ініт дерева:

просто бінарне

Node\* makeTree(char arr[], int n, int level, string side){  
 if(n==0) return nullptr;  
 Node\*temp = new Node(arr[0], level, side);  
 int nl = n/2;  
 int nr = n-nl-1;  
 char arr1[nl], arr2[nr];  
 int c = 0;  
 for(int i = 1;i<=nl;i++){  
 arr1[i-1] = arr[i];  
 }  
 for(int i = nl+1;i<n;i++){  
 arr2[c++] = arr[i];  
 }  
 temp->left = makeTree(arr1, nl, level+1, "left");  
 temp->right = makeTree(arr2, nr, level+1, "right");  
 return temp;  
}

бінарне пошуку:

Node\* makeTree(char arr[], int n, int level){  
 if(n==0) return nullptr;  
 int middle = n/2;  
 Node\*temp = new Node(arr[middle], level);  
 int nl = 0, nr = 0;  
 for(int i = 0;i<n;i++){  
 if(arr[i]<arr[middle])  
 nl++;  
 else if(i!=middle)  
 nr++;  
 }  
 char arr1[nl], arr2[nr];  
 int ind1 = 0, ind2 = 0;  
 for(int i = 0;i<n;i++){  
 if(arr[i]<arr[middle])  
 arr1[ind1++] = arr[i];  
 else if(i!=middle)  
 arr2[ind2++] = arr[i];  
 }  
 temp->left = makeTree(arr1, nl, level+1);  
 temp->right = makeTree(arr2, nr, level+1);  
 return temp;  
}

прінт дерева:

прямий обхід

void Tree::print() {  
 Node\*current = root;  
 Stack stack;  
 while(current || !stack.isEmpty()){  
 while(current){  
 stack.push(current);  
  
 for(int i = 0;i<stack.back()->level;i++){  
 cout<<" ";  
 }  
 cout<<"|\_\_"<<stack.back()->data<<endl;  
  
 current = current->left;   
 }  
 current = stack.back()->right;   
 stack.pop();  
 }  
}

симетричний обхід

void Tree::print() {  
 Node\*current = root;  
 Stack stack;  
 while(current || !stack.isEmpty()){  
 while(current){  
 stack.push(current);  
 current = current->left;   
 }  
  
 for(int i = 0;i<stack.back()->level;i++){  
 cout<<" ";  
 }  
 cout<<"|\_\_"<<stack.back()->data<<endl;  
  
 current = stack.back()->right;   
 stack.pop();  
 }  
}

зворотній обхід

void Tree::print() {  
 Node\*current = root;  
 Stack stack;  
 while(current || !stack.isEmpty()){  
 while(current){  
 stack.push(current);  
 current = current->right;  
 }  
  
 for(int i = 0;i<stack.back()->level;i++){  
 cout<<" ";  
 }  
 cout<<"|\_\_"<<stack.back()->data<<endl;  
  
 current = stack.back()->left;  
 stack.pop();  
 }  
}

видалення елемента:

void Tree::erase(char a[], int size, char symb){  
 Node\*current = root;  
 Stack stack;  
 int count = 0;  
 while(current || !stack.isEmpty()){  
 while(current){  
 stack.push(current);  
  
 if(current->data != symb)  
 count++;  
  
 current = current->left;  
 }  
 current = stack.back()->right;  
 stack.pop();  
 }  
 char arr[count];  
 int c = 0;  
 for(int i = 0;i<size;i++){  
 if(a[i]!=symb)  
 arr[c++] = a[i];  
 }  
 root = makeTree(arr, count, 0, "root");  
}

к-сть входжень елемента:

void Tree::insertions(char symb){  
 Node\*current = root;  
 Stack stack;  
 string res = "";  
 int count = 0;  
 while(current || !stack.isEmpty()){  
 while(current){  
 stack.push(current);  
  
 if(current->data == symb){  
 count++;  
 res+="level: "+ to\_string(current->level)+"; side: "+current->side+"\n";  
 }  
  
 current = current->left;  
 }  
 current = stack.back()->right;  
 stack.pop();  
 }  
 cout<<"Symbol "<<symb<<":\n"<<"Insertions: "<<count<<";\n"<<res;  
}

скелет стека:

void Stack::push(Node \*temp) {  
 sNode \*t = new sNode;  
 t->data = temp;  
 t->prev = top;  
 top = t;  
}  
  
void Stack::pop() {  
 sNode \*t;  
 t = top;  
 if(t){  
 top = t->prev;  
 delete t;  
 }  
}  
  
Node\* Stack::back() {  
 return top->data;  
}  
  
bool Stack::isEmpty() {  
 return !top;  
}

скелет хедера **дерево:**

#include <iostream>  
using namespace std;  
struct Node{  
 char data;  
 int level;  
 string side;  
 Node\*left, \*right;  
 Node(char d, int l, string s):data(d), level(l), side(s), left(nullptr), right(nullptr){}  
};  
class Tree {  
 Node\*root;  
public:  
 Tree(Node\*t):root(t){}  
 void print();  
 void insertions(char);  
 void erase(char[], int, char);  
};  
Node\* makeTree(char[], int, int, string);  
  
struct sNode{  
 Node\* data;  
 sNode\* prev;  
};  
class Stack{  
 sNode\*top;  
public:  
 Stack():top(nullptr){}  
 void push(Node\*);  
 void pop();  
 Node\* back();  
 bool isEmpty();  
};

видалення ВСІХ повторюваних елементів:

void Tree::deleteNode(char arr[], int size){  
 string r = "";  
  
 for(int i = 0;i<size-1;i++){  
 bool d = false;  
 for(int j = i+1;j<size;j++){  
 bool ty = false;  
 for(char k : r){  
 if(arr[i] == k)  
 ty = true;  
 }  
 if(!ty){  
 if(arr[i]==arr[j])  
 d = true;  
 }  
 }  
 if(d){  
 r+=arr[i];  
 }  
 }  
 int c=0;  
 for(int i = 0;i<size;i++){  
 bool t = false;  
 for(char k : r){  
 if(arr[i]==k)  
 t = true;  
 }  
 if(!t)  
 c++;  
 }  
 char res[c];  
 int ind = 0;  
 for(int i = 0;i<size;i++){  
 bool t = false;  
 for(char k : r){  
 if(arr[i]==k)  
 t = true;  
 }  
 if(!t)  
 res[ind++] = arr[i];  
 }  
  
 root = makeTree(res, c, 0);  
}