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 Аксёнова Е.А.

Преподаватель Филатов Ар.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с понятием бинарного дерева. Реализовать бинарное дерево для решения задания на языке программирования C++.

Задание.

Вариант 2.

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.

Описание алгоритм работы.

При запуске программы из стандартного потока ввода считывается строка. Она объявляется потоком ввода, и переписываем её в новую строку, избегая пробелов. Далее проверяется валидность поданной строки. Для этого реализована функция bool checkString(), она проверяет количество скобок, а также наличие хотя бы корня в дереве. Затем создается объект класса Tree tree, для которого вызываются методы для поиска длины максимального пути и суммы длин всех путей. Они выводятся в стандартный поток вывода.

Класс Node:

Класс содержит приватные поля: “умные” указатели (std::shared_ptr) на левое и правое поддеревья, а также символ data, хранящий данные узла. Конструктор реализован по умолчанию.

Класс Tree:

В классе Tree содержится публичное поле root (указатель на корень всего дерева). В конструкторе класса Tree вызывается рекурсивная функция createTree(). В функции createNode() из поданной строки выделяется данные, хранящиеся в узле, а также указатели на левое и правое поддерево, после чего

возвращается новый экземпляр класса Node. Метод maxRoute() проверяет, если дерево состоит только из корня, то возвращает 0, если же нет, то проверяет, есть ли указатель на левое и правое поддерево, вызывается для каждого из них, а потом прибавляет большее из полученных чисел. В методе sumRoute() объявляется целочисленная переменная sum равная 0. Далее проверяется есть ли указатель на левое и правое поддерево, если есть, то добавляем к переменной sum 1 и значение данного метода от левого или правого поддерева соответственно. Метод printTree() выводит открывающуюся скобку, и значение лежащее в узле, затем проверяет, если существуют указатели на левое или правое поддеревья, то вызывает для них данную функцию, если же указатель равен nullptr, то выводятся просто пустые скобки, завершается всё выводом закрывающейся скобки.

Разработанный программный код см. в приложении А.

Формат входных и выходных данных.

На вход программе подается скобочная запись бинарного дерева (строка, состоящая из символов и круглых скобок).

Программа выводит два числа: первое - длина максимального путиб второе - сумма длин всех путей.

Описание основных структур данных и функций.

- Класс Node - создание узлов и атомов иерархического списка.
- Класс Tree – создание иерархического списка
    a. Метод printTree() - вывод бинарного дерева
    b. Метод createTree() - создание бинарного дерева
    c. Метод maxRoute() - поиск длины максимального пути в бинарном дереве
    d. Метод sumRoute() - поиск суммы длин путей в бинарном дереве

Тестирование.

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции и линковки программы, после чего будет создан исполняемый файл lab3. Тестирование производится посредством bash-скрипта. Далее прописываем команду chmod +x test_script. А затем прописываем “./test_script”, что приводит к запуску тестирования. Результаты тестирования представлены в приложении Б.

Ниже представлено графическое представление бинарного дерева (a()(b(c)(d))) (Рисунок 1):

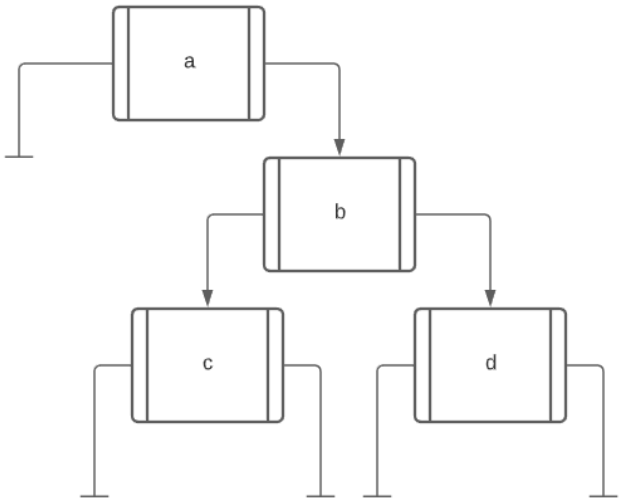

Рисунок 1 - Графическое представление бинарного дерева

Выводы.

Было изучено понятие бинарного дерева. Было реализовано бинарное дерево с помощью языка программирования C++, с использованием умных указателей std::shared_ptr.

Была реализована программа, создающая бинарное дерево, ищущая длину максимального пути и сумму длин путей дерева.

Использование оправдано, так как приходится тратить меньше ресурсов на обработку бинарного дерева, чем на обработку списков (в худшем случае столько же).

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include "tree.h"

bool checkString(std::string str) {
    int size = str.size();
    if (size < 3) {
        std::cerr << "The tree has no root" << '\n';
        return 0;
    }
    if (str[0] != '(') {
        std::cerr << "The first symbol is not correct" <<
'\n';
        return 0;
    }
    int counter = 1;
    for (int i = 1; i < size; i++) {
        if (str[i] == '(') {
            counter += 1;
        }
        if (str[i] == ')') {
            counter -= 1;
        }
    }
    if (counter) {
        std::cerr << "Wronq numder of brackets\n";
```

```
            return 0;
        }
    }

    int main() {
        std::string str1, str2;
        char value;
        std::getline(std::cin, str1);
        std::stringstream ss(str1);

        while (ss >> value)
        {
            str2.push_back(value);

            if (ss.peek() == ' ') {
                ss.ignore();
            }
        }
        if (!checkString(str2)) {
            std::cout<<"You entered wrong string"<<'\n';
            return 0;
        }
        Tree tree(str2);
        std::cout << tree.maxRoute(tree.root) << '\n';
        std::cout << tree.sumRoute(tree.root) << '\n';
        tree.printTree(tree.root);
    }
```

Название файла: tree.cpp

```
#include "tree.h"
Tree::Tree(std::string& str) {
    root = createTree(str);
}
```

```
Tree::~Tree() {
}
std::shared_ptr<Node> Tree::createTree(std::string& str) {
    std::string left, right;
    if (str.size() <3) {
        return nullptr;
    }
    if(str.size()==3){
        std::shared_ptr<Node>             node             =
std::make_shared<Node>();
        node->left = nullptr;
        node->right = nullptr;
        node->data = str[1];
        return node;
    }
    if (str.size() > 3) {
        int count = 0;
        int i = 2;
        do {
            if (str[i] == '(') {
                count += 1;
            }
            if (str[i] == ')') {
                count -= 1;
            }
            i++;
        } while (count);
        left = str.substr(2, i-2);
        right = str.substr(i,str.size()-1-i);
        std::cout << left << ' ' << right<<'\n';
        std::shared_ptr<Node>             node             =
std::make_shared<Node>();
        node->left = createTree(left);
```

```
        node->right = createTree(right);
        node->data = str[1];
        return node;
    }

}

int Tree::maxRoute(std::shared_ptr<Node> cur) {
    int max = this->root == cur ? 0 : 1;
    int leftCount = 0, rightCount = 0;
    if (cur->left || cur->right) {
        if (cur->left)
            leftCount = maxRoute(cur->left);
        if (cur->right)
            rightCount = maxRoute(cur->right);
        max += leftCount > rightCount ? leftCount : rightCount;
    }
    return max;

}

int Tree::sumRoute(std::shared_ptr<Node> cur) {
    int sum = 0;
    if (cur->left) {
        sum += 1 + sumRoute(cur->left);
    }
    if (cur->right) {
        sum += 1 + sumRoute(cur->right);
    }
    return sum;
}
```

```
void Tree::printTree(std::shared_ptr<Node> cur) {
    std::cout << '(' << cur->data;
    if (cur->left || cur->right) {
        if (cur->left) {
            printTree(cur->left);
        }
        else {
            std::cout << "()";
        }
        if (cur->right) {
            printTree(cur->right);
        }
        else {
            std::cout << "()";
        }
    }
    std::cout << ')';
}
```

Название файла: tree.h

```
#ifndef TREE_H
#define TREE_H
#include <iostream>
#include <string>
#include "node.h"
class Tree {
public:
    Tree(std::string&);
    ~Tree();
    void printTree(std::shared_ptr<Node>);
    std::shared_ptr<Node> root;
    int maxRoute(std::shared_ptr<Node>);
    int sumRoute(std::shared_ptr<Node>);
private:
```

```
        std::shared_ptr<Node> createTree(std::string&);
    };
    #endif
```

Название файла: node.cpp

```
    #include "node.h"
    Node::Node(){}
```

Название файла: node.h

```
    #ifndef NODE_H
    #define NODE_H
    #include <iostream>
    class Node {
    public:
        Node();
    private:
        std::shared_ptr<Node> left, right;
        char data;
        friend class Tree;
    };
    #endif
```

Название файла: Makefile

```
all: main.o node.o tree.o
        g++ main.o node.o tree.o -o lab3
main.o: Source/main.cpp Source/tree.h
        g++ -c ./Source/main.cpp
tree.o: Source/tree.cpp Source/tree.h Source/node.h
        g++ -c Source/tree.cpp
node.o: Source/node.cpp Source/node.h
        g++ -c Source/node.cpp
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Результат проверки
1.	(a)	0 0	correct
2.	(a(b)(c))	1 2	correct
3.	(a()(b(c)(d)))	2 3	correct
4.	(a(b(d()(h))(e))(c(f(i)(j))(g()(k(l)()))))	4 11	correct
5.	()	You entered wrong string	correct
6.	(a(b)(c)	You entered wrong string	correct