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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья

Студент гр. 7383 _____ Медведев И. С.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2018

Содержание

Цель работы

Познакомиться с бинарными деревьями и научиться реализовывать их на языке программирования C++.

Для заданного бинарного дерева *b* типа *BT*:

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

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

в) напечатать элементы из всех листьев дерева *b*;

г) подсчитать число узлов на заданном уровне *n* дерева *b* (корень считать узлом 1-го уровня);

Вариант 2(а-г)-д.

Реализация задачи

`int main( )` – головная функция, в которой пользователю предоставляется выбор: считать бинарное дерево с файла или с консоли. Затем запускаются методы класса BinaryTree для вычисления высоты дерева, подсчета узлов на заданном пользователем уровне, вывода листьев и вычисления внутреннего пути дерева, а затем запускается деструктор.

Методы класса BinaryTree:

Конструктор, который задает начальные параметры для переменных `BinaryTree* left` (указатель на левое поддерево), `BinaryTree* right` (указатель на правое поддерево), `BinaryTree* parent` (родитель элемента).

Деструктор, удаляющий указатели на родителя, левое и правое поддерево.

`void readBT(const string &string, int &i)` – главная функция считывания бинарного дерева. В этой функции запускаются функции для считывания корня и считывания левого и правого поддерева. На вход подается строка с бинарным деревом и индекс текущего символа.

`void read_root(const string &string, int &i)` – функция, которая создает корень дерева/поддерева. На вход подается строка с бинарным деревом и индекс текущего символа.

`void readUnder(const string &string, int &i, int side)` – функция для считывания левого и правого поддерева. На вход подается строка с бинарным деревом, индекс текущего символа и флаг, показывающий какое поддерево мы считываем.

`int height()` – функция для нахождения длины бинарного дерева. Функция находит самое длинное ветвление.

`int nodes()` – функция считает количество узлов в дереве.

`void leafs()` – функция находит листья и печатает их. Если у узла указатель на `left` и `right` равен `NULL`, то узел является листом.

`int nodesLvl(int n)` – функция получает на вход уровень дерева. Функция выводит количество узлов на этом уровне.

`int internal_lenght()` – функция вычисляет длину внутреннего пути дерева.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Вывод

В ходе выполнения лабораторной работы были изучены основные понятия о бинарных деревьях, был реализовано бинарное дерево на базе динамической памяти на языке программирования C++. Также была написана программа для нахождения высоты дерева, длины внутреннего пути дерева, вывода листьев и подсчета числа узлов на заданном уровне.

ПРИЛОЖЕНИЕ А.

КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstring>
#include <cctype>
#define LEFT 0
#define RIGHT 1

using namespace std;


class BinaryTree {
    private:
            char element;

        BinaryTree *left;
        BinaryTree *parent;
        BinaryTree *right;

    public:
        void readBT(const string &string, int &i) {
                if (string[i] == '(') {
                    i++;
                    read_root(string, i);
                    readUnder(string, i, LEFT);
                    readUnder(string, i, RIGHT);
                    if (string[i] == ')')
                            i++;
                    else{
                    cout << "')' expected at the index:"<<i<<endl;
                   exit(1);


                }

                }
            else{
                cout << "'(' expected at the index:"<<i<<endl;
                 exit(1);

                }

        }

            void read_root(const string &string, int &i){
            if (isdigit(string[i]) || isalpha(string[i])) {
                 element = string[i];
                 i++;
                }
                else {
                     cout<<"Expected number or letter at the index: "<<i<<endl;
```

```
            exit(1);
        }
}
    void readUnder(const string &string, int &i, int side){
    if (string[i] == '#')
            i++;
        else if(string[i] != ')'){
            if (side == LEFT) {
                    left = new BinaryTree;
                    left->parent = this;
                    left->readBT(string, i);
            }
            else {
                    right = new BinaryTree;
                    right->parent = this;
                    right->readBT(string, i);
            }
        }

}

BinaryTree(){
        left = NULL;
        parent = NULL;
        right = NULL;
}

~BinaryTree(){
    if (left)
            delete left;
        if (right)
            delete right;
}

int height(){
    int height_left = 0;
        if (left != NULL)
            height_left = left->height();
        int height_right = 0;
        if (right != NULL)
            height_right = right->height();
        if (height_left > height_right)
            return 1 + height_left;
        return 1 + height_right;
}


int nodes(){
    if (left == NULL && right == NULL)
        return 1;
    int left_nodes = 0 , right_nodes = 0;
    if (left != NULL)
        left_nodes = left->nodes();
```

```
            if (right != NULL)
                right_nodes = right->nodes();
            return left_nodes + right_nodes + 1;
        }


        void leafs(){
            if(left == NULL && right == NULL)
                    cout << element << " ";
            if(left != NULL)
                    left->leafs();
            if(right != NULL)
                    right->leafs();
            }


        int nodesLvl(int n){
            int result = 0;
            if (n<=0){
            cout<<"Expected positive number and not zero"<<endl;
            }
            else if (n == 1)
                return 1;
            else{
                if(left)
                    result += left->nodesLvl(n-1);
                if(right)
                    result += right->nodesLvl(n-1);
            }
            return result;
        }


        int internal_lenght(){
             int lenght = 0;
            if(left != nullptr)
                lenght += left->internal_lenght() + left->nodes();
            if(right != nullptr)
                lenght += right->internal_lenght() + right->nodes();
            return lenght;

        }
};

#include <fstream>


    int main(){
        BinaryTree b;
        int i = 0, lvl;
        string string;
        char forSwitch;
        ifstream infile("BT.txt");
```

```
        while (true){
            i = 0;
            cout<<"Press 1 to read from console, press 2 to read from file, press 0
to exit."<<endl;
            cin >> forSwitch;
            switch (forSwitch) {
                case '2':{
                    getline(infile, string);
                    break;
                }
                case '1':{
                    cout<<"Enter the BT"<<endl;
                    cin>>string;
                    break;
                }
                case '0':{
                    return 0;
                }
                default:{
                    cout<<"Incorrect input"<<endl;
                    break;
                }
            }
            b.readBT(string, i);
            cout<<"Height of binary tree: "<<b.height()<<endl;
            cout<<"Enter level of tree"<<endl;
            cin>>lvl;
            cout<<"Number of nodes on the "<<lvl<<" level: "<<b.nodesLvl(lvl)<<endl;
            cout<<"Leafs: ";
            b.leafs();
            cout<<endl;
            cout<<"Internal length: "<<b.internal_lenght()<<endl;
            b.~BinaryTree();
        }


}
```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Таблица 1 — Результаты тестов.

Input	Output	True/False
(A(b##)(c##)) 2	Height of binary tree: 2 Number of nodes on the 2 level: 2 Leafs: b c Internal length: 2	True
(awe)	‘(‘ expected at the index:2	True
(a(b)(c)	‘)‘ expected at the index:8	True
(a(b)(c(d(e)(f))(r))) 3	Height of binary tree: 4 Number of nodes on the 3 level: 2 Leafs: b e f r Internal length: 12	True
(1(2)($))	Expected number or letter at the index:6	True

Окончание таблицы 1

(a(b(d#(h##))(e##))(c(f(i)(j))(g#(k(l)#)))) 4	Height of binary tree: 4 Number of nodes on the 4 level: 4 Leafs: h e i j l Internal length: 26	True