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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья.

Студентка гр. 9304 _____ Паутова Ю.В.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с понятием бинарного дерева, получить навыки программирования функций для работы с бинарными деревьями. Реализовать бинарное дерево на языке программирования C++.

Задание.

Вариант 1

Задано бинарное дерево b типа BT с типом элементов Elem. Для введенной пользователем величины E (var E: Elem):

- определить, входит ли элемент E в дерево b;
- определить число вхождений элемента E в дерево b;
- найти в дереве b длину пути (число ветвей) от корня до ближайшего узла с элементом E (если E не входит в b, за ответ принять –1).

Описание алгоритма работы.

Программе с помощью аргументов командной строки подаются два файла: файл1, содержащий строку-список и элемент, и файл2, в который будет записан результат работы программы. Сначала проверяем были ли переданы программе файлы, если да, то проверяем, открылись ли они для считывания/записи. Если оба файла были открыты, тогда создаются: объект класса string: tree, объект класса BinTree: bin_tree и переменная типа Elem(char): E. Строка-дерево записывается в tree, после чего проверяется на корректность функцией isCorrect().

Если строка-дерево корректна, то вызывается метод класса BinTree read_tree(), который создает бинарное дерево с помощью метода read_node, который рекурсивно пробегается по строке-дереву tree и создает узел.

Затем считывается элемент и записывается в переменную E, которая затем передается в метод класса BinTree set_E(), который заполняет поля E и number_E. Файл1 закрывается.

После вызывается метод класса BinTree back_tracking(), который рекурсивно пробегается по бинарному дереву ЛКП обходом и сравнивает

значения узлов с заданным значением и вычисляет пути до узлов. Если значения совпали, то значение переменной number_E увеличивается, а длина пути до этого узла записывается в вектор paths.

Если значение поля number_e > 0 в файл2 записывается строка: “<E> is found (<number_E> times)\nPath length = <bin_tree.get_min_path()>”, иначе строка: ““<E> isn’t found\nPath length = -1”. Файл2 закрывается и программа завершается.

Разработанный программный код см. в приложении А.

Формат входных и выходных данных.

Входные данные представлены в виде строки, которая является скобочной записью бинарного дерева, и элемента, наличие которого в бинарном дереве нужно проверить. Визуализацию бинарного дерева см. в приложение С.

Выходные данные представлены в виде строки, в которой говорится: был найдем заданный элемент или нет, сколько раз заданный элемент был найден, какова длина кратчайшего пути до заданного элемента.

Описание основных структур данных и функций.

Структуры данных:

- Class BinTreeNode – узел бинарного дерева.
    - Поле std::shared_ptr<BinTreeNode> left – указатель на левое поддерево.
    - Поле std::shared_ptr<BinTreeNode> right – указатель на правое поддерево.
    - Поле Elem data – значение, лежащее в узле.
- Class BinTree – реализация бинарного дерева
    - Поле Elem E– хранит значение элемента, наличие которого нужно проверить.
    - Поле int number_E – хранит количество вхождений элемента в бинарное дерево.

- Поле int path – хранит длину пути до узла.
- Поле std::vector<int> paths – хранит длины путей до каждого найденного заданного элемента.
- Поле std::shared_ptr<BinTreeNode> head – указатель на корень бинарного дерева.
- BinTree(BinTree& other) – конструктор копирования.
- BinTree(BinTree&& other) – конструктор перемещения.
- BinTree& operator=(BinTree& other) – оператор копирования.
- BinTree& operator=(BinTree& other) – оператор перемещения.
- Метод set_E() – принимает переменную типа Elem(char) и передает ее значение полю E, полю number_E значение 0.
- Метод get_number_E() – возвращает значение поля number_E.
- Метод get_min_path() – возвращает длину кратчайшего пути до заданного элемента.
- Метод get_head() – возвращает указатель на корень бинарного дерева.
- Метод copy() – копирует бинарное дерево и возвращает указатель на корень.
- Метод read_node() – принимает ссылку на объект класса string, создает узел.
- Метод read_tree() – принимает ссылку на объект класса string, создает бинарное дерево.
- Метод printBinTree() – принимает std::shared_ptr<BinTreeNode>, выводит бинарное дерево в скобочной записи.
- Метод back_tracking() – принимает указатель на объект класса BinTreeNode, выполняет ЛКП обход бинарного дерева.

- Метод insert() – принимает элемент для вставки и создает узел, хранящий данное значение.

Функции:

- bool isCorrect() – принимает ссылку на объект класса string, и проверяет корректность строки со скобочной записью бинарного дерева.

Тестирование.

Тестирование происходит с помощью bash-скрипта. Он с помощью команд терминала запускает программу, подавая на вход файлы с тестами из директории Tests, и выводит результат. Также запускает программу без указания файлов и с указанием несуществующего файла test7.txt. Для запуска тестирования в консоли используется команда *make run_tests*.

Результаты тестирования см. в приложении В.

Выводы.

Произошло ознакомление с понятием бинарного дерева, были получены навыки программирования функций для работы с бинарными деревьями. Было реализовано бинарное дерево на языке программирования C++.

Была разработана программа, считывающая скобочную запись бинарного дерева, создающая на её основе бинарное дерево и проверяющая бинарное дерево на вхождение заданного элемента. Проведено тестирование работы программы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: **main.cpp**

```
#include <iostream>
#include <fstream>
#include <string>
#include <stack>

#include "BinTree.h"

bool isCorrect(std::string tree);

int main(int argc, char** argv){
    Elem E;
    std::string tree;
    BinTree bin_tree;
    if (argc < 3){
        std::cout << "BinTree = ";
        getline(std::cin, tree);
        std::cout << "Element to search for: ";
        std::cin >> E;

    }
    else{
        std::ifstream in(argv[1]);
        if (!in.is_open()){
            std::cout << argv[1] << ": No such file or directory" << s
td::endl;
            return 1;
        }
        std::string string;
        while(getline(in, string)){
            if (string.find('(') != std::string::npos){
                tree = string;
            }
            else{
                if (string.length() > 24){
                    std::cout << "More than 1 search item entered." <<
 std::endl;
                    in.close();
                    return 1;
                }
                else{
                    E = string[string.length()-1];
                }
            }
        }
        in.close();
    }
    if (!isCorrect(tree)){
        std::cout << "The bintree isn't correct." << std::endl;
        return 1;
    }
    bin_tree.set_E(E);
    bin_tree.read_tree(tree);
```

```
    bin_tree.back_tracking(bin_tree.get_head());

    if (argc == 3){
        std::ofstream out(argv[2]);
        if (!out.is_open()){
            std::cout << argv[2] << ": No such file or directory" << s
td::endl;
            return 1;
        }
        if (bin_tree.get_number_E()){
            out << E <<" is found (" << bin_tree.get_number_E() << " t
imes)\n";
            out << "path length = " << bin_tree.get_min_path() << "\n";
        }
        else{
            out << E <<" isn't found\n";
            out << "path length = -1\n";
        }
        out.close();
    }
    else{
        if (bin_tree.get_number_E()){
            std::cout << E <<" is found (" << bin_tree.get_number_E()
<< " times)\n";
            std::cout << "path length = " << bin_tree.get_min_path();
        }
        else{
            std::cout << E <<" isn't found\n";
            std::cout << "path length = -1\n";
        }
    }
    return 0;
}

bool isCorrect(const std::string& tree){
    if (tree[0] != '('){
        return false;
    }
    std::stack<char> Stack;
    for (char i : tree){
        if (i == '('){
            Stack.push(i);
        }
        if (i == ')'){
            if (Stack.empty()){
                return false;
            }
            Stack.pop();
        }
    }
    if (Stack.empty()){
        if (tree.find("((") == std::string::npos)
            return true;
    }
    return false;
}
```

Название файла: **BinTreeNode.h**

```
#ifndef BINTREENODE_H
#define BINTREENODE_H
#include <memory>

typedef char Elem;

class BinTreeNode{
public:
    Elem data;
    std::shared_ptr<BinTreeNode> left {nullptr};
    std::shared_ptr<BinTreeNode> right {nullptr};
};

#endif
```

Название файла: **BinTree.h**

```
#ifndef BINTREE_H
#define BINTREE_H

#include <string>
#include <stack>
#include <vector>
#include <memory>
#include <iostream>

#include "BinTreeNode.h"

class BinTree{
public:
    BinTree() = default;
    ~BinTree() = default;

    BinTree(BinTree&& other);
    BinTree& operator=(BinTree&& other);

    BinTree(BinTree& other);
    BinTree& operator=(BinTree& other);

    void set_E(Elem E);

    void read_tree(std::string& tree);

    void printBinTree(std::shared_ptr<BinTreeNode> cur);
    void back_tracking(std::shared_ptr<BinTreeNode> cur); //обход ЛКП
    void insert(Elem data_to_insert);

    std::shared_ptr<BinTreeNode> get_head();
    int get_number_E();
    int get_min_path();

private:
    std::shared_ptr<BinTreeNode> copy(std::shared_ptr<BinTreeNode> cur
);
```

```
    std::shared_ptr<BinTreeNode> read_node(std::string& tree);
    Elem E;
    int number_E;
    int path;
    std::vector<int> paths;
    std::shared_ptr<BinTreeNode> head;
};

#endif
```

Название файла: **BinTree.cpp**

```
#include "BinTree.h"

BinTree::BinTree(BinTree&& other){
    std::swap(other.head, this->head);
}
BinTree& BinTree::operator=(BinTree&& other){
    if (&other != this)
        this->head = std::move(other.head);
    return *this;
}

BinTree::BinTree(BinTree& other){
    this->head = copy(other.head);
}
BinTree& BinTree::operator=(BinTree& other){
    if (&other != this)
        this->head = copy(other.head);
    return *this;
}
std::shared_ptr<BinTreeNode> BinTree::copy(std::shared_ptr<BinTreeNode
> cur){
    if (cur){
        std::shared_ptr<BinTreeNode> node = std::make_shared<BinTreeNo
de>();
        node->left = copy(cur->left);
        node->right = copy(cur->right);
        node->data = cur->data;
        return node;
    }
    return nullptr;
}

void BinTree::set_E(Elem E){
    this->E = E;
    this->path = 0;
    this->number_E = 0;
}

void BinTree::read_tree(std::string& tree){
    this->head = read_node(tree);
}

std::shared_ptr<BinTreeNode> BinTree::read_node(std::string& tree){
```

```
    std::shared_ptr<BinTreeNode> node = std::make_shared<BinTreeNode>(
);
    if(tree.length() < 3){
        return nullptr;
    }
    if (tree.length() == 3){
        node->left = nullptr;
        node->right = nullptr;
        node->data = tree[1];
        return node;
    }
    std::stack<Elem> Stack;
    int index = 2;
    int cur_index = index;
    while (cur_index < (int)tree.length()){
        if (tree[cur_index] == '('){
            Stack.push(tree[cur_index]);
         }
        if (tree[cur_index] == ')'){
            if (Stack.empty()){
                break;
            }
            Stack.pop();
        }
        cur_index++;
        if (Stack.empty()){
            cur_index--;
            break;
        }
    }
    std::string left, right;
    left.insert(0, tree, index, cur_index-index+1);
    right.insert(0, tree, cur_index+1, tree.length()-2-cur_index);
    node->left = read_node(left);
    node->right = read_node(right);
    node->data = tree[1];
    return node;
}

void BinTree::printBinTree(std::shared_ptr<BinTreeNode> cur){
    std::cout << '(';
    if (cur){
        std::cout << cur->data;
        printBinTree(cur->left);
        printBinTree(cur->right);
    }
    std::cout << ')';
}

void BinTree::back_tracking(std::shared_ptr<BinTreeNode> cur){ //обход
 ЛКП
    if (cur){
        this->path += 1;
        back_tracking(cur->left);
        //std::cout << cur->data << " path = " << this->path << "\n";
        if (cur->data == this->E){
            this->number_E++;
            this->paths.push_back(this->path);
```

```
        }
        this->path += 1;
        back_tracking(cur->right);
    }
    //else{
        this->path -= 1; //std::cout << "() ";
    //}
}

void BinTree::insert(Elem data_to_insert){
    if (!this->head){
        this->head = std::make_shared<BinTreeNode>();
        this->head->data = data_to_insert;
        return;
    }
    std::shared_ptr<BinTreeNode> node_to_insert {this->head};
    while(true){
        if(data_to_insert <= node_to_insert->data){
            if(!node_to_insert->left){
                node_to_insert->left = std::make_shared<BinTreeNode>();
                node_to_insert->left->data = data_to_insert;
                node_to_insert->left->left = node_to_insert->left-
>right = nullptr;
                break;
            }
            else{
                node_to_insert = node_to_insert->left;
                continue;
            }
        }
        else{
            if(!node_to_insert->right){
                node_to_insert-
>right = std::make_shared<BinTreeNode>();
                node_to_insert->right->data = data_to_insert;
                node_to_insert->right->left = node_to_insert->right-
>right = nullptr;
                break;
            }
            else{
                node_to_insert = node_to_insert->right;
                continue;
            }
        }
    }
}

std::shared_ptr<BinTreeNode> BinTree::get_head(){
    return this->head;
}
int BinTree::get_number_E(){
    return this->number_E;
}
int BinTree::get_min_path(){
    int min_path = this->paths.back();
    this->paths.pop_back();
    while(!this->paths.empty()){
        if (min_path > this->paths.back()){
```

```
            min_path = this->paths.back();
        }
        this->paths.pop_back();
    }
    return min_path;
}
```

Название файла: **Makefile**

```
lab3: main.o BinTree.o
    g++ -std=c++17 main.o BinTree.o -o lab3

main.o: source/main.cpp source/BinTree.h
    g++ -Wall -std=c++17 -c source/main.cpp

BinTree.o: source/BinTree.cpp source/BinTree.h source/BinTreeNode.h
    g++ -Wall -std=c++17 -c source/BinTree.cpp

run_tests:
    ./myscript

clean:
    rm -rf *.o
```

Название файла: **./myscript**

```
#!/bin/bash
for n in {1..6}
do
    arg=$(cat Tests/test$n.txt)
    echo -e "\nTest $n"
    echo "list = $arg"
    ./lab3 Tests/test$n.txt result$n
    res=$(find . -name “result$n”)
    if [ "$res" \> "" ]
    then
    cat result$n
    fi
done
echo -e "\nTest 7\nВеденная команда: ./lab3"
./lab3
echo -e "\nTest 8"
./lab3 Tests/test8.txt result8
echo –e
rm result*
```

ПРИЛОЖЕНИЕ В
ТЕСТИРОВАНИЕ

Таблица В.1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарии
1	(a(b(d)(e()(g)))(c()(f))) Element to search for: e	e is found (1 times) path length = 2	Корректная запись бинарного дерева, которое содержит заданный элемент; число вхождений элемента в дерево равно одному; длина пути до ближайшего узла с заданным элементом равна 2
2	(6(0(4(2)(6))(2(1)(4(3)(5))))(2(8(7)(0))(4(3)(5)))) Element to search for: 2	2 is found (3 times) path length = 1	Корректная запись бинарного дерева, которое содержит заданный элемент; число вхождений элемента в дерево равно трем; длина пути до ближайшего узла с заданным элементом равна 1

3	(b(d)(e()(g))) Element to search for: a f	More than 1 search item entered.	Корректная запись бинарного дерева; некорректный ввод элемента для поиска
4	(0(1(2)(3(4)(5)))(6(7(8)(9)))) Element to search for: 0	0 is found (1 times) path length = 0	Корректная запись бинарного дерева, которое содержит заданный элемент; число вхождений элемента в дерево равно одному; длина пути до ближайшего узла с заданным элементом равна 0 (заданный элемент хранится в корне дерева)
5	(a(b(()(c)))(d)) Element to search for: h	The bintree isn't correct.	Некорректная запись бинарного дерева.
6	(1(2(4)(5))(3(6)(7))) Element to search for: 0	0 isn't found path length = -1	Корректная запись бинарного дерева, которое не содержит заданный элемент; длина пути до ближайшего узла

			с заданным элементом в этом случае принимается за (-1)
7	./lab3	BinTree = <ввод из консоли> Element to search for: <ввод из консоли> <вывод результата в консоль>	Программа запущена без аргументов (файлов для считывания и записи данных), пользователь сам вводит скобочную запись бинарного дерева и элемент для поиска.
8	./lab3 Tests/test8.txt result8	Tests/test8.txt: No such file or directory	Программе передан несуществующий файл

ПРИЛОЖЕНИЕ С
ВИЗУАЛИЗАЦИЯ

Краткая запись иерархического списка	Визулизация
(a(b(d)(e()(g)))(c()(f)))