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. 0304 Докучаев Р.А.

Преподаватель Берленко Т.А.

Санкт-Петербург
2021

Цель работы.

Изучить структуру данных — дерево на примере построения через последовательность индексов родителей вершин $parent_0$, …, $parent_{n-1}$, и последующего вычисления высоты этого дерева при помощи рекурсии.

Задание.

На вход программе подается корневое дерево с вершинами {0, . . . , n−1}, заданное как последовательность $parent_0$, . . . , $parent_{n-1}$, 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

Основные теоретические положения.

Были использованы заголовочные файлы языка C++ *iostream* и *vector*. Структура дерева была реализована при помощи двух классов – *tree_node* и *tree*. Класс *tree* содержит указатель на свой корень, а также методы для нахождения высоты дерева. Класс *tree_node* содержит вектор ссылок на своих потомков.

Выполнение работы.

1. Подключение заголовочных файлов
2. Объявление классов *tree_node* и *tree*
3. Реализация конструктора *tree(vector <int>& parent_index)* и деструктора класса *~tree* с функцией *void tree::destroyer(tree_node* top)*, которую деструктор использует для очистки памяти после завершения работы программы

4. Реализация метода класса tree_node add_child(tree_node* node), который осуществляет вставку нового узла в конец вектора ссылок
5. Реализация методов класса tree private_get_height(tree_node* top), который находит высоту дерева при помощи обхода в глубину, и get_height(), который является оболочкой метода private_get_height
6. Создание функции main(), в которой происходит считывание числа вершин дерева и списка родителей для каждой вершины, создаётся объект класса и при помощи его метода подсчитывается высота дерева

Тестирование.

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	OK
2.	5 -1 0 4 0 3	4	OK
3.	1 -1	1	OK
4.	0	0	OK
5.	12 -1 0 1 2 3 4 5 6 7 8 9 10 11 12	12	OK

Выводы.

Была изучена абстрактная структура данных – дерево, и была разработана программа, осуществляющая подсчёт высоты дерева, введённого пользователем.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include "tree.h"
#include <vector>

int main(){
    int n = 0;
    cin >> n;
    vector <int> parent_index(n);
    for(int i = 0; i < n; i++){
        cin >> parent_index[i];
    }

    tree w_tree(parent_index);
    cout << w_tree.get_height() << endl;
    return 0;
}
```

Название файла: tree.h

```
#ifndef TREE_H
#define TREE_H

#include <vector>
using namespace std;

class tree_node{
    public:
        vector <tree_node*> children;
        void add_child(tree_node* node);
};

class tree{
    private:
        tree_node* root;

        unsigned int private_get_height(tree_node* top);
        void destroyer(tree_node* top);

    public:
        tree(vector <int>& parent_index);
        ~tree();
        unsigned int get_height();
};

#endif
```

Название файла: tree.cpp

```
#include "tree.h"

void tree_node::add_child(tree_node* node){
    children.push_back(node);
}
```

```
tree::tree(vector <int>& parent_index){
    root = nullptr;
    vector <tree_node*> nodes(parent_index.size());
    for(int i = 0; i < parent_index.size(); i++)
        nodes[i] = new tree_node();
    for(int i = 0; i < parent_index.size(); i++){
        int element = parent_index[i];
        if(element == -1)
            root = nodes[i];
        else
            nodes[element]->add_child(nodes[i]);
    }
}

tree::~tree(){
    if(root){
        destroyer(root);
    }
}

void tree::destroyer(tree_node* top){
    if(!top) return;

    size_t n = top->children.size();
    for(size_t i = 0; i < n; i++){
        destroyer(top->children[i]);
    }

    delete top;
}

unsigned int tree::get_height(){
    return private_get_height(root);
}

unsigned int tree::private_get_height(tree_node* top){
    if(!top) return 0;
    unsigned int top_max = 0;
    for(size_t i = 0; i < top->children.size(); i++){
        unsigned int height = private_get_height(top->children[i]);
        if(height > top_max) top_max = height;
    }
    return top_max + 1;
}
```