1.

#include<iostream>

#include<functional>

using namespace std;

// Arbol Binario

template<class T>

class NodeBB {

public:

    T element;

    NodeBB\* left;

    NodeBB\* right;

};

template<class T>

class TreeBB {

    typedef function<int(T, T)> Comp;

    NodeBB<T>\* root;

    Comp compare;

    void(\*procesar)(T);

private:

    bool \_insert(NodeBB<T>\*& node, T e) {

        if (node == nullptr) { //si es que  nodo se encuentra sin dirección de memoria, registra el valor

            node = new NodeBB<T>();

            node->element = e; //asigna valor en el elemento del nodo

            return true;

        }

        else

        {

            int n = compare(node->element, e); //se realiza la diferencia para conocer a que dirección irá el valor (izquierda o derecha)

            if (n == 0) return false;

            else if (n < 0) return \_insert(node->right, e); //verifica si el n es negativo, para que el recorrido vaya por la derecha, por ser mayor que el node->element

            else return \_insert(node->left, e); //verifica si el n es positivo, para que el recorrido vaya por la izquierda, por ser menor que el node->element

        }

    }

    bool \_search(NodeBB<T>\* node, T e) {

        if (node == nullptr) return false;

        else {

            int n = compare(node->element, e);

            if (n == 0)

                return true;

            else if (n < 0)

                return \_search(node->right, e);

            else

                return \_search(node->left, e);

        }

    }

    bool \_modify(NodeBB<T>\* node, T e, T value){

        if (node == nullptr) return false;

        else {

            int n = compare(node->element, e);

            if (n == 0)

            {

                cout<<node->element<<" = "<<value<<endl;

                node->element = value;

                return true;

            }

            else if (n < 0){

                return \_modify(node->right, e, value);

            }

            else {

                return \_modify(node->left, e, value);

            }

        }

    }

    bool \_null() {

        return root== nullptr;

    }

    int \_quantity(NodeBB<T>\* node) {

        if (node == nullptr) return 0;

        else {

            int ql = \_quantity(node->left);

            int qr = \_quantity(node->right);

            return ql + qr + 1;

        }

    }

    int \_maximum(NodeBB<T>\* node) {

        if (node->right == nullptr) return node->element;

        else return \_maximum(node->right);

    }

    int \_minimum(NodeBB<T>\* node) {

        if (node->left == nullptr) return node->element;

        else return \_minimum(node->left);

    }

    void \_enOrden(NodeBB<T>\* node) {

        if (node == nullptr) return;

        \_enOrden(node->left);

        procesar(node->element);

        \_enOrden(node->right);

    }

    void \_preOrden(NodeBB<T>\* node) {

        if (node == nullptr) return;

        procesar(node->element);

        \_preOrden(node->left);

        \_preOrden(node->right);

    }

public:

    TreeBB(void(\*value)(T)) {

        this->compare = [](T a, T b)->int {return a - b; }; //ingresar el "node->element" y el "e" para realizar la resta.

        this->procesar = value;

        root = nullptr;

    }

    bool insert(T e) {  return \_insert(root, e); } //envía la raiz y el elemento que ingresa

    bool search(T e) { return \_search(root, e); }

    bool modify(T e, T value){ return \_modify(root,e,value);}

    int quantity(T e) { return \_quantity(root); }

    int minimum() { return \_minimum(root); }

    int maximum() { return \_maximum(root); }

    void enOrden() { \_enOrden(root); }

    void preOrden() { \_preOrden(root); }

};

void print(int e)

{

    cout << " " << e;

}

// Arbol Ternario

template<class T>

class NodeTB {

public:

    T element;

    NodeTB\* left;

    NodeTB\* equ;

    NodeTB\* right;

};

template<class T>

class TreeTB {

    typedef function<int(T, T)> Comp;

    NodeTB<T>\* root;

    void(\*procesar)(T);

    Comp compare;

private:

    bool \_insert(NodeTB<T>\*& node, T e) {

        if (node == nullptr) {

            node = new NodeTB<T>();

            node->element = e;

            return true;

        }

        else

        {

            int n = compare(node->element, e);

            if (n < 0) return \_insert(node->right, e);

            else if (n == 0) return \_insert(node->equ, e);

            else return \_insert(node->left, e);

        }

    }

    bool \_search(NodeTB<T>\*& node, T e){

        if (node == nullptr) return false;

        else {

            int n = compare(node->element, e);

            if (n == 0)

                return true;

            else if (n < 0)

                return \_search(node->right, e);

            else

                return \_search(node->left, e);

        }

    }

    void \_preOrden(NodeTB<T>\* node) {

        if (node == nullptr) return;

        procesar(node->element);

        \_preOrden(node->left);

        \_preOrden(node->equ);

        \_preOrden(node->right);

    }

    void \_enOrden(NodeTB<T>\* node) {

        if (node == nullptr) return;

        \_enOrden(node->left);

        if (node->equ != nullptr) \_enOrden(node->equ);

        procesar(node->element);

        \_enOrden(node->right);

    }

    int \_quantity(NodeTB<T>\* node) {

        if (node == nullptr) return 0;

        else {

            int ql, qr, qe;

            ql = \_quantity(node->left);

            qr = \_quantity(node->right);

            qe = \_quantity(node->equ);

            return ql + qr + qe + 1;

        }

    }

public:

    TreeTB(void(\*value)(T)) {

        this->procesar = value;

        this->compare = [](T a, T b)->int {return a - b; };

        root = nullptr;

    }

    bool insert(T e) {

        return \_insert(root, e);

    }

    bool search(T e){

        return \_search(root,e);

    }

    void preOrden() {

        \_preOrden(root);

    }

    void enOrden() {

        \_enOrden(root);

    }

    int quantity() {

        return \_quantity(root);

    }

};

// main

int main() {

    TreeBB<int>\* treeBB = new TreeBB<int>(print);

    TreeTB<int>\* treeTB = new TreeTB<int>(print);

    // Ingrese la cantidad de elementos

    int n;

    cin>>n;

    while(n--){

        int x;

        cin>>x;

        treeBB->insert(x);

        treeTB->insert(x);

    }

    //Listado en Pre Orden del arbol binario

    cout<<"Listado en pre orden del arbol binario"<<endl;

    treeBB->preOrden();

    cout<<endl;

    //Listado en Pre Orden del arbol ternario

    cout<<"Listado en pre orden del arbol ternario"<<endl;

    treeTB->preOrden();

    cout<<"\n";

    // Ingrese cantidad de elementos

    int q;

    cin>>q;

    while(q--){

       int elemento;

       char Q;

       cin>>Q;

       if(Q == 'l')

       {

           cout<<"Listado en orden del arbol binario\n";

           treeBB->enOrden();

           cout<<"\nListado en orden del arbol ternario\n";

           treeTB->enOrden();

           cout<<"\n";

       }

       else if(Q == 'i'){

           int elemento;

           cout<<"\nIngrese el numero a ingresar en los arboles: ";

           cin>>elemento;

           treeTB->insert(elemento);

           treeBB->insert(elemento);

           cout<<"Ingresado correctamente...\n";

       }

       else if(Q=='u'){

           int newElemento;

           cout<<"Ingrese el elemento a modificar: ";

           cin>>elemento;

           cout<<"Ingrese el nuevo valor del elemento: ";

           cin>>newElemento;

           treeBB->modify(elemento, newElemento);

       }

       else if( Q == 's'){

           cout<<"Ingrese el elemento a buscar: ";

           cin>>elemento;

           treeTB->search(elemento)? cout<<"Se encontro\n":cout<<"No se encontro\n";

       }

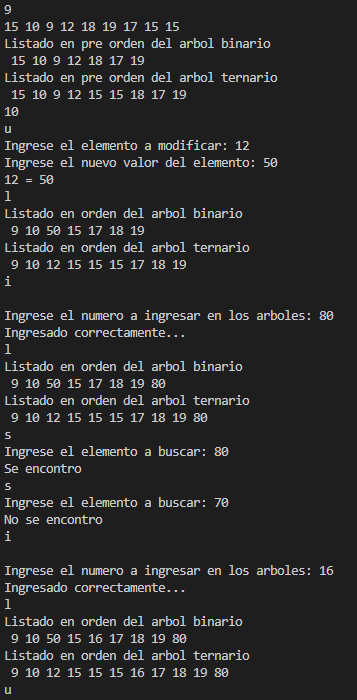
    }

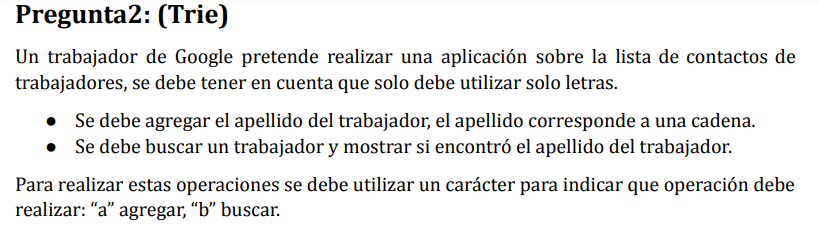
    cin.get();

    cin.ignore();

    return 0;

}



2.

#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

// Clase para representar un nodo del Trie

class TrieNode {

public:

    unordered\_map<char, TrieNode\*> children;

    bool isEndOfWord;

    TrieNode() {

        isEndOfWord = false;

    }

};

// Clase para el Trie

class Trie {

private:

    TrieNode\* root;

public:

    Trie() {

        root = new TrieNode();

    }

    TrieNode \* getTrieNode(){

        return root;

    }

    // Inserta una palabra en el Trie

    void insert(string word) {

        TrieNode\* current = root;

        //cout<<word<<endl;

        for (char c : word) {

            if (current->children.find(c) == current->children.end()) {

                //cout<<"Caracter: "<<c<<" "<<int(c)<<endl;

                current->children[c] = new TrieNode();

            }

            current = current->children[c];

        }

        current->isEndOfWord = true;

    }

    // Busca una palabra en el Trie

    bool search(string word) {

        TrieNode\* current = root;

        for (char c : word) {

            if (current->children.find(c) == current->children.end()) {

                return false;

            }

            current = current->children[c];

        }

        return current->isEndOfWord;

    }

    // Verifica si una palabra es un prefijo en el Trie

    bool startsWith(string prefix) {

        TrieNode\* current = root;

        int cnt = 0;

        for (char c : prefix) {

            if (current->children.find(c) == current->children.end()) {

                return false;

            }

            current = current->children[c];

        }

        return true;

    }

};

// Ejemplo de uso del Trie

int main() {

    vector<string> palabras;

    int n;

    do{

        cin>>n;

    }while(n < 1 && n > 200);

    while(n--){

        char ope;

        cin>>ope;

        string palabraBuscar;

        do{

            cin>>palabraBuscar;

        }while(palabraBuscar.size() > 18);

        //Insertar

        if(tolower(ope) == 'a'){

            palabras.push\_back(palabraBuscar);

        }

        //Buscar

        else if(tolower(ope) == 'b'){

            int cnt = 0;

            for (int i = 0; i < palabras.size(); i++)

            {

                Trie trieAux;

                trieAux.insert(palabras[i]);

                if(trieAux.startsWith(palabraBuscar))

                {

                    cnt++;

                }

            }

            cout<<"La cantidad de palabras que empiezan con "<<palabraBuscar<<" es: "<<cnt<<endl;

        }

        else{

            cout<<"Operacion invalida, ingrese A para insertar o B para buscar"<<endl;

        }

    }

    return 0;

}

