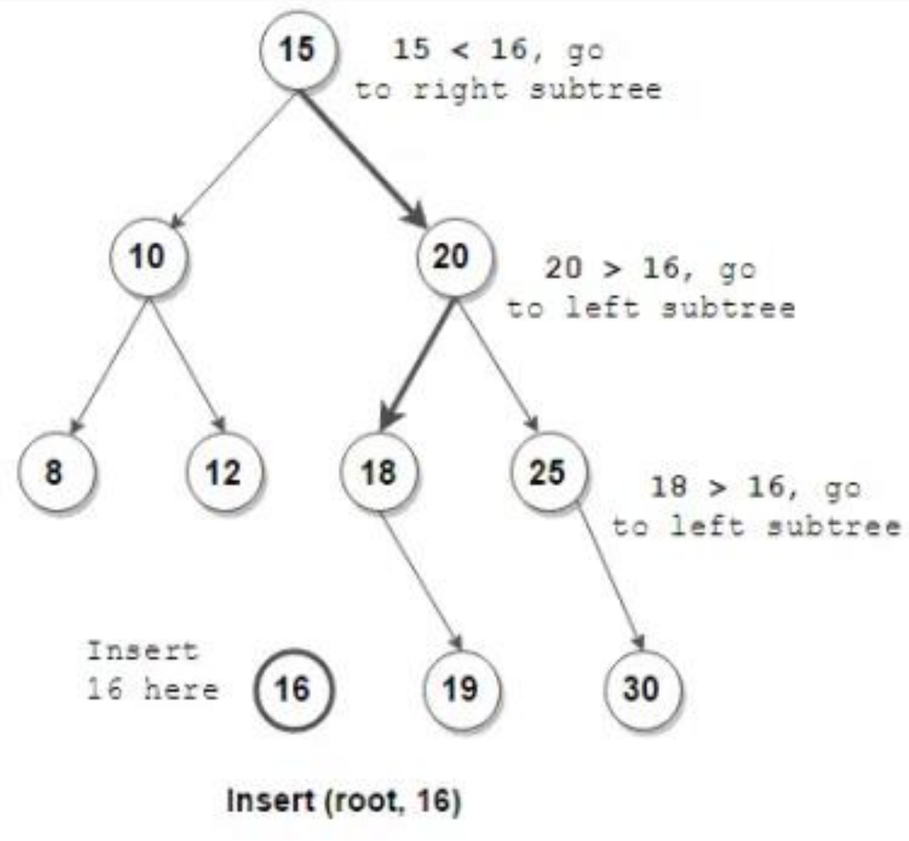
```
bool find(NodeBT* node, T value){
    if(node == nullptr)
        return false;
    else if(value < node->data)
        return find(node->left, value);
    else if(value > node->data)
        return find(node->right, value);
    else return true;
}
```

Árbol binario de búsqueda

¿Cómo harían la inserción de un elemento?

Insertar un elemento, siempre va a ser en un nodo hoja

1. Primero se crea el nuevo nodo con el valor a insertar
2. Se empieza buscando en el BST, desde su raíz hasta encontrar su posición
3. Una vez el siguiente sub-árbol a buscar sea NULL, entonces ahí se procede a insertar el nuevo elemento

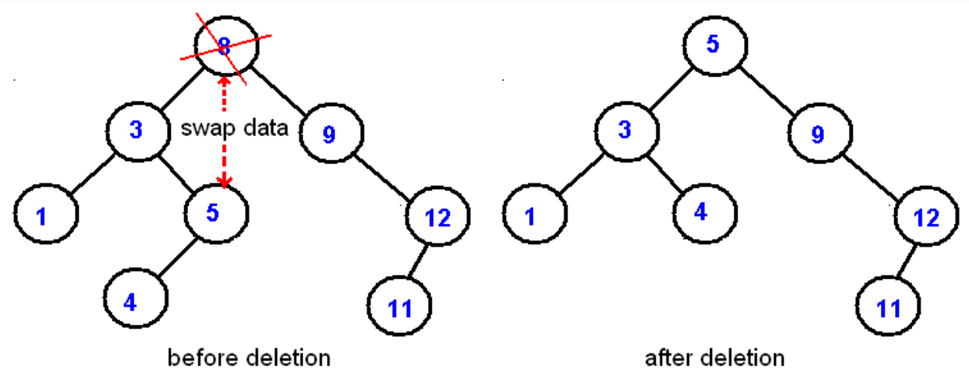


Árbol binario de búsqueda

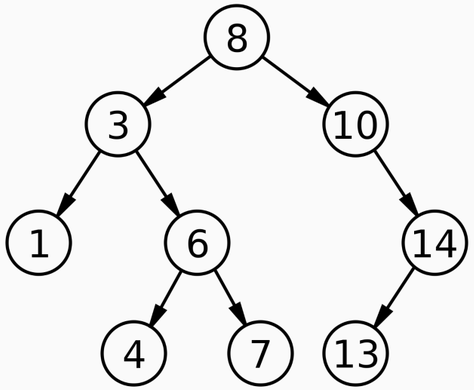
¿Cómo harían el remover un elemento?

Eliminar elementos en árboles binarios, es básicamente buscar el nodo a eliminar y controlar los 3 casos siguientes:

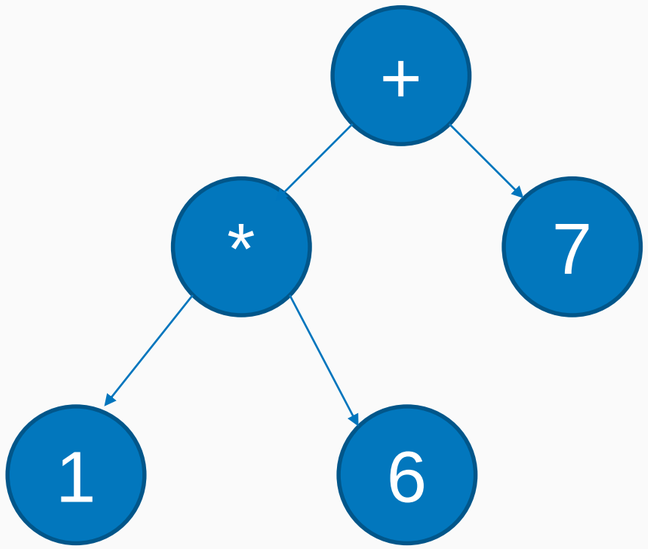
1. Eliminar una hoja
2. Eliminar un nodo con un hijo: Se reemplaza el nodo con su hijo
3. Eliminar un nodo con dos hijos: Se busca el siguiente elemento (o el anterior) al nodo a eliminar y se reemplaza (este mismo se elimina recursivamente)



En el árbol de la derecha: ¿Cuál sería el siguiente elemento o el anterior, en el caso de eliminar 3?



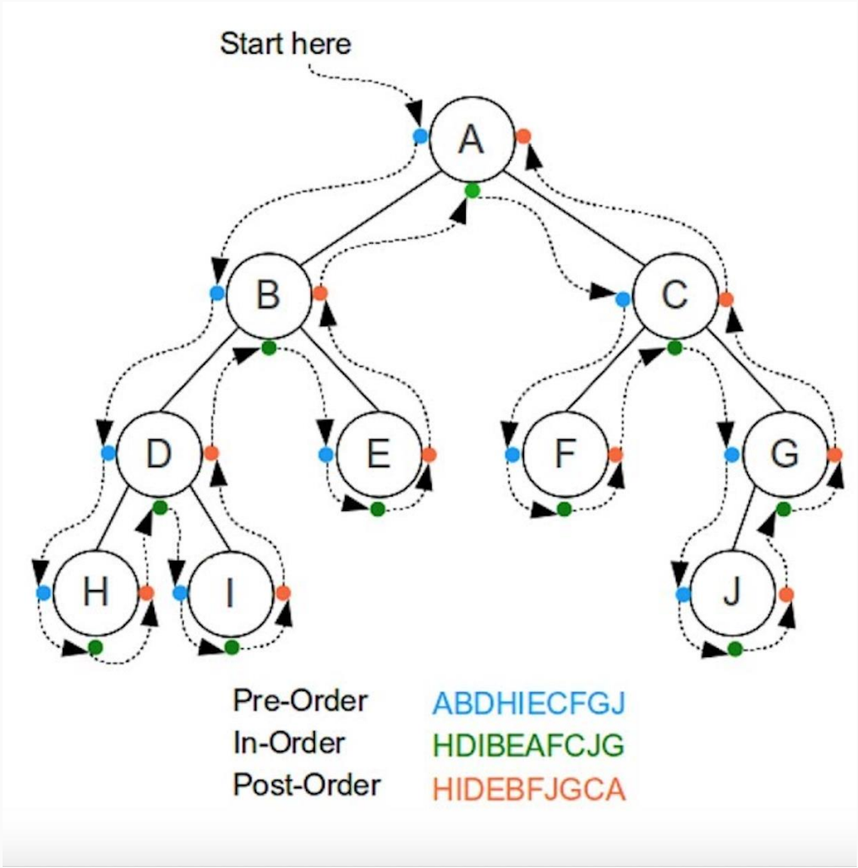
Recorridos de un BST



Postorder(+): 1 6 * 7 +

Preorder(+): + * 1 6 7

Inorder(+): 1 * 6 + 7



Recorridos de un BST

Recorrer significa visitar cada nodo en orden específico

1. Post-order: Se recorre el hijo izquierdo, luego derecho y al final se imprime la data

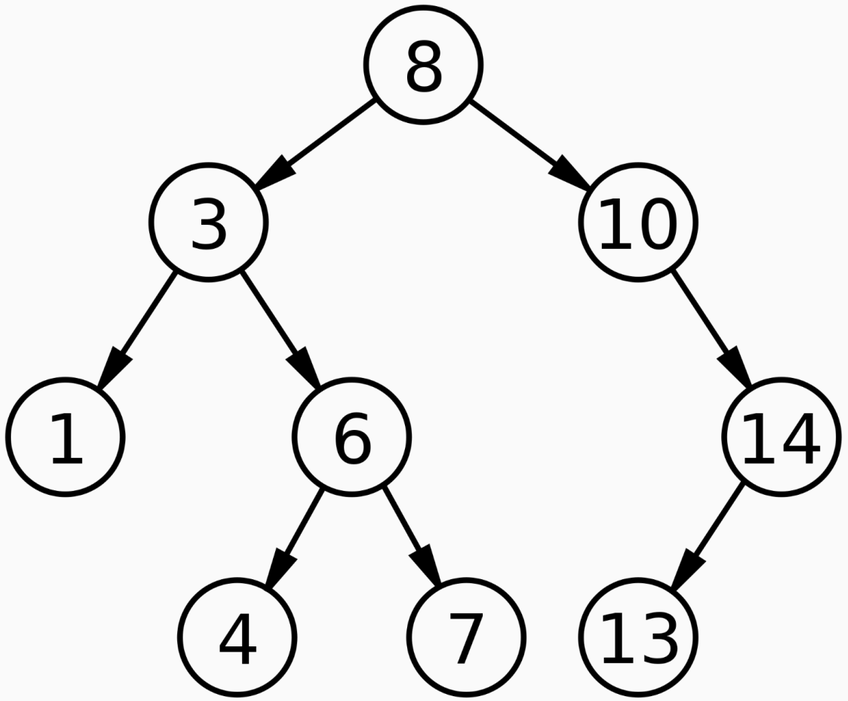
1,

2. In-order: Se recorre el hijo izquierdo, se imprime la data y al final se recorre el hijo derecho.

1,

3. Pre-order: Primero se imprime la data, y luego se recorre el hijo izquierdo y al final el derecho

8,



Recorridos de un BST

Recorrer significa visitar cada nodo en orden específico

1. Post-order: Se recorre el hijo izquierdo, luego derecho y al final se imprime la data

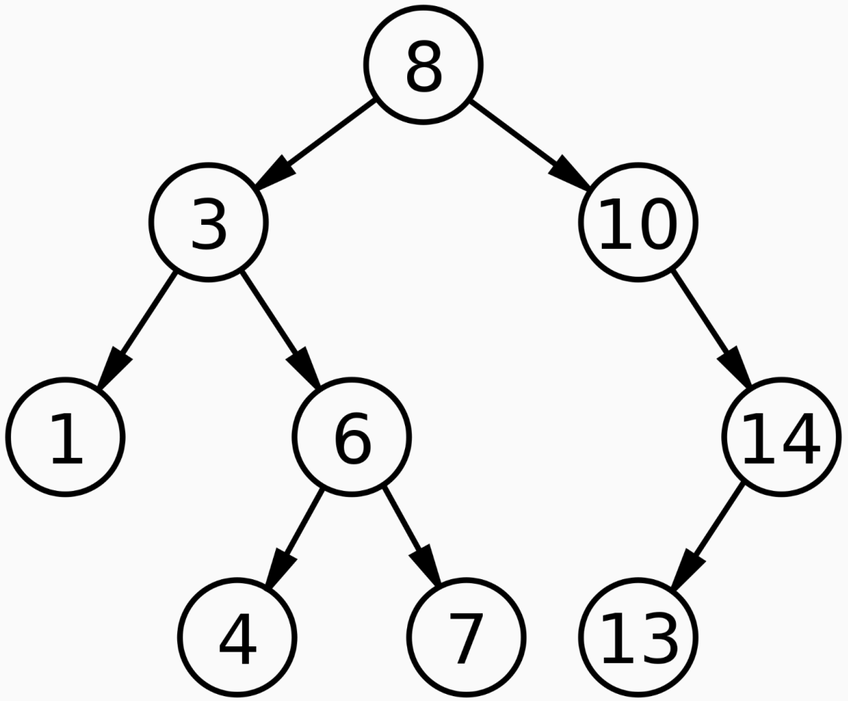
1, 4, 7, 6, 3, 13, 14, 10, 8

2. In-order: Se recorre el hijo izquierdo, se imprime la data y al final se recorre el hijo derecho.

1, 3, 4, 6, 7, 8, 10, 13, 14

3. Pre-order: Primero se imprime la data, y luego se recorre el hijo izquierdo y al final el derecho

8, 3, 1, 6, 4, 7, 10, 14, 13



Recorridos de un BST

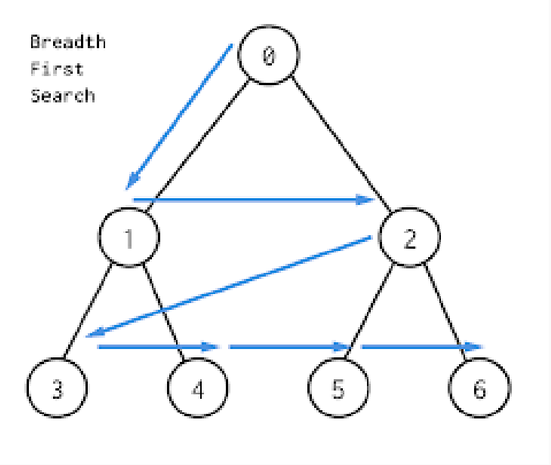
Recorrer significa visitar cada nodo en orden específico

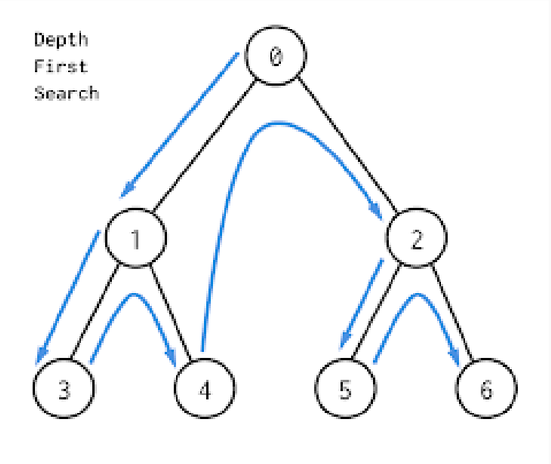
4. **Recorrido en anchura:** empieza por la raíz y visita todos sus hijos, luego repite el proceso explorando todos los hijos en el siguiente nivel del árbol hasta llegar al último nivel.

   0, 1, 2, 3, 4, 5, 6

5. **Recorrido en profundidad:** empieza por la raíz y explora todos los nodos a lo largo de una rama hasta el ultimo nivel, luego retrocede y vuelve a repetir el proceso en cada uno de los hermanos del nodo ya procesado.

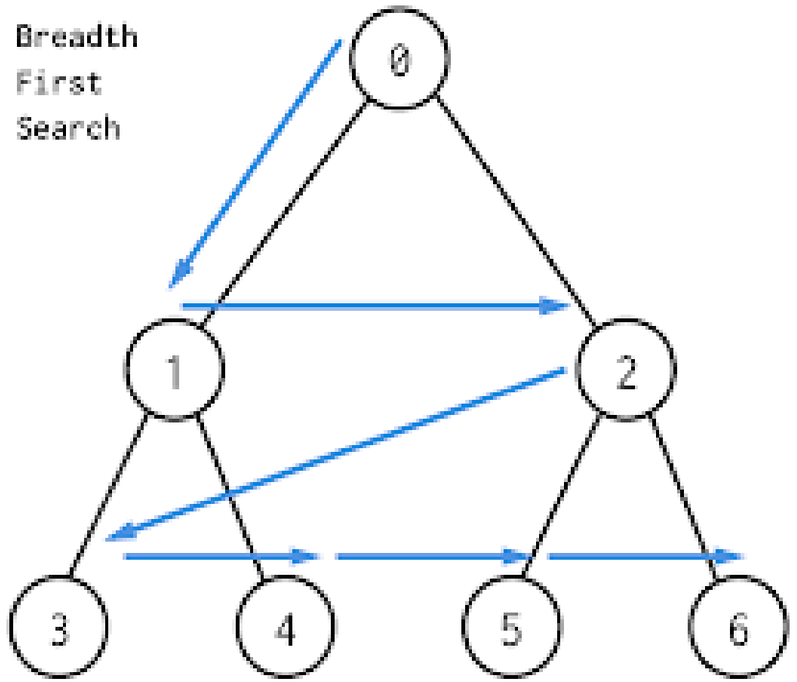
   0, 1, 3, 4, 2, 5, 6





Recorridos de un BST

Recorrido en anchura:



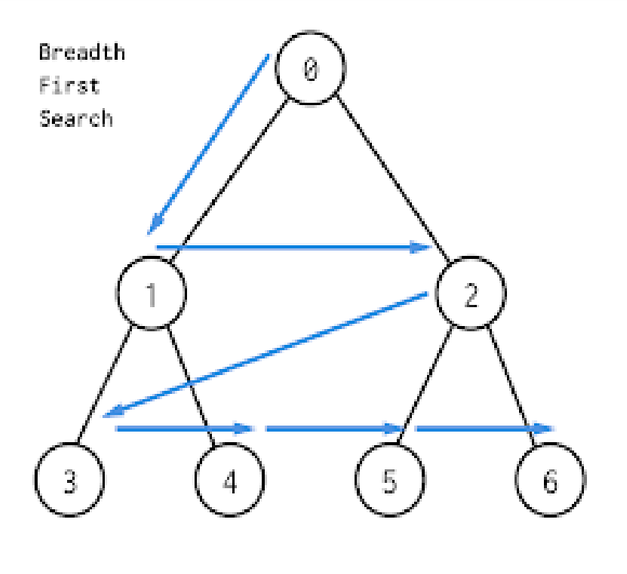
front

tail

Display (pop): 0, 1, 2, 3, 4, 5, 6

Recorridos de un BST

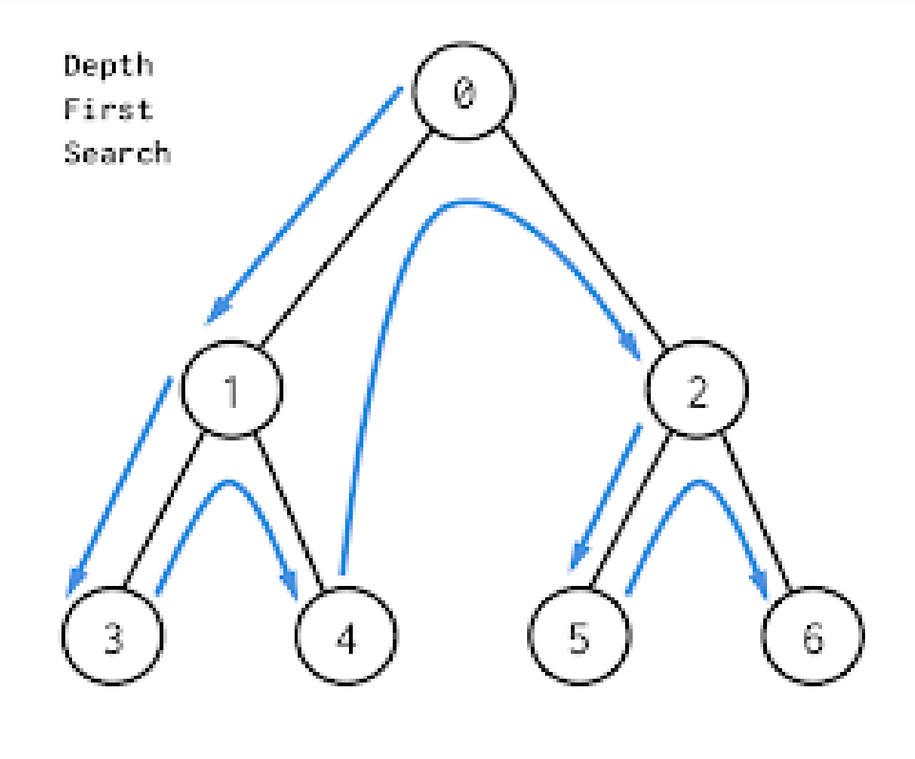
Recorrido en anchura:

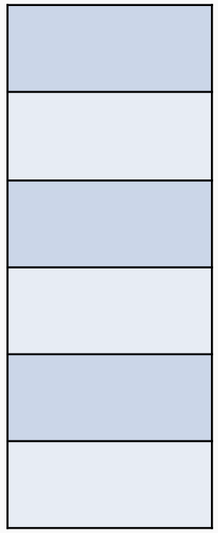



```
void BreadthFirstSearch(){
    Queue<NodeBT*> queue;
    queue.enqueue(this->root);
    while(!queue.is_empty()){
        NodeBT* node = queue.dequeue();
        cout<<node->data<<endl;
        if(node->left != nullptr)
            queue.enqueue(node->left);
        if(node->right != nullptr)
            queue.enqueue(node->right);
    }
}
```

Recorridos de un BST

Recorrido en profundidad:



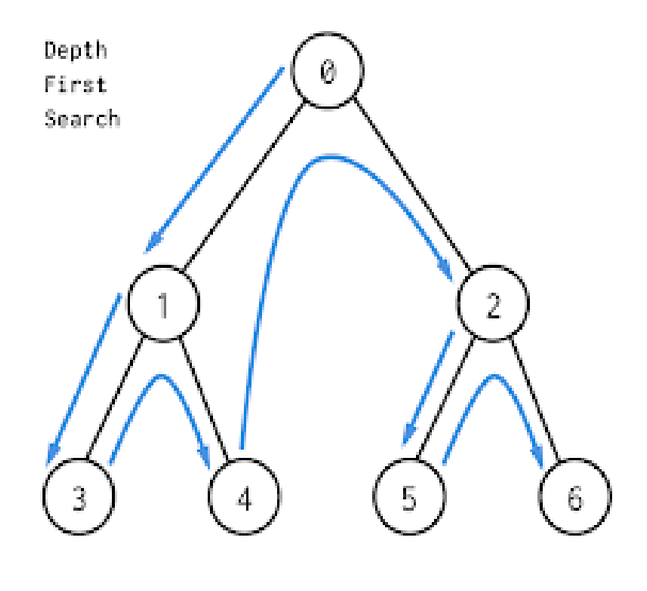


front

Display (pop): 0, 1, 3, 4, 2, 5, 6

Recorridos de un BST

Recorrido en profundidad:




```
1 BSTree<int>::iterator ite = bstree->begin();
2 while(ite != bstree->end()) {
3     cout << *ite << endl;
4     ++ite;
5 }
```

```
template<typename F>
void DepthFirstSearch(F process){
    Stack<NodeBT*> stack;
    stack.push(this->root);
    while(!stack.is_empty()){
        NodeBT* node =  stack.pop();
        process(node->data);
        if(node->right != nullptr)
            stack.push(node->right);
        if(node->left != nullptr)
            stack.push(node->left);
    }
}
```

Ejercicios

Inserte los siguientes elementos en un árbol binario de búsqueda:

12, 8, 20, 5, 1, 15, 25, 7, 11, 9, 13, 22, 18, 26

Ejecute la impresión pre-order, in-order, post-order, anchura y profundidad

pre-order:

in-order:

post-order:

anchura:

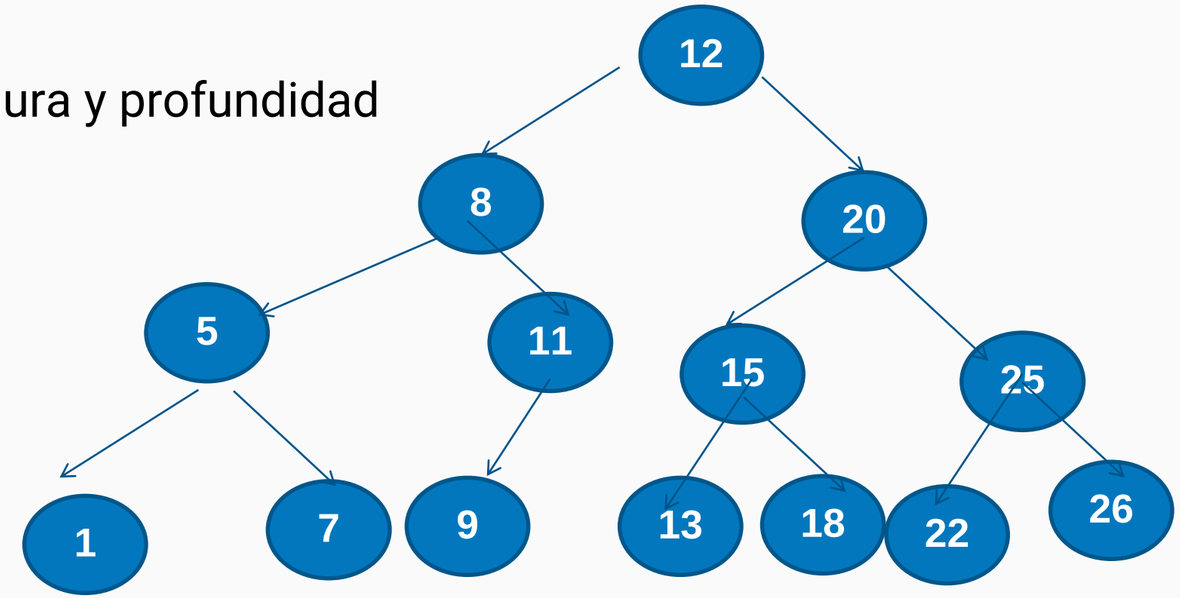
profundidad:

Ejercicios

Inserte los siguientes elementos en un árbol binario de búsqueda:

12, 8, 20, 5, 1, 15, 25, 7, 11, 9, 13, 22, 18, 26

Ejecute la impresión pre-order, in-order, post-order, anchura y profundidad



pre-order: 12,8,5,1,7,11,9,20,15,13,18,25,22,26

in-order: 1,5,7,8,9,11,12,13,15,18,20,22,25,26

post-order: 1,7,5,9,11,8,13,18,15,22,26,25,20,12

anchura: 12,8,20,5,11,15,25,1,7,9,13,18,22,26

profundidad: 12,8,5,1,7,11,9,20,15,13,18,25,22,26

Ejercicios

Inserte los siguientes elementos en un árbol binario de búsqueda:

1,5,7,8,9,11,12,13,15,18,20,22,25,26

Ejecute la impresión

pre-order

in-order

post-order

anchura

profundidad

Ejercicios

¿Cuál es la complejidad de inserción, eliminación y búsqueda?

¿Qué problemas se puede presentar en el árbol binario de búsqueda?

¿Cómo insertamos elementos que ya están ordenados?

¿Cómo harían para que un árbol sea balanceado?

UTEC
UNIVERSIDAD DE INGENIERÍA
Y TECNOLOGÍA

Welcome to Algorithms and Data Structures! - CS2100