7. Arboles

Un arbol es un conjunto de nodos organizados segun una jerarquia. Si contiene al menos un nodo, entonces debe tener un *nodo raiz*. Este es el nodo de nivel mas alto, a partir de este naces los demas. Es el nivel mas alto de jerarquia. Todos los nodos del arbol estan unidos al nodo raiz mediante algun camino.

Los nodos pueden tener mutiples nodos hijos, pero solamente un nodo padre. El vinculo entre dos nodos se llama *brazo*. En los arboles se definen *nodo hermano, ancestro* y *descendiente*. Sus definiciones son analogas a los arboles genealogicos.

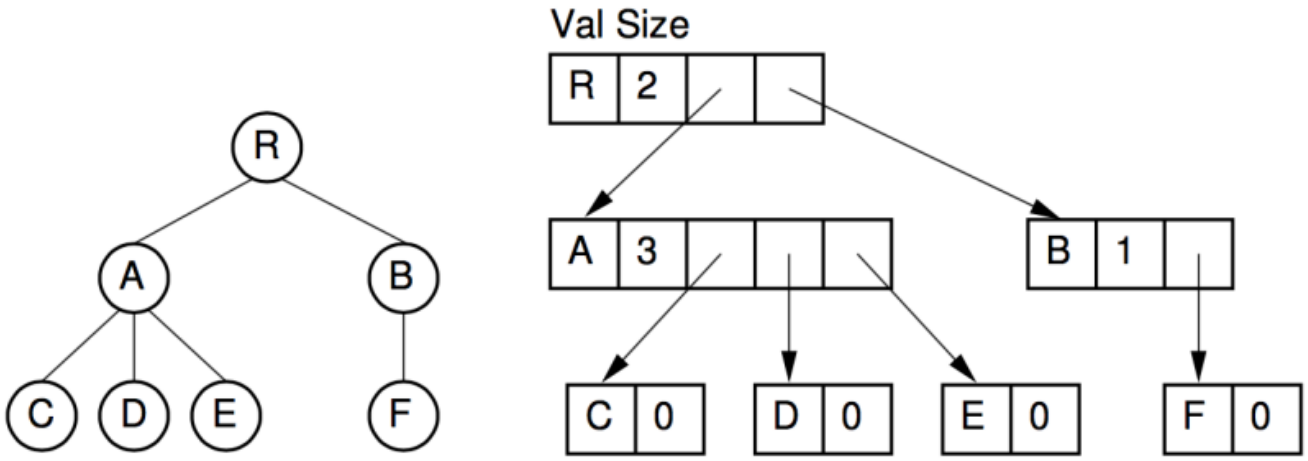
Un nodo sin hijos se llama *hoja*. Un nodo con hijos se llama *nodo interno*. El nivel de un nodo es el numero de brazos entre el nodo y la raiz, mas uno. La altura de un arbol no vacio es el numero de brazos en el camino mas largo entre el nodo raiz y una hoja. Si no tiene brazos, se dice que tiene altura 0. Si no tiene nodos, se dice que tiene altura -1.

A partir de un nodo dentro de u n arbol, se puede definir un *sub-arbol*. Este conocimiento me da la idea de que puede ser muy potente trabajar los arboles de manera recursiva.

Codear arboles

Podemos representar los nodos de los arboles con un elemento y una lista de punteros a nodos hijo. Para lograr mejor eficiencia de memoria, esta lista se puede crear con un `vector`.

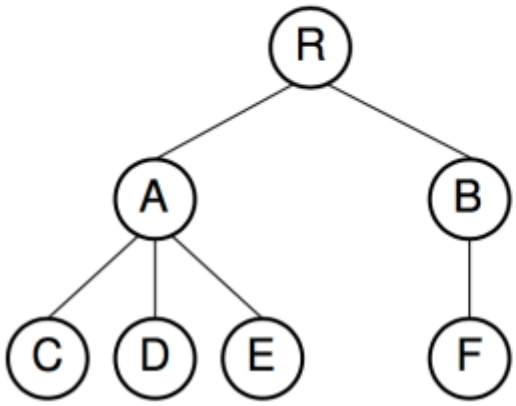
```
template <class T>
class Treenode
{
        public:
                T value;
                vector<TreeNode<T> *> children;
};
```

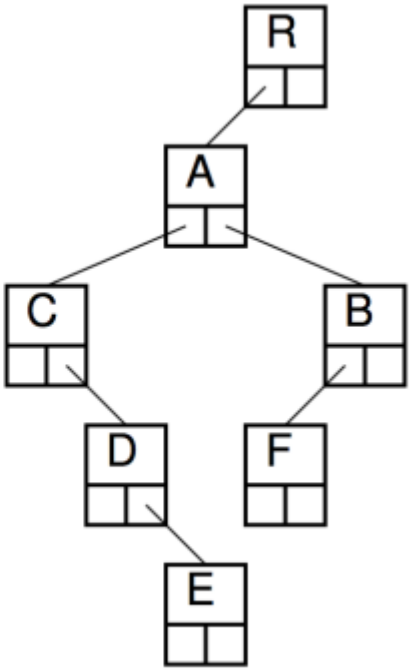


Asi es como se ve un arbol hecho con `vector`. Con `list`, todos los elementos quedan asociados por punteros. Como ya se dijo, sera mucho mas eficiente en insercion y eliminacion de nodos, pero ocupa mas memoria.

Se pueden representar los nodos con un elemento y dos punteros; uno al primer hijo por izquierda y, otro, el siguiente hermano por derecha.

```
template <class T>
class TreeNodeB
{
public:
        TreeNodeB<T> *firstChild;
        TreeNodeB<T> *nextSibling;
};
```





Se pueden crear arboles estrictamente binarios o `Quadtrees` u `Octrees` con expresiones como `(TreeNode<T> *[4]child)` para el segundo caso. La cantidad de hijos en este tipo de arboles es acotada superiormente.

Recorrido de arboles

Un *recorrido* involucra visitar cada uno de los nodos de manera sistematica. La *visita* involucra algua operacion sobre un nodo, procesarlo, modificarlo, mandarlo a disco.

Recorrido DFS

En un recorrido *depth-first search* se visitan todos los nodos recursivamente, entrando primero en profundidad. Para la recursion de este recorrido se una una pila.

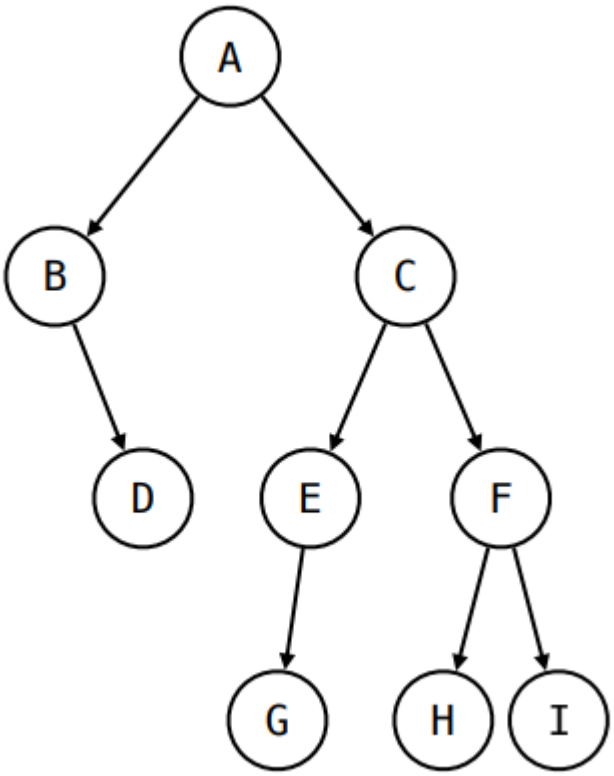
```
void traverseDFS(TreeNode *node)
{
        if (node == NULL)
                return;

        visitPreorder(node);

        for (auto childNode : node->children)
                traverseDFS(childNode);

        visitPostorder(node);
}
```

En este caso la recursion de realiza en el call stack, pero tambien se podria usar un `stack` de STL. Hay que tener cuidado, con el call stack, de no sobrepasar el stack cuando el arbol sea muy grande. De lo contrario, puede ocurrir un stack overflow.



Para este arbol, el recorrido DFS pre-orden resulta: `ABDCEGFHI`
El recorrido DFS post-orden es: `DBGEHIFCA`

En el recorrido **pre-orden** se recorre el arbol por subarboles, primero por izquierda y luego por derecha. Se devuelve un nodo cuando se lo alcanza por primera vez. Cuando meto un nodo al call stack lo imprimo, y luego los elimino del call stack para seguir visitando.

En el recorrido **post-orden** se hace lo mismo pero se devuelve el nodo cuando se lo visita por ultima vez. Primero meto los nodos en el call stack, luego, cuando no tienen mas hijos los visito. Tambien esta el recorrido **in-order**, que devuelve el nodo cuando se lo visita por segunda vez.

Para las hojas, es util plantear que tienen dos subarboles vacios. El truco para recorrer los arboles de esta manera es dibujarlos, y circular por su exterior en sentido antihorario desde la raiz.

Recorrido BFS

El *breadth-first search* visita todos los nodos iterativamente, entrando en anchura. Se comienza por el nodo raiz, bajando nivel por nivel hasta cubrir todo al arbol. Esta iteracion se reliza con una cola, por ejemplo, un `queue` de STL.

```
void traverseBFS(TreeNode *root)
{
        if (root == NULL)
                return;

        queue q;
        q.push(root);
        while (!q.empty())
        {
                TreeNode *node = q.pop();

                visit(node);
                f (root == NULL) return; queue q; q.push(root); while
(!q.empty()) { TreeNode *no
                for (auto childNode : node->children)
                        q.push(childNode);
        }
}
```

El nodo se puede visitar cuando se pushea o cuando se popea. Generalmente es mas facil hacerlo cuando se popea.

Para el mismo arbol que se mostro antes el recorrido BFS es: `ABCDEFGHI`
Cuando llego a un nodo, encolo sus hijos. Luego, cuando desencolo un hijo, encolo los hijos de ese.

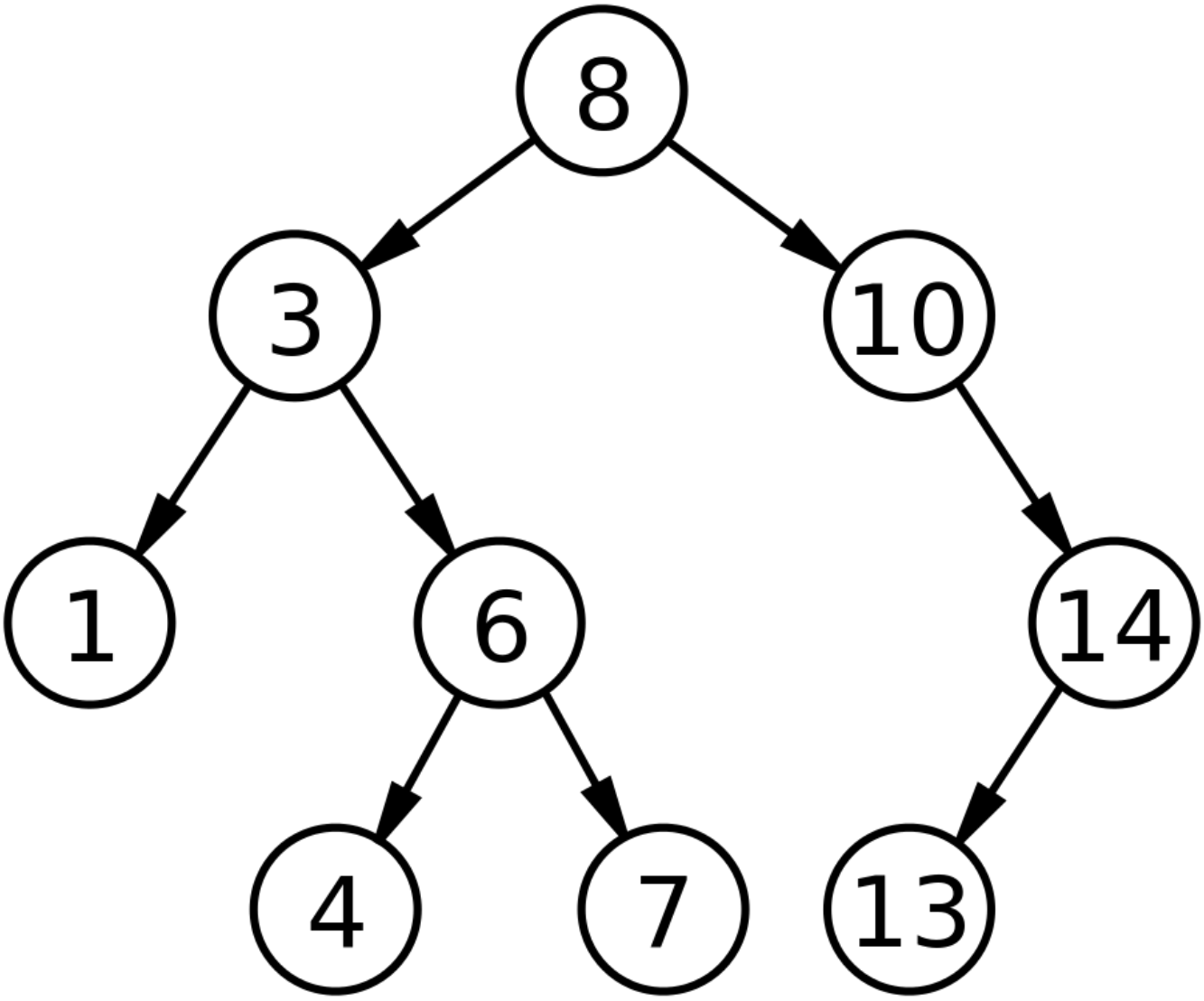
Arboles binarios de busqueda

Un arbl binario de busqueda es un arbol binario en el que cada nodo posee un valor llamado *clave* La clase es unica, no se repite entre nodos. La clave de cada nodo es mayor a la de su subarbol izquierdo y menor a la de su subarbol derecho.

A continuacion, una implementacion de este modelo.

```
template <class Key, class Value>
class BTSTNode
{
public:
        Key key;
        Value value;
```

```
        BSTNode<Key, Value> *left:
        BSTNode<Key, Value> *right;
};
```



Algoritmo de Busqueda Estandar

Para encontrar nodos en un arbol binario se puede aplicar el algoritmode busqueda estandar. Si la clave que se busca es menor a la clave del nodo actual, se baja por la izquierda. De lo contrario, se baja por la derecha. Esta decision se repite hasta encontrar el nodo. (binary search)

```
BSTNode *findNode(BSTNode *n, const Key &key)
{
        if (n == NULL)
                return NULL;
        if (n->key == key)
```

```
                return n;
        if (n->key < key)
                return findNode(n->left, key);
        else
                return findNode(n->right, key);
};
```

Un dato interesante sobre los arboles de busqueda binarios es que el recorrido in-order de uno de ellos devuelve los nodos enumerados en orden creciente.

Algoritmo de Insercion Estandar

Para insertar un elemento en un arbol de busqueda binario debo intentar buscarlo en el mismo y luego insertarlo donde esperaria encontrarlo.

```
void insertNode(BSTNode *&n, const Key &key, const Value &value)
{
        if (n == NULL)
        {
                n = new BSTNode;
                n->key = key;
                n->value = value;
                n->left = n->right = NULL;
                return;
        }

        if (n->key == key)
                return;

        if (n->key < key)
                insertNode(n->left, key, value);

        else
                insertNode(n->right, key, value);
}
```

Algoritmo de Eliminacion Estandar

Para eliminar una hoja, simplemente la elimino y hago que el punteor quepreviamente apuntaba a la misma apunte a `NULL`.

Para eliminar un nodo interno con un solo hijo, se puede seguir el mismo procedimiento que en una lista simplemente enlazada. El problema viene cuando tengo que eliminar un nodo con dos hijos.

En este caso debo buscar por la izquierda, el que mas a la derecha esta (es decir el mayor de los que son menores al eliminado), se lo elimina y se lo inserta arriba. Luego, se subsanan las consecuencias de haber eliminado ese nodo en su subarbol.

Arbol Balanceado

Se define al arbol balanceado como un arbol en el que la suma de las alturas de sus subarboles izquierdo y derecho difieren, a lo sumo, en uno. Para arboles balanceados, la busqueda es de, a lo sumo, $O(\log(n))$. Esto es eficiente.

Arboles AVL

Si se insertan los elementos a un arbol binario de busqueda de manera ordenada, quedaran ordenados como en una lista. Esto deja en evidencia que los algoritmos estandar de insercion y eliminacion. Los arboles autobalanceados AVL logran el balanceo mediante un factor de balance que se anade a cada nodo.

Este factor es un entero que precomputa la diferencia de las alturas de los subarboles. Cuando se detecta que el arbol no esta balanceado, se realizan rotaciones para balancear el arbol.

Comentario muy feliz: todo esto esta incluido en STL. Ver las funcioenes `set`, `map`, `multiset` y `multimap`.