

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
ТЕМА: БДП и ХЕШ-ТАБЛИЦЫ

Студент гр. 9381

Шахин Н.С

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить и реализовать такую структуру данных, как идеально сбалансированное бинарное дерево поиска. Написать функции проверки вхождения элемента, и добавления элемента в структуру.

Задание.

Вариант 21. БДП: Идеально сбалансированное; действие 1+2а.

- 1) По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;
- 2) Выполнить одно из следующих действий:
 - а) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если входит, то в скольких экземплярах. Добавить элемент e в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

Основные теоретические положения.

Идеально сбалансированным назовем такое бинарное дерево T , что для каждого его узла x справедливо соотношение $|n_L(x) - n_R(x)| \leq 1$, где $n_L(x)$ – количество узлов в левом поддереве узла x , а $n_R(x)$ – количество узлов в правом поддереве узла x .

Бинарное дерево называется бинарным деревом поиска, если для каждого его узла справедливо: все элементы правого поддерева больше этого узла, а все элементы левого поддерева – меньше этого узла.

Описание работы алгоритма.

Построение дерева.

Сначала данные сохраняются в массиве и сортируются в порядке возрастания, и считается длина массива без повторяющихся элементов, для того чтобы построить идеально сбалансированное дерево. Вызывается рекурсивный метод построения БДП. По формулам высчитывается сколько

элементов должно быть в левом и правом поддереве корня, функция вызывается для левого поддерева, потом записывается корень, затем функция вызывается для правого поддерева. Если элементы в массиве повторяются, то увеличивается поле `amount`, хранящее количество повторений.

Поиск в дереве.

Поиск в дереве производится рекурсивно функцией `find`. Поиск производится по принципу: если элемент равен значению корня, то он находится в корне дерева. Если элемент меньше значения корня, то он находится в левом поддереве элемента. Если элемент больше корня, то он находится в правом поддереве. Если выбранное поддерево оказывается пустым, то поиск завершается «неудачно» – элемента `x` в дереве `b` нет.

Добавление элемента в дерево.

Чтобы добавить элемент в дерево нужно перестроить его полностью, поэтому все элементы дерева сохраняются в порядке возрастания в массиве вместе с новым элементом, а затем строится новое дерево. Для того чтобы найти местоположение нового элемента в массиве, находим где этот элемент должен быть в БДП, и затем сохраняем в массив элементы до нового и после, а затем удаляем дерево и функцией `make tree` строим новое.

Функции и структуры данных.

Для хранения одного элемента дерева написан класс `Node`.

У класса есть поля:

`Node* right` – указатель на правое поддерево.

`int data` – данные в узле.

`int amount` – количество повторений элемента.

`Node* left` – указатель на левое поддерево.

Для работы с элементом дерева реализованы методы:

`Node* getLeft()` – возвращает указатель на левое поддерево.

`Node* getRight()` - возвращает указатель правое поддерево.

`int getData() const` – возвращает значение поля `data`.

`void setLeft(Node* l)` – присваивает значение полю `left`.

`void setRight(Node* r)` – присваивает значение полю `right`.

`void setData(int d)` – присваивает значение полю `data`.

`int getAmount() const` – возвращает количество повторений.

`void incAmount()` – увеличивает количество повторений на 1.

Для хранения дерева реализован класс `BinaryTree`.

Поля класса:

`Node* tree` – указатель на корень дерева.

Методы класса:

`explicit BinaryTree(std::vector<int>& arr)` – конструктор. В конструктор передаётся `arr` – массив типа `int`.

`void printTree(Node* node)` – функция для вывода дерева. В функцию передаётся указатель на элемент дерева `node` типа `Node*`.

`Node* getTree()` – функция возвращает указатель на корень дерева.

`~BinaryTree()` – деструктор класса.

`void findElem(int e)` – функция для поиска элемента, вызываемая пользователем. В функцию передаётся элемент для поиска.

`void addElem(int e)` – функция для добавления элемента, вызываемая пользователем. В функцию передаётся элемент для добавления.

`Node* makeTree(int n, std::vector<int>& arr, int& pos, int indent)` – функция для создания дерева. В функцию передаются: `int n` количество элементов, `std::vector<int>& arr` массив элементов, `int& pos` – текущая позиция в массиве, `int indent` – сдвиг для вывода промежуточных результатов.

`void destroy(Node*& buf)` – функция для удаления дерева и отчищения памяти. В функцию передаётся `Node*& buf` – указатель на элемент дерева по ссылке.

`void treeToArr(Node* node, std::vector<int>& arr, int e)` – функция для выгрузки элементов из части дерева с новым элементом в массив. В функцию передаются: `Node* node` – указатель на элемент дерева, `std::vector<int>& arr` – массив для сохранения элементов, `int e` – элемент для добавления.

`void treeToArrHelp(Node* node, std::vector<int>& arr)` – функция для выгрузки элементов из части дерева без нового элемента в массив. В функцию передаются: `Node* node` – указатель на элемент дерева, `std::vector<int>& arr` – массив для сохранения элементов.

`bool find(Node* node, int x)` – функция для поиска элемента в массиве. В функцию передаются: `Node* node` – указатель на корень дерева, `int x` – элемент для поиска.

Функции для считывания данных.

`int makeArr(int option, std::vector<int>& arr)` – функция для считывания массива. Функция возвращает код ошибки или 0, если считывание прошло успешно. В функцию передаются: `int option` – выбор считывания из консоли или файла, `std::vector<int>& arr` – массив целых чисел по ссылке.

`int getNum(string& input)` – функция для преобразования строки в число. Функция возвращает число. Функция принимает: `string& input` – строку по ссылке.

`void skip (string& str, int n = 1)` – функция для удаления первых `n` элементов строки. В функцию передаются: `string& str` – строка по ссылке, `int n` – количество символов, которые надо удалить.

Тестирование программы.

№	Входные данные	Выходные данные
1	1 3 5 5 7 9 9 9 11 13	Choose console or file: 1 - Console 2 - File Your choice: 1 Write a statement: 1 3 5 5 7 9 9 9 11 13 Your input: 1 3 5 5 7 9 9 9 11 13

		<p>Sorting array</p> <p>1 3 5 5 7 9 9 9 11 13</p> <p>Making tree</p> <p>amount of elements: 7</p> <p>3 Elements goes to the left</p> <p>3 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 3</p> <p>1 Elements goes to the left</p> <p>1 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>1 is a root</p> <p>Right</p> <p>()</p> <p>3 is a root</p> <p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>5 is a root</p> <p>Right</p> <p>()</p> <p>7 is a root</p> <p>Right</p> <p>amount of elements: 3</p> <p>1 Elements goes to the left</p> <p>1 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p>
--	--	--

		<p>Left</p> <p>()</p> <p>9 is a root</p> <p>Right</p> <p>()</p> <p>11 is a root</p> <p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>13 is a root</p> <p>Right</p> <p>()</p> <p>-----</p> <p>1 - Find Element</p> <p>2 - Add Element</p> <p>3 - Print tree</p> <p>4 - Exit</p> <p>Your choice 1</p> <p>Enter element: 9</p> <p>find element 9</p> <p>9 > 7 Go to the right</p> <p>9 < 11 Go to the left</p> <p>Elem 9 appears in tree 3 times</p> <p>-----</p> <p>1 - Find Element</p> <p>2 - Add Element</p> <p>3 - Print tree</p> <p>4 - Exit</p> <p>Your choice 2</p> <p>Enter element: 4</p> <p>Add elem 4</p> <p>rebuild tree</p>
--	--	--

		<p>making array</p> <p>1 goes to array</p> <p>3 goes to array</p> <p>New elem 4 goes to array</p> <p>5 goes to array</p> <p>7 goes to array</p> <p>9 goes to array</p> <p>11 goes to array</p> <p>13 goes to array</p> <p>amount of elements: 8</p> <p>4 Elements goes to the left</p> <p>3 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 4</p> <p>2 Elements goes to the left</p> <p>1 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 2</p> <p>1 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>1 is a root</p> <p>Right</p> <p>()</p> <p>3 is a root</p> <p>Right</p> <p>()</p> <p>4 is a root</p> <p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p>
--	--	--

		<p>()</p> <p>5 is a root</p> <p>Right</p> <p>()</p> <p>7 is a root</p> <p>Right</p> <p>amount of elements: 3</p> <p>1 Elements goes to the left</p> <p>1 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>9 is a root</p> <p>Right</p> <p>()</p> <p>11 is a root</p> <p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>13 is a root</p> <p>Right</p> <p>()</p> <p>-----</p> <p>1 - Find Element</p> <p>2 - Add Element</p> <p>3 - Print tree</p> <p>4 - Exit</p> <p>Your choice 3</p> <p>7(4(3(100)0)(500))(11(900)(1300))</p> <p>-----</p> <p>1 - Find Element</p>
--	--	--

		2 - Add Element 3 - Print tree 4 - Exit Your choice 4 end of program
2	5 2 4 -3 7 9 12 0	Choose console or file: 1 - Console 2 - File Your choice: 1 Write a statement: 5 2 4 -3 7 9 12 0 Your input: 5 2 4 -3 7 9 12 0 Sorting array -3 0 2 4 5 7 9 12 Making tree amount of elements: 8 4 Elements goes to the left 3 Elements goes to the right Left amount of elements: 4 2 Elements goes to the left 1 Elements goes to the right Left amount of elements: 2 1 Elements goes to the left 0 Elements goes to the right Left amount of elements: 1 0 Elements goes to the left 0 Elements goes to the right Left () -3 is a root Right () 0 is a root Right () 2 is a root

		<p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>4 is a root</p> <p>Right</p> <p>()</p> <p>5 is a root</p> <p>Right</p> <p>amount of elements: 3</p> <p>1 Elements goes to the left</p> <p>1 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>7 is a root</p> <p>Right</p> <p>()</p> <p>9 is a root</p> <p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>12 is a root</p> <p>Right</p> <p>()</p> <p>-----</p> <p>1 - Find Element</p> <p>2 - Add Element</p> <p>3 - Print tree</p>
--	--	---

	<p>4 - Exit</p> <p>Your choice 3</p> <p>5(2(0(-3(0(0)0)(4(0(0)))(9(7(0(0))(12(0(0))))</p> <p>-----</p> <p>1 - Find Element</p> <p>2 - Add Element</p> <p>3 - Print tree</p> <p>4 - Exit</p> <p>Your choice 1</p> <p>Enter element: 1</p> <p>find element 1</p> <p>1 < 5 Go to the left</p> <p>1 < 2 Go to the left</p> <p>1 > 0 Go to the right</p> <p>No element 1 in bst</p> <p>-----</p> <p>1 - Find Element</p> <p>2 - Add Element</p> <p>3 - Print tree</p> <p>4 - Exit</p> <p>Your choice 2</p> <p>Enter element: 1</p> <p>Add elem 1</p> <p>rebuild tree</p> <p>making array</p> <p>-3 goes to array</p> <p>0 goes to array</p> <p>New elem 1 goes to array</p> <p>2 goes to array</p> <p>4 goes to array</p> <p>5 goes to array</p> <p>7 goes to array</p> <p>9 goes to array</p> <p>12 goes to array</p> <p>amount of elements: 9</p> <p>4 Elements goes to the left</p>
--	--

		<p>4 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 4</p> <p>2 Elements goes to the left</p> <p>1 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 2</p> <p>1 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>-3 is a root</p> <p>Right</p> <p>()</p> <p>0 is a root</p> <p>Right</p> <p>()</p> <p>1 is a root</p> <p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>2 is a root</p> <p>Right</p> <p>()</p> <p>4 is a root</p> <p>Right</p> <p>amount of elements: 4</p> <p>2 Elements goes to the left</p> <p>1 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 2</p>
--	--	---

		<p>1 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>5 is a root</p> <p>Right</p> <p>()</p> <p>7 is a root</p> <p>Right</p> <p>()</p> <p>9 is a root</p> <p>Right</p> <p>amount of elements: 1</p> <p>0 Elements goes to the left</p> <p>0 Elements goes to the right</p> <p>Left</p> <p>()</p> <p>12 is a root</p> <p>Right</p> <p>()</p> <p>-----</p> <p>1 - Find Element</p> <p>2 - Add Element</p> <p>3 - Print tree</p> <p>4 - Exit</p> <p>Your choice 1</p> <p>Enter element: 1</p> <p>find element 1</p> <p>1 < 4 Go to the left</p> <p>Elem 1 appears in tree 1 times</p> <p>-----</p>
--	--	--

		1 - Find Element 2 - Add Element 3 - Print tree 4 - Exit Your choice 4 end of program
3	4 5 ab 8	Choose console or file: 1 - Console 2 - File Your choice: 1 Write a statement: 4 5 ab 8 error not integer in array

Вывод.

Была реализована и изучена такая структура данных, как идеально сбалансированное дерево поиска. Были написаны функции проверки вхождения элемента, и добавления элемента в структуру.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл structs.h

```
#ifndef AISD_LB5_STRUCTS_H
#define AISD_LB5_STRUCTS_H

#include <iostream>
#include <algorithm>
#include <vector>
#include <stack>
#include <string>
#include <cstring>
#include <unistd.h>
#include <fstream>

using namespace std;

#endif //AISD_LB5_STRUCTS_H
```

Файл main.cpp

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

int main() {
    string choice;
    cout<<"Choose console or file:\n 1 - Console\n 2 - File\nYour choice: ";
    getline(cin, choice);
    int ch = 1;
    switch (choice[0]) {
        case '1':
            ch = 1;
            break;
        case '2':
            ch = 2;
            break;
    }
    std::vector<int> arr;
    if(makeArr(arr, ch))
        return 1;
    BinaryTree* tree = new BinaryTree(arr);
    int c = 0;
    while (c!=4){
        cout<<"\n-----\n1          -          Find
Element\n2 - Add Element\n3 - Print tree\n4 - Exit\nYour choice ";
        cin>>c;
        int elem;
        switch (c) {
```



```

        case 1:
            cout<<"Enter element: ";
            cin>>elem;
            tree->findElem(elem);
            break;
        case 2:
            cout<<"Enter element: ";
            cin>>elem;
            tree->addElem(elem);
            break;
        case 3:
            tree->printTree(tree->getTree());
            break;
        case 4:
            cout<<"end of program"<<endl;
            break;
    }
}
delete tree;
return 0;
}

```

Файл InOut.h

```

#ifndef AISD_LB5_INOUT_H
#define AISD_LB5_INOUT_H

#include "structs.h"

int makeArr( std::vector<int>& arr, int option);
int getNum(string& input);
void skip (string& str, int n = 1);

#endif //AISD_LB5_INOUT_H

```

Файл InOut.cpp

```

#include "InOut.h"

int makeArr( std::vector<int>& arr, int option){
    if(option == 1) {
        cout<<"Write a statement: ";
        string res;
        getline(cin, res);
        while (!res.empty()){
            if(!(isdigit(res[0]) || res[0] == '-')){
                cout<<" error not integer in array\n";
                arr.clear();
                return 1;
            }
            int n = getNum(res);
            arr.push_back(n);
            skip(res,1);
        }
        return 0;
    } else {
        string filename;
        cout<<"filename: ";
    }
}

```

```

        cin>> filename;
        ifstream infile(filename);
        if (!infile) {
            cout << "> File can't be open!" << endl;
            return 1;
        }
        string str;
        string num;
        getline(infile, str);
        while (!str.empty()) {
            if (!(isdigit(str[0]) || str[0] == '-')) {
                cout << "not integer in array\n";
                arr.clear();
                return 1;
            }
            int n = getNum(str);
            arr.push_back(n);
            skip(str, 1);
        }
        return 0;
    }
}

int getNum(string& str){
    string strNum;
    while (isdigit(str[0]) || (strNum.length() == 0 && str[0] == '-'))
    {
        strNum += str[0];
        skip(str, 1);
    }
    return stoi(strNum);
}

void skip(string& str, int n){
    if (str.length() >= n) {
        str = str.substr(n);
    }
}

```

Файл Node.h

```

#ifndef AISD_LB5_NODE_H
#define AISD_LB5_NODE_H

class Node {
private:
    Node* right;
    int data;
    int amount;
    Node* left;
public:
    Node(): right(nullptr), data(0), left(nullptr), amount(0){}

    Node* getLeft();

    Node* getRight();

```

```

    int getData() const;

    int getAmount() const;

    void incAmount();

    void setLeft(Node* l);

    void setRight(Node* r);

    void setData(int d);

};

```

```

#endif //AISD_LB5_NODE_H

```

Файл Node.cpp

```

#include "Node.h"

int Node::getData() const {
    return data;
}

Node* Node::getLeft() {
    return left;
}

Node* Node::getRight() {
    return right;
}

int Node::getAmount() const {
    return amount;
}

void Node::setData(int d) {
    data = d;
}

void Node::setLeft(Node *l) {
    left = l;
}

void Node::setRight(Node *r) {
    right = r;
}

void Node::incAmount() {
    amount++;
}

```

Файл Tree.h

```

#ifndef AISD_LB5_TREE_H
#define AISD_LB5_TREE_H

```

```

#include "Node.h"
#include "structs.h"

class BinaryTree{
public:
    explicit BinaryTree(std::vector<int>& arr);
    void printTree(Node* node);
    Node* getTree();
    ~BinaryTree();
    void findElem(int e);
    void addElem(int e);
private:
    Node* tree;
    Node* makeTree(int n, std::vector<int>& arr, int& pos, int indent);
    void destroy(Node*& buf);
    void treeToArr(Node* node, std::vector<int>& arr, int e);
    void treeToArrHelp(Node* node, std::vector<int>& arr);
    bool find(Node* node, int x);

};

#endif //AISD_LB5_TREE_H

```

Файл Tree.cpp

```

#include "Tree.h"

BinaryTree::BinaryTree(std::vector<int> &arr) {
    cout<<"Your input: ";
    for(int i : arr){
        cout<<i<<' ';
    }
    cout<<endl;
    cout<<"Sorting array"<<endl;
    sort(arr.begin(), arr.end());
    for(int i : arr){
        cout<<i<<' ';
    }
    cout<<endl;
    int countUnique = 1;
    int pos = 0;
    for(int i = 0; i < arr.size()-1; i++){
        if(arr[i]<arr[i+1]){
            countUnique++;
        }
    }
    cout<<"Making tree"<<endl;
    tree = makeTree(countUnique, arr, pos, 0);
}

Node * BinaryTree::getTree() {
    return tree;
}

```

```

Node * BinaryTree::makeTree(int n, std::vector<int>& arr, int& pos, int
indent) {
    if (n == 0) {
        for(int i = 0; i < indent; i++){cout<<" ";}
        cout<<"()"<<endl;
        return nullptr;
    }
    for(int i = 0; i < indent; i++){cout<<" ";}
    cout<<"amount of elements: "<< n<<endl;
    Node *buf = new Node();
    int nL, nR;
    nL = n / 2;
    for(int i = 0; i < indent; i++){cout<<" ";}
    cout<<nL<<" Elements goes to the left"<<endl;
    nR = n - nL - 1;
    for(int i = 0; i < indent; i++){cout<<" ";}
    cout<<nR<<" Elements goes to the right"<<endl;
    for(int i = 0; i < indent+1; i++){cout<<" ";}
    cout<<"Left"<<endl;
    buf->setLeft(makeTree(nL, arr, pos, indent+1));
    for(int i = 0; i < indent; i++){cout<<" ";}
    cout<<arr[pos]<<" is a root"<<endl;
    buf->setData(arr[pos]);
    buf->incAmount();
    while (arr[pos] == arr[pos + 1]) {
        pos++;
        buf->incAmount();
    }
    pos++;
    for(int i = 0; i < indent+1; i++){cout<<" ";}
    cout<<"Right"<<endl;
    buf->setRight(makeTree(nR, arr, pos, indent+1));
    return buf;
}

void BinaryTree::printTree(Node* node) {
    if(node == nullptr)
        return;
    cout<<node->getData();
    cout<<'(';
    printTree(node->getLeft());
    cout<<')';
    cout<<'(';
    printTree(node->getRight());
    cout<<')';
}

void BinaryTree::destroy(Node*& buf){
    if(buf != nullptr) {
        Node* left = buf->getLeft();
        Node* right = buf->getRight();
        destroy(left);
        destroy(right);
        delete buf;
    }
}

BinaryTree::~BinaryTree() {
    destroy(tree);
}

```

```

void BinaryTree::findElem(int e) {
    cout<<"find element "<< e<<endl;
    find(tree, e);
}

bool BinaryTree::find(Node *node, int x) {
    if(node == nullptr) {
        cout<<"\nNo element "<<x<<" in bst"<<endl;
        return false;
    }
    if(x == node->getData()){
        cout<<"\nElem "<<x<<" appears in tree "<<node->getAmount()<<"
times"<<endl;
        return true;
    }else if(x < node->getData()){
        cout<<x<<" < "<<node->getData()<<" Go to the left"<<endl;
        if(find(node->getLeft(), x))
            return true;
    } else if(x > node->getData()) {
        cout<<x<<" > "<<node->getData()<<" Go to the right"<<endl;
        if(find(node->getRight(), x))
            return true;
    }
    return false;
}

void BinaryTree::treeToArrHelp(Node *node, std::vector<int>& arr) {
    if(node != nullptr){
        treeToArrHelp(node->getLeft(), arr);
        cout<<node->getData()<<" goes to array"<<endl;
        for(int i = 0; i<node->getAmount(); i++){
            arr.push_back(node->getData());
        }
        treeToArrHelp(node->getRight(), arr);
    }
}

void BinaryTree::treeToArr(Node *node, std::vector<int> &arr, int e) {
    if(node == nullptr){
        arr.push_back(e);
        cout<<"New elem "<<e<<" goes to array"<<endl;
    }
    else if(e < node->getData()){
        treeToArr(node->getLeft(), arr, e);
        cout<<node->getData()<<" goes to array"<<endl;
        for(int i = 0; i<node->getAmount(); i++){
            arr.push_back(node->getData());
        }
        treeToArrHelp(node->getRight(), arr);
    }
    else if(e > node->getData()){
        treeToArrHelp(node->getLeft(), arr);
        cout<<node->getData()<<" goes to array"<<endl;
        for(int i = 0; i<node->getAmount(); i++){
            arr.push_back(node->getData());
        }
        treeToArr(node->getRight(), arr, e);
    }
}

```

```

    } else if(e == node->getData()){
        treeToArrHelp(node->getLeft(), arr);
        node->incAmount();
        cout<<node->getData()<<" goes to array"<<endl;
        for(int i = 0; i<node->getAmount(); i++){
            arr.push_back(node->getData());
        }
        treeToArrHelp(node->getRight(), arr);
    }
}

void BinaryTree::addElem(int e) {
    cout<<"Add elem "<< e<<endl;
    cout<<"rebuild tree"<<endl;
    std::vector<int> arr;
    cout<<"making array"<<endl;
    treeToArr(tree, arr, e);
    destroy(tree);
    int countUnique = 1;
    int pos = 0;
    for(int i = 0; i < arr.size()-1; i++){
        if(arr[i]<arr[i+1]){
            countUnique++;
        }
    }
    tree = makeTree(countUnique, arr, pos, 0);
}

```