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

Студент гр. 9304 _____ Попов Д.С

Преподаватель _____ Филатов А.Ю

Санкт-Петербург

2020

Цель работы.

Изучить понятие бинарного дерева. Реализовывать программу с использованием бинарного дерева в языке C++.

Задание.

1м. 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).

Выполнение работы.

На вход программе подается 2 строки: представление структуры бинарного дерева в скобочном виде и элемент для поиска. Строка со структурой передается в функцию *checkBT* которая проверяет расположение скобок, их кол-во и расположение узлов. Если структура не удовлетворяет требованиям — программа завершает свою работу. Строка со структурой подается в конструктор класса *Tree,* где на ее основе методом *createBT* создается массив указателей на объекты класса *Node*. Класс *Node* содержит шаблонное поле с хранимыми данными *data* и метод для получения этих данных *getData*. Для поиска необходимого элемента в массиве в классе *Tree* реализован метод *result*, который определяет, содержит ли бинарное дерево нужный элемент, а если содержит, то сколько раз он встречается и длину пути до ближайшего. Так же определены конструктор и оператор перемещения.

Разработанный программный код см. в приложении А.

Формат входных и выходных данных.

На вход программе подается две строки со структурой бинарного дерева и элемента для поиска.

Программа должна определить, присутствует ли введенный элемент в бинарном дереве, а так же кол-во вхождений и кратчайший путь до ближайшего элемента.

Скобочное представление (a(b(f(m)(m))(g(m)(n)))(c(h)(j(k)(m)))) будет иметь следующий вид:

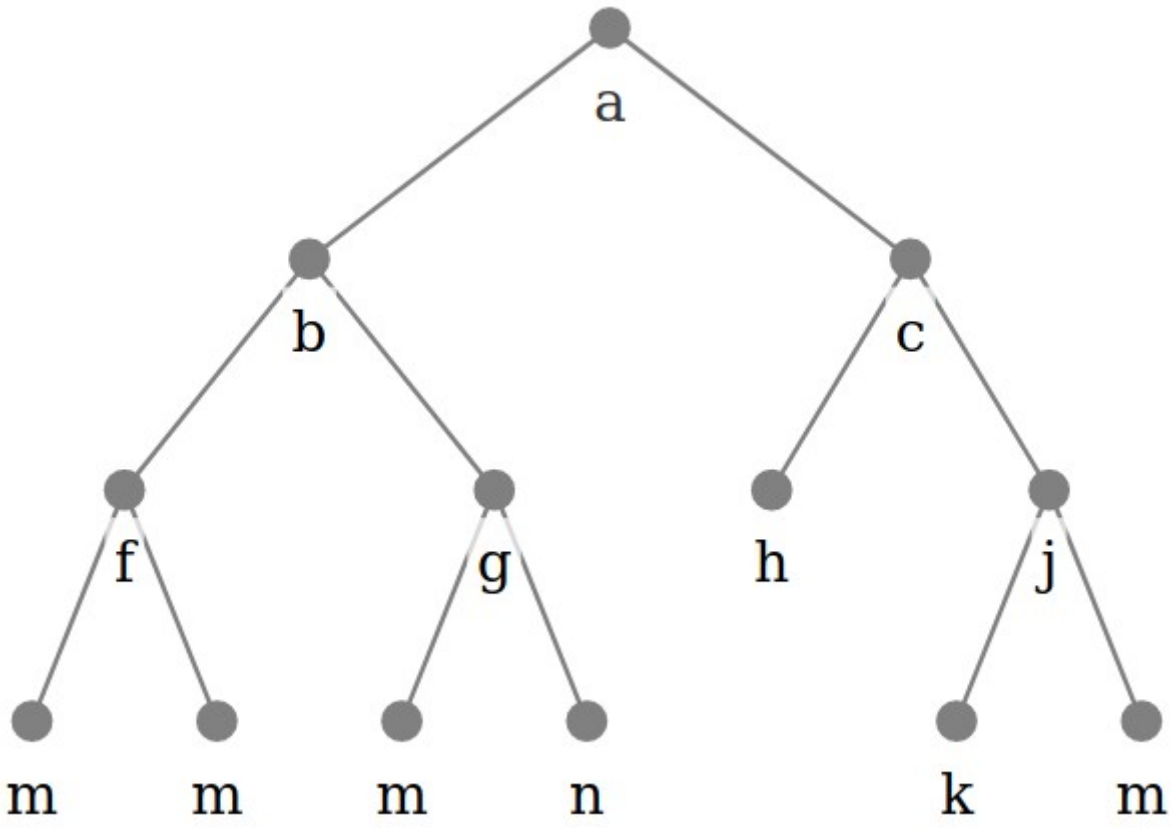

Рис. 1 — Графическое представление одного из тестов.

Тестирование.

Для проведения тестирования был написан bash-скрипт ./script. Скрипт запускает программу и в качестве входных аргументов подает готовые строки заранее подготовленные в файлах, расположенные в папке ./Tests

Результаты тестировния см. в приложении Б.

Выводы.

Было изучено понятие бинарного дерева. Реализовано бинарное дерево с помощью языка программирования C++, с использованием шаблонов.

Была реализована программа, создающая бинарное дерево на основе массива. Метод хранения в одном массиве оправдан тока в случае использования полных деревьев, ибо пропущенный на ранних уровнях узел даст огромное кол-во пустых ячеек.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <vector>
#include <memory>

#define SIZE 128



template <typename Elem>
class Node{
    Elem data;
public:
    Node(Elem newData):data(newData){}
    Elem getData(){
        return data;
    }
};


template <typename Elem>
class Tree{
    std::vector<Node<Elem>*> arrNode;
    void createBinTree(std::string BT){

        size_t first = 0;
        size_t iter = 0;
        size_t count = 0;
        size_t arrLevel = 1;
        std::vector<Node<Elem>*> nodePtr = arrNode;

        auto rec = [&iter, &nodePtr, &BT, &arrLevel](size_t count, size_t
first, auto &&rec){

            iter++;
```

```
    while(count > nodePtr.size()){
        arrLevel++;
        std::vector<Node<Elem>*> newArr;
        newArr.resize(SIZE * arrLevel);
        for(size_t i = 0; i < nodePtr.size(); i++){
            if(nodePtr[i] != nullptr){
                newArr[i] = nodePtr[i];
            }
        }
        nodePtr = newArr;
    }

    if(BT[first] == '(' && BT[iter] == ')'){
        return;
    }

    if(BT[first] == '(' && BT[iter] == '('){
        if(iter - first == 2){
            rec((count - 1) * 2 + 1, iter, rec);
        }else{
            rec((count - 1) * 2 + 2, iter, rec);
        }
    }

    if(BT[first] == '(' && isalpha(BT[iter])){

        if(!first){
            nodePtr[0] = new Node<Elem>(BT[iter]);
        }else{
            nodePtr[count] = new Node<Elem>(BT[iter]);
        }

        count++;
    }

    rec(count, first, rec);
};
```

```
        rec(count, first, rec);
        arrNode = nodePtr;

    }

public:
    Tree(std::string BT){
        arrNode.resize(SIZE);
        createBinTree(BT);
    }
    ~Tree(){
        for(size_t i = 0; i < arrNode.size(); i++){
            if(arrNode[i]){
                delete arrNode[i];
            }
        }
    }
    Tree(Tree&& tree){
        std::swap(tree.arrNode, arrNode);
    }
    Tree& operator=(Tree&& tree){
        arrNode = std::move(tree.arrNode);
        return *this;
    }
    void result(std::string findElem){
        if(findElem.size() != 1){
            std::cout << "Некорректный элемент для поиска!" << std::endl;
        }else{
            size_t position = 0;
            size_t count = 0;
            bool flag = 1;
            for(size_t i = 0; i < arrNode.size(); i++){
                if(arrNode[i]){
                    if(arrNode[i]->getData() == *findElem.c_str()){
                        if(flag){
                                std::cout << "Элемент " << findElem << "
присутствует в дереве!\n";
                            position = i;
                            flag = 0;
```

```
                        }
                        count++;
                    }
                }
            }

            if(!flag){
                std::cout << "Число вхождении: " << count << '\n';
                count = 0;
                while(1){
                    if(position == 0){
                        break;
                    }
                    position = (position - 1) / 2;
                    count++;
                }
                  std::cout << "Длинна пути до ближайшего элемента: " <<
count << std::endl;
            }else{
                std::cout << "-1" << std::endl;
            }
        }
    }
};



bool checkBT(std::string inputString){
    int level = 0;
    int node = 0;

    if(inputString.size() == 0){
        return 0;
    }

    for(size_t i = 0; i < inputString.size(); i++){

        if(isalpha(inputString[i])){
            node++;
```

```
        }else if(inputString[i] == '('){
            level++;
        }else if(inputString[i] == ')'){
            level--;
            node--;
        }

        if((level - node) < 0){
            return 0;
        }

        if((node - level < -1)){
            return 0;
        }

        if(level < 0){
            return 0;
        }
    }

    if(level != 0){
        return 0;
    }

    if(inputString[0] != '('){
        return 0;
    }

    if(inputString[inputString.size() - 1] != ')'){
        return 0;
    }

    return 1;
}



int main(int argc, char* argv[]){
```

```
    setlocale(LC_ALL, "ru");

    std::string inputString {};
    getline(std::cin, inputString);

    if(!checkBT(inputString)){
        std::cout << "Неверная структура дерева!" << std::endl;
        return -1;
    }

    std::unique_ptr<Tree<char>> BT(new Tree<char>(inputString));

    std::string findElem {};
    getline(std::cin, findElem);

    BT->result(findElem);

    return 0;
}
```

Название файла: script

```
#!/bin/bash
for n in {1..6}
do
      arg=$(cat Tests/test$n.txt)
      echo -e "\nTest $n:"
      echo "BinTree = $arg"
      ./lab3 < Tests/test$n.txt
done
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ ПРОГРАММЫ

Результаты тестирования представлены в табл.1

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(a(b)(c)) e	-1	Элемента нет в дереве
2.	(a(b)(c(d)(e(f)(g(k)(l))))) l	Элемент l присутствует в дереве! Число вхождении: 1 Длинна пути до ближайшего элемента: 4	Корректная работа программы
3.	(a(b(f(m)(m))(g(m)(n)))(c(h)(j(k)(m)))) m	Элемент m присутствует в дереве! Число вхождении: 4 Длинна пути до ближайшего элемента: 3	Корректная работа программы
4.	c(a)b) b	Неверная структура дерева!	Отсутствует скобка
5.	(aa(b)(c)) a	Неверная структура дерева!	Лишний узел.
6.	(2(1)(3)) 2	Неверная структура дерева!	Неверный тип данных.