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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Вычисление высоты дерева

Студент гр. 0381 _____ Дзаппала Д.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Изучить такую СД, как дерево. Написать программу, которая вычисляет высоту заданного дерева.

Задание.

На вход программе подается корневое дерево с вершинами {0, . . . , n−1}, заданное как последовательность $parent_0, \ldots, parent_{n-1}$, где $parent_i$ — родитель i-й вершины. Требуется вычислить и вывести высоту этого дерева.

Формат входа.

Первая строка содержит натуральное число n. Вторая строка содержит n целых чисел $parent_0$, ... , $parent_{n-1}$. Для каждого $0 \le i \le n - 1$, $parent_i$ — родитель вершины i; если parent i = −1, то i является корнем. Гарантируется, что корень ровно один и что данная последовательность задаёт дерево.

Формат выхода.

Высота дерева.

Примечание: высотой дерева будем считать количество вершин в самом длинном пути от корня к листу.

Выполнение работы.

Был создан класс Tree, представляющий дерево. Класс содержит поля: ассоциативный контейнер map (tree), который работает по принципу ключ — значение. Ключом является целое число (лист дерева/родитель), значением контейнер vector, который хранит в себе «детей» листа; поле, хранящее корень дерева (tree_root); кол-во листов дерева (parent_count); высоту дерева, которую будет вычислять программа. Конструктор класса принимает vector с

деревом. В конструкторе проверяется, находится ли что-то в дереве, иначе высота дерева равна нулю. Далее, в «векторе» ищется корень дерева, который является индексом элемента со значением -1. В случае, если такого нет, то высота дерева равна -1 и программа выходит из конструктора. После этой проверки, идет проход по «вектору», в цикле создается временный «вектор», в который будут добавляться «дети» листа, и начинается еще один цикл. После добавления «детей», если они есть, в tree добавляется пара ключ — значение. В классе был создан приватный метод, для нахождения высоты дерева. В метод подается корень дерева, временно созданный «вектор», куда будут добавляться значения высоты дерева, в момент, когда у «родителя» больше не будет «детей». Функция рекурсивно вызывается для «дитя» «родителя». После рекурсивного обхода, берется максимальное значение высоты из временно созданного «вектора» и присваивается полю tree_height. Также была создана функция TEST, принимающая ссылку на объект дерева и правильное значение для теста.

Тестирование.

Здесь результаты тестирования, которые помещаются на одну страницу.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	5 4 -1 4 1 1	3	
2.	7 5 -1 1 1 1 6 4	5	

Выводы.

Была изучена СД — дерево, написан алгоритм нахождения высоты заданного дерева.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: 1Lab.cpp

```
#include<iostream>

#include<vector>
#include<map>
#include<algorithm>
#include<cassert>
using namespace std;

class Tree{
private:
map<int, vector<int>> tree;
int tree_root;
int parent_count;
int tree_height;

void find_height(int root, vector<int>& cur_len, int cur_height){

for(auto& child: tree[root])
find_height(child, cur_len, cur_height + 1);
cur_len.push_back(cur_height);
}

public:
Tree(const vector<int>& tree_parents){
if (tree_parents.size() == 0){
tree_height = 0;
return;
}

parent_count = tree_parents.size();
tree_root = -1;
for (int i = 0; i < tree_parents.size(); i++)
if (tree_parents[i] == -1){
tree_root = i;
break;
}
if (tree_root == -1){
tree_height = -1;
return;
}
vector<bool> check_parent(parent_count, false);
for (int i = 0; i < tree_parents.size(); i++){
if (tree_parents[i] == -1)
```

```
        continue;
        vector<int> childs;

        if (!check_parent[tree_parents[i]])
        check_parent[tree_parents[i]] = true;
        else
        continue;

        for (int j = i; j < tree_parents.size(); j++){
        if(tree_parents[j] == j)
        continue;
        else if (tree_parents[j] == tree_parents[i])
        childs.push_back(j);
        }
        tree.insert(pair<int, vector<int>>(tree_parents[i], childs));
        }
        // tree.erase(-1);
        vector<int> cur_len;
        find_height(tree_root, cur_len, 1);
        vector<int>::iterator      it      =      max_element(begin(cur_len),
end(cur_len));
        tree_height = *it;

        }
        void print_tree() const {
        for (auto& parents: tree){
        cout << "Key: " << parents.first << ", value: ";
        for (int i = 0; i < parents.second.size(); i++)
        cout << parents.second[i] << ", ";
        cout << endl;
        }
        }

        int height() const{
        return tree_height;
        }
        };

        void TEST(Tree& obj, int correct_answer){
        static int i = 0;
        assert(obj.height() == correct_answer);
        cout << ++i << ": ok" << endl;
        }

        int main(){

        // int n;
        // cin >> n;
```

```
// vector<int> par(n);
// for (int& p: par)
// cin >> p;

// Tree a(par);
// a.print_tree();
// cout << a.height() << endl;

Tree t1({4, -1, 4, 1, 1}); TEST(t1, 3);

Tree t2({-1, 0, 4, 0, 3}); TEST(t2, 4);

Tree t3({9, 7, 5, 5, 2, 9, 9, 9, 2, -1}); TEST(t3, 4);

Tree t4({6, 6, 6, 6, 6, 2, -1, 2}); TEST(t4, 3);

Tree t5({}); TEST(t5, 0);

Tree t6({5, -1, 1, 1, 1, 6, 4}); TEST(t6, 5);

Tree t7({9, 2, -1, 1, 3, 4, 0, 4, 9, 7}); TEST(t7, 8);

Tree t8({9, 2}); TEST(t8, -1);

return 0;
}
```