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

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студент гр. 8381 _____ Михайлов Д.А.

Преподаватель _____ Ерёменко А.А.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с основными характеристиками и особенностями такой структуры данных, как бинарное дерево, изучить особенности ее реализации на языке программирования C++. Разработать программу, использующую бинарное дерево для обработки формулы.

Задание.

Для заданного бинарного дерева b типа BT с произвольным типом элементов:

- определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

- вычислить длину внутреннего пути дерева b, т. е. сумму по всем узлам длин путей от корня до узла.

Основные теоретические положения.

Дерево – конечное множество T, состоящее из одного или более узлов, таких, что:

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

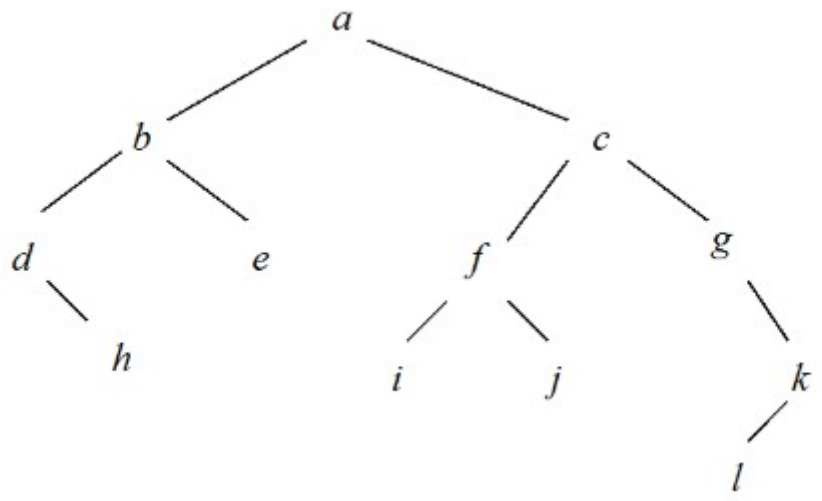
б) остальные узлы (исключая корень) содержатся в m ³ 0 попарно не пересекающихся множествах T1, T2, ..., Tm, каждое из которых, в свою очередь, является деревом. Деревья T1, T2, ..., Tm называются поддеревьями данного дерева.

Лист (концевой узел) — узел множество поддеревьев которого пусто.

Упорядоченное дерево — дерево в котором важен порядок перечисления его поддеревьев.

Лес – множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев.

При программировании и разработке вычислительных алгоритмов удобно использовать именно такое рекурсивное определение, поскольку рекурсивность является естественной характеристикой этой структуры данных.



Бинарное дерево

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом

Выполнение работы.

Написание работы производилось на базе операционной системы Ubuntu 18 в среде разработки Clion. Сборка, отладка производились там же. Исходные коды файлов программы представлены в приложениях А-Ж.

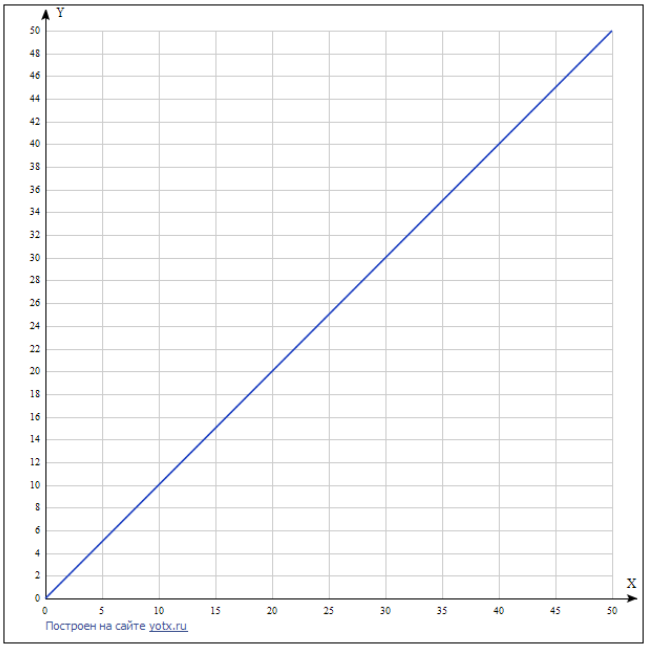
Проект разделен на две части, api отвечает за ввод данных в программу, bintree описывает структуру бинарного дерева:

Таблица 1 – Основные методы работы с бинарным деревом

Метод	Назначение
`BinTree()`	Создает пустое бинарное дерево
`createTree(std::vector<std::string> tokens)`	Создает бинарное дерево из векторов строк-элементов, полученного из входной строки
`max_depth(Node *hd)`	Возвращает максимальную глубину дерева
`getFullWeight(Node *hd, int now)`	Возвращает количество узлов на заданном уровне

Оценка сложности алгоритма.

Алгоритмы нахождения максимальной глубины и количества улов на заданном уровне являются рекурсивными, каждый узел дерева обрабатывается один раз, следовательно, сложность алгоритма $O(N)$



Тестирование программы.

Был проведен ряд тестов, проверяющих корректность работы программы. Результаты тестирование приведены в табл. файле Test.txt

Выводы.

В ходе выполнения лабораторной работы была написана программа, создающая бинарное дерево, подсчитывающая его максимальную глубину, а также находящая количество узлов на заданном уровне.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.C

```
#include <iostream>
#include "api.h"

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

    std::cout << "Hello, World!\n";
    inputF();    inputF();    inputF();    inputF();    inputF();
    std::cout << "...";
    return 0;
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. API.CPP

```
#include "api.h"
#include<string>

std::string delSpace (std::string rowInput){
    std::string out="";
    for (auto i = 0;i < rowInput.length();i++){
        if (rowInput[i]!=' ' && rowInput[i]!='\n' && rowInput[i]!='\t' )
            out.push_back(rowInput[i]);
    }
    return out;
}

std::vector<std::string> mySplit(std::string rowInput){
    std::string st = delSpace(rowInput);
    auto i = 0;
    std::vector<std::string> out;
    std::string tmp="";
    for(;i<st.length();i++){
        if (st[i] == ')' || st[i] == '(') out.push_back(std::string(1,st[i]));
        else {
            for (; st[i]!=')' && st[i]!='(' ;i++){
                tmp.push_back(st[i]);
            }
            out.push_back(tmp);
            out.push_back(std::string(1,st[i]));
            std::cout<<"tmp: "<<tmp<<'\n';
            tmp.clear();
        }
    }
    for (int i = 0; i < out.size(); i++) {
        std::cout << out.at(i) << ' ';
    }
    std::cout<<std::endl;
    return out;
}


void inputF(){
    std::string data ;
    getline(std::cin, data);
    std::string input = delSpace(data);
    if (input == "") {
        std::cout<<"Error!\n Введите дерево\n";
        return;
    }
    else{
        if (input[0] != '(' || input[input.size() - 1] != ')'){
            std::cout<<"Error!\n Дерево введенно некорректно\n";
            return;
        }
        else{
            BinTree* BT = new  BinTree();
            BT->Head = BT->createTree(mySplit(input));

std::cout<<"------------------------------------------------------------\n";
            std::cout<<"Глубина дерева: "<<BT->max_depth(BT->Head) -
1<<std::endl;
```

```
std::cout<<"-------------------------------------------------------\n";
            std::cout<<"Внутренний вес: "<<BT->getFullWeight(BT-
>Head)<<std::endl;

std::cout<<"-------------------------------------------------------\n";
//            example (6 (5 (4) (9))(5))
        }
    }
}
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. API.H

```
#ifndef api_hpp
#define api_hpp

#include <stdio.h>
#include<vector>
#include<string>
#include "bintree.h"

std::string delSpace (std::string rowInput);

std::vector<std::string> mySplit(std::string rowInput);

void inputF();
#endif /* api_hpp */
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINTREE.CPP

```
#include "bintree.h"
#include "api.h"

BinTree::BinTree(){
    Head = new Node;
    Head->data = "";
    Current = Head;
}

// Node* BinTree::createTree(QStringList tokens){
Node* BinTree::createTree(std::vector<std::string> tokens){
    std::cout<<"-_-_-_-_-Next step-_-_-_-_-\n";
    Node* finalNode = new Node;
    if(tokens.size()==2) return finalNode;
    int i =1;
    std::string ltree = "";
    std::string rtree = "";

    finalNode->data = tokens[i++];
    int index_i = i;    /* Индекс открывающей скобки левого поддерева */
    if(tokens[i] == "("){
        auto openBrackets = 1;
        auto closeBrackets = 0;

        while (openBrackets != closeBrackets) {
            i++;
            if (tokens[i] == "("){
                openBrackets++;
            }
            else if (tokens[i] == ")"){
                closeBrackets++;
            }

        }

        for (;index_i<=i; index_i++){
            ltree.append(tokens[index_i]);
        }
        std::cout<<"Открытые скобки: "<<openBrackets<<'\n'
                 <<"Закрытые скобки: "<<closeBrackets<<'\n';
        std::cout<<"Вниз (лево): "<<tokens[index_i]<<'\n';
        finalNode->left = createTree(mySplit(ltree));

        i++;

        if (tokens[i] == ")"){      /* Если правого поддерева нет (достигнут
конец строки после структуры левого поддерева*/
            return finalNode;
        }

        int index_j = i;    /* Индекс открывающей скобки левого поддерева */
        if(tokens[i] == "("){
            auto openBrackets = 1;
            auto closeBrackets = 0;

            while (openBrackets != closeBrackets) {
                i++;
                if (tokens[i] == "("){
```

```
                openBrackets++;
            }
            else if (tokens[i] == ")"){
                closeBrackets++;
            }


        }

        for (;index_j<=i; index_j++){
            rtree.append(tokens[index_j]);
        }
        std::cout<<"Открытые скобки: "<<openBrackets<<'\n'
                 <<"Закрытые скобки: "<<closeBrackets<<'\n';

        std::cout<<"Вниз (право): "<<tokens[index_j]<<'\n';
        finalNode->right = createTree(mySplit(rtree));
    }


}
std::cout<<"Вверх \n";
return finalNode;
}
int BinTree::max_depth(Node *hd){
    std::cout<<"-_-_-_-_-Уровень ниже-_-_-_-_-\n";
    if((hd == NULL) || (hd->data == "^")) {
        std::cout<<"Вверх, глубина: 0\n";
        return 0;
    }
    else{
        std::cout<<"Вниз (лево)\n";
        int lDepth = max_depth(hd->left);
        std::cout<<"Вниз (право)\n";
        int rDepth = max_depth(hd->right);
        std::cout<<"Глубина левого:"<<lDepth<<"\n"
                 <<"Глубина правого:"<<rDepth<<"\n";
        if (lDepth > rDepth) {
            std::cout<<"Вверх, глубина: "<<lDepth+1<<"\n";
            return(lDepth + 1);
        }
        else {
            std::cout<<"Вверх, глубина: "<<rDepth+1<<"\n";
            return(rDepth + 1);
        }
    }

}
int BinTree::getFullWeight(Node *hd, int now){
    if (hd==nullptr){
        std::cout<<"Вверх: 0\n";
        return 0;
    }
    else {
        std::cout<<"Вниз (лево)\n";
        int l =getFullWeight(hd->left,now + 1);
        std::cout<<"Вниз (право)\n";
        int r =getFullWeight(hd->right,now + 1);
        now +=l+r ;
```

```
        std::cout<<"Вверх: "<<now<<"\tБыло: "<<now-l-r<<"\tЛево: "<<l<<"\tПраво:
"<<r<<"\n";
        return now;

    }
}
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINTREE.H

```
#ifndef bintree_hpp
#define bintree_hpp

#include <stdio.h>
#include<iostream>
#include <cmath>
#include<vector>
#include<string>



struct Node
{

    std::string data = "";
    Node* left = nullptr;
    Node* right = nullptr;

};

class BinTree
{
private:
    Node* Current = nullptr;
    std::string data;
public:
    Node* Head = nullptr;
    BinTree();
    Node* createTree(std::vector<std::string> tokens);
    int max_depth(Node *hd);
    int getFullWeight(Node *hd, int now = 0);
};


#endif /* bintree_hpp */
```