**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. 9382 _____ Михайлов Д.А.

Преподаватель _____ Фирсов М.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.

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

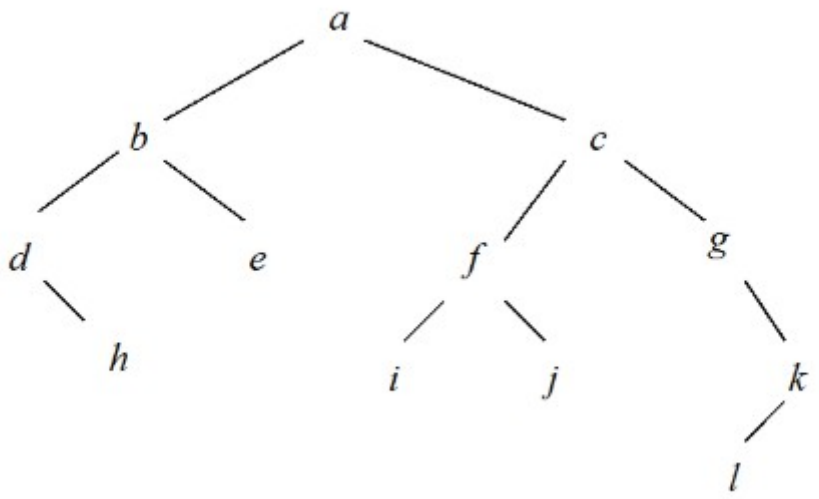



Рисунок 1 - Бинарное дерево

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.**

Таблица 1 – Основные функции работы с бинарным деревом

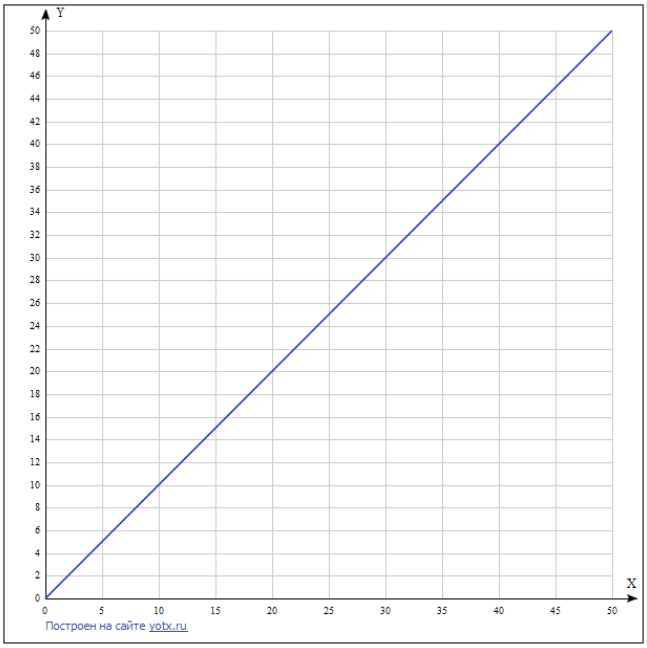
| Функция | Назначение |
|---|---|
| `void BinTree()` | Создает пустое бинарное дерево |
| `createTree(std::vector<std::string> tokens)` | Создает бинарное дерево из массива строк-элементов, полученного из входной строки |
| `max_depth(Node *hd)` | Возвращает максимальную глубину дерева |
| `getFullWeight(Node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

**Описание алгоритма**

Чтобы хранить бинарное дерево создается класс BinTree, дерево заполняется с помощью рекурсивной функции createTree, использующей векторный способ заполнения данных. Функция max_depth обходит дерево спускаясь максимально вниз-влево, пока есть такая возможность, в случае ее отсутствия поднимается на один уровень вверх и пробует спуститься вниз-направо, после чего опять вниз-влево, функция рекурсивна, если достигнуть нижнего уровня не получается, происходит проверка верхних ветвей.

**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. 2.

Таблица 2 – Тестирование программы

| Входная строка | Вывод |
|---|---|
| ( 6 ( 4 (5)  (9) ) (5 )) | Длина внутреннего пути дерева: 6<br>Максимальная глубина дерева: 2 |
| ( 4 (5 ) (5 | Дерево введено некорректно |
| ( 6 ( 4 (5)  (9) ) (5 )) | Длина внутреннего пути дерева: 6<br>Максимальная глубина дерева: 2 |
| ( 6 ( 4 (5)  (9) ) (5 (7) (9) )) | Длина внутреннего пути дерева: 10<br>Максимальная глубина дерева: 2 |

**Выводы.**

В ходе выполнения лабораторной работы была написана программа, создающая бинарное дерево, подсчитывающая его максимальную глубину

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.C

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

using namespace std;

/**
* Удаляет пробелы
* @param rowInput - входная строка
* @return
*/
string delSpace (string rowInput) {
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;
}

/**
* Парсит строку и каждый символ записывает как отдельный элемент массива
* @param rowInput - входная строка
* @return
*/
vector<string> mySplit(string rowInput) {
string st = delSpace(rowInput);
auto i = 0;
vector<string> out;
string tmp;
for(;i<st.length(); i++){
if (st[i] == ')' || st[i] == '(') out.push_back(string(1, st[i]));
else {
for (; st[i]!=')' && st[i]!='('; i++) {
tmp.push_back(st[i]);
}
out.push_back(tmp);
out.push_back(string(1,st[i]));
cout <<" tmp: " << tmp << endl;
tmp.clear();
}
}
for (int i = 0; i < out.size(); i++) {
cout << out.at(i) << ' ';
```

```
}
cout << endl;
return out;
}

/** Узел дерева */
struct Node {
string data = "";
Node* left = nullptr;
Node* right = nullptr;
};

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

/** Создает пустое бинарное дерево */
BinTree::BinTree() {
Head = new Node;
Head->data = "";
Current = Head;
}

/**
* Создает бинарное дерево из массива строк-элементов, полученного из входной строки
* @param tokens - массив элементов бинарнорго дерева
* @return
*/
Node* BinTree::createTree(vector<string> tokens) {
cout << "-_-_-_-_-Next step-_-_-_-_-" << endl;
Node* finalNode = new Node;
if (tokens.size() == 2) return finalNode;
int i = 1;
string ltree = "";
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]);
}
cout << "Открытые скобки: " << openBrackets << endl;
cout << "Закрытые скобки: "<<closeBrackets << endl;
cout << "Вниз (лево): "<<tokens[index_i] << endl;
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]);
}
cout << "Открытые скобки: " << openBrackets << endl;
cout << "Закрытые скобки: "<<closeBrackets << endl;
cout << "Вниз (право): "<<tokens[index_j] << endl;
finalNode->right = createTree(mySplit(rtree));
}
}
cout << "Вверх" << endl;
return finalNode;
```

```
}

/**
* Возвращает максимальную глубину дерева
* @param hd - узел
* @return
*/
int BinTree::max_depth(Node *hd) {
cout << "-_-_-_-_-Уровень ниже-_-_-_-_-" << endl;
if ((hd == nullptr) || (hd->data == "^")) {
cout << "Вверх, глубина: 0" << endl;
return 0;
}
else{
cout << "Вниз (лево)" << endl;
int lDepth = max_depth(hd->left);
cout << "Вниз (право)" << endl;
int rDepth = max_depth(hd->right);
cout << "Глубина левого:" << lDepth << endl;
cout << "Глубина правого:" << rDepth << endl;
if (lDepth > rDepth) {
cout << "Вверх, глубина: " << lDepth+1 << endl;
return(lDepth + 1);
}
else {
cout << "Вверх, глубина: " << rDepth+1 << endl;
return(rDepth + 1);
}
}
}

/**
* Возвращает количество узлов на заданном уровне
* @param hd - узел
* @param now - уровень
* @return
*/
int BinTree::getFullWeight(Node *hd, int now) {
if (hd==nullptr) {
cout << "Вверх: 0" << endl;
return 0;
}
else {
cout<<"Вниз (лево)" << endl;
int l = getFullWeight(hd->left,now + 1);
cout<<"Вниз (право)" << endl;
int r = getFullWeight(hd->right,now + 1);
now += l+r ;
```

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

/**
* Запуск основного алгоритма
* @param input - строка с деревом
*/
void process(string input) {
input = delSpace(input);
if (input == "") {
cout << "\x1b[31mОшибка! Дерево не введено\x1b[0m" << endl << endl ;
return;
}
else {
if (input[0] != '(' || input[input.size() - 1] != ')') {
cout << "\x1b[31mОшибка! Дерево введенно некорректно\x1b[0m" << endl <<
endl ;
return;
}
else {
int weight, depth;
BinTree* BT = new BinTree();
BT->Head = BT->createTree(mySplit(input));
cout   <<"-------------------------------------------------------"   <<
endl;
weight = BT->getFullWeight(BT->Head);
cout   <<"-------------------------------------------------------"   <<
endl;
depth = BT->max_depth(BT->Head) - 1;
cout << "\x1b[33m=========================================" << endl;
cout << "Длина внутреннего пути дерева: " << weight << endl;
cout << "Максимальная глубина дерева: " << depth << endl;
cout  <<  "=========================================\x1b[0m"  <<  endl  <<
endl;
}
}
}

/** Обработка консольного ввода */
void inputTerminal() {
string input;
cout << "Введите дерево: ";
cin.ignore(numeric_limits  <  streamsize  >  ::max(),  '\n');  //  сброс
содержимого буфера перед вводом новой строки
```

```
getline(cin, input); // считываем строку, которая может содержать пробелы
process(input);
}

/** Обработка ввода из файла */
void inputFile() {
string input;
ifstream file("input.txt"); // открываем файл для чтения
while (getline(file, input)) { // обрабатываем его построчно
cout << "Из файла считано дерево: " << input << endl;
process(input);
}
file.close();
}

int main() {
char cmd;
cout << "\x1b[32mОпределение максимальной глубины бинарного дерева\x1b[0m" << endl;
cout << "Пожалуйста, выберите способ ввода данных: 1 - из файла, 0 - из консоли, любое другое значение - выход из программы" << endl;
cout << "Ваш выбор: ";
cin >> cmd;
switch (cmd) {
case '0':
inputTerminal();
break;
case '1':
inputFile();
break;
default:
break;
}
return 0;
}
```