1. Problema

Implementar las operaciones insert y delete de un BST (Binary Search Tree).

2. Código

2.1. Tree.h

```
#include "list"
#include "algorithm"
#include "fstream"

using namespace std;

template <typename T>
class Tree
{
    public:
        Tree();
        class Nodo{
            public:
                Nodo();
                Nodo(T dato);
                T dato;
                Nodo * hijos[2];
                void destruir();
        };
        bool find(T, Nodo **&);
        bool insert(T dato);
        void del(T valor);
        virtual ~Tree();
        void printDot();
    protected:
    private:
        void _menorIzq(Nodo **&);
        Nodo * root;
};
template <typename T>
bool Tree<T>::insert(T dato){
    Nodo **nodo;
    if(find(dato,nodo))return false;
    *nodo = new Nodo(dato);
    return true;
}

template <typename T>
bool Tree<T>::find(T dato, Nodo **&nodo){
    nodo = &root;
    while(*nodo != nullptr){
        if((*nodo)->dato == dato)return true;
        nodo = &((*nodo)->hijos[(*nodo)->dato < ↩
            dato]);
    }
    return false;
}
template <typename T>
void Tree<T>::_menorIzq(Nodo ** &nodo){
    nodo = &((*nodo)->hijos[0]);
    while(true){
        if(!(*nodo)->hijos[1])return;
        nodo = &((*nodo)->hijos[1]);
    }
}
template <typename T>
void Tree<T>::del(T valor){
    Nodo ** nodo;
    if(!this->find(valor,nodo))return;
    if((*nodo)->hijos[0] and (*nodo)->hijos[1]){
        Nodo ** temp = nodo;
        _menorIzq(temp);
        swap((*temp)->dato, (*nodo)->dato);
        nodo = temp;
    }
    if((*nodo)->hijos[0]){
        *nodo = (*nodo)->hijos[0];
    }
    else if((*nodo)->hijos[1]){
        *nodo = (*nodo)->hijos[1];
    }
    else{
        *nodo = nullptr;
    }
}
template <typename T>
void Tree<T>::printDot(){
    ofstream archivo("eje.dot");
    if(archivo.fail()){
        cout<<"EL archivo no se pudo abrir"<<endl;
        return;
    }
    archivo<<"digraph{";
    list<Nodo *> result;
    if(root) result.push_back(root);
    for(auto iter = result.begin(); iter != result↩
        .end(); ++iter){
        archivo<<(*iter)->dato<<endl;
        if((*iter)->hijos[0]){
            archivo<<(*iter)->dato<<"->"<<(*iter)↩
                ->hijos[0]->dato<<endl;
            result.push_back((*iter)->hijos[0]);
        }
        if((*iter)->hijos[1]){
            archivo<<(*iter)->dato<<"->"<<(*iter)↩
                ->hijos[1]->dato<<endl;
            result.push_back((*iter)->hijos[1]);
        }
    }
    archivo<<"}";
    archivo.close();
    system("dot -Tpdf eje.dot -o eje.pdf");
}
template <typename T>
void Tree<T>::Nodo::destruir(){
    if(hijos[0])hijos[0]->destruir();
    if(hijos[1])hijos[1]->destruir();
    delete this;
}
template <typename T>
Tree<T>::Tree(){
    root = nullptr;
}

template <typename T>
Tree<T>::Nodo::Nodo(){
    hijos[0] = nullptr;
    hijos[2] = nullptr;
}

template <typename T>
Tree<T>::Nodo::Nodo(T dato){
    this->dato = dato;
    hijos[0] = nullptr;
    hijos[1] = nullptr;
}

template <typename T>
Tree<T>::~Tree(){
    if(root) root->destruir();
}
```

La funcion printDot es usada para generar un pdf con el arbol dibujado. Para esto utilizo un software llamado graphviz. Creamos un archivo .dot y luego ejecutamos una linea de comando para generar el respectivo pdf. Esta funcion la usaremos para visulizar los ejemplos.

2.2. main.cpp

```
#include <iostream>
#include "Tree.h"

using namespace std;

int main()
{
    Tree<int> arbolito;
    arbolito.insert(9);
    arbolito.insert(5);
    arbolito.insert(14);
    arbolito.insert(3);
    arbolito.insert(8);
    arbolito.insert(11);
    arbolito.insert(20);
    arbolito.insert(16);
    arbolito.insert(18);
    arbolito.printDot();
    arbolito.del(18);
    arbolito.del(14);
    arbolito.del(20);
    arbolito.printDot();
}
```

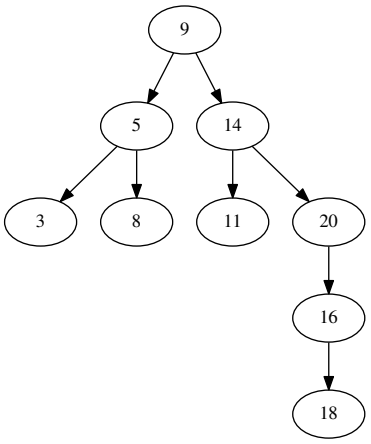

Figura 1: Imagen antes del delete

3. Ejemplos

Los archivos .dot tienen una determinada estructura. Todo se escribe dentro de 'digraph'. El archivo .dot que se genera al usar por primera vez la funcion 'printDot' es el siguiente:

```
digraph{
9
9->5
9->14
5
5->3
5->8
14
14->11
14->20
3
8
11
20
20->16
16
16->18
18
}
```

Los nodos se escriben solos para inicializarlos para luego poder crear sus relaciones con el operador '->'.

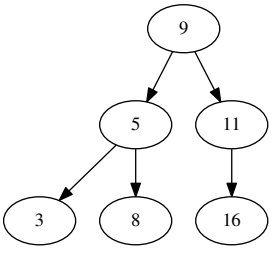

Figura 2: Imagen despues del delete