**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 parent0, . . . , parentn−1, где parenti —

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 parent0 , ... , parentn−1. Для каждого 0 ≤ i ≤ n − 1, parenti — 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 |
|  | 5  4 -1 4 1 1 | 3 |  |
|  | 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;

}