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: _____ Шквиря Е.В.

Преподаватель: _____ Берленко Т. А.

Санкт-Петербург

2021

Цель работы.

Изучить способы работы со структурой данных Дерево, задающееся с помощью последовательности чисел, определяющие отношение между вершинами, и вычисление его высоты.

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.

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

Выполнение работы.

Порядок выполнения поставленной задачи программой:

1) Для удобства и последующего тестирование программы на тестах выполнение начало выполнения задачи было вынесено в отдельную функцию *solve*, которой на вход подаётся размер дерева и массив связей вершин графа.
2) Для хранения вершины и дерева в целом была реализована структура *Node* и класс *Tree*. *Node* хранит в себе массив рёбер, через которыем можно пройти к следующей вершине. *Tree* хранит в себе информацию о корне и вершинах дерева, а также содержит методы для работы с деревом.
3) Вычисление высоты дерева происходит с помощью рекурсивного алгоритма путём спуска в «подграфы» каждой вершины и поиска среди них максимальной высоты.
4) Для тестирования программы была создана функция *test*, которой на вход подаётся ответ на тест, ссылка на функцию для выполнения задачи и тестовые данных на вход. В случае правильного ответа в консоль выведется «OK», иначе буде выполнен выброс исключения с информацией о разных ответах.

Разработанный программный код см. в приложении А.

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	Верный ответ
2.	5 -1 0 4 0 3	4	Верный ответ
3.	1 -1	1	Верный ответ
4.	0	0	Верный ответ
5.	8 -1 0 1 1 3 0 5 4	5	Верный ответ
6.	10 -1 0 1 1 3 0 5 4 6 8	5	Верный ответ
7.	3 -1 0 1	3	Верный ответ
8.	2 -1 0	2	Верный ответ

Выводы.

В ходе работы было изучено устройство структуры данных Дерево и способ его обхода с целью получения высоты дерева.

Разработана программа, получающая от пользователя размер дерева и массив с отношениями между вершинам и в дальнейшем высчитывающая высоту дерева. Программа тестировалась на граничных случаях и общих тестах.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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


struct Node {
	std::vector<int> edges;
	Node(std::vector<int> edges) : edges(edges) {}
	Node() {}
};


class Tree {
private:
	int root;
	std::vector<Node> tree;


public:
	Tree(int size) {
		root = -1;
		tree.resize(size);
	}


	bool setRoot(int root) {
		if (0 <= root && root < tree.size()) {
			this->root = root;
			return true;
		}
		else return false;
	}


	bool putEdge(int parent, int child) {
		if (0 <= parent && parent < tree.size() && 0 <= child && child <= tree.size())
{
```

```
                tree[parent].edges.push_back(child);
                return true;
            }
            else return false;
        }

        int getRoot() const {
            return this->root;
        }

        int calcHeights(int startNode) {
            if (startNode < 0 || startNode >= tree.size())
                return 0;
            int maxHeight = 0;
            for(auto node : tree[startNode].edges)
                maxHeight = std::max(maxHeight, calcHeights(node));
            return maxHeight + 1;
        }
};

int solve(int n, std::vector<int> parents) {
    int size = n;
    Tree tree = Tree(size);
    int parent;
    for (int i = 0; i < size; ++i) {
        parent = parents[i];
        if (parent == -1)
            tree.setRoot(i);
        else
            tree.putEdge(parent, i);
    }
    return tree.calcHeights(tree.getRoot());
}

int main() {
#ifdef TESTING
    test(3, solve, 5, {4, -1, 4, 1, 1});
    test(4, solve, 5, {-1, 0, 4, 0, 3});
```

```
	test(1, solve, 1, {-1});
	test(0, solve, 0, {});
	test(5, solve, 8, {-1, 0, 1, 1, 3, 0, 5, 4});
	test(5, solve, 10, {-1, 0, 1, 1, 3, 0, 5, 4, 6, 8});
	test(3, solve, 3, {-1, 0, 1});
	test(2, solve, 2, {-1, 0});
#else
	int size;
	std::cin >> size;
	std::vector<int> parents(size);
	for(auto &parent : parents)
		std::cin >> parent;
	std::cout << solve(size, parents);
#endif
	return 0;
}
```

Название файла: test.h

```
//#define TESTING

void test(int answer, int (*solve)(int, std::vector<int>), int size, std::vector<int> tree) {
	try {
		int result = (*solve)(size, tree);
		if (result != answer) {
			throw std::runtime_error("Exception, " + std::to_string(result) + " != "
+ std::to_string(answer));
		}
	} catch (std::runtime_error error) {
		std::cout << error.what() << '\n';
		return;
	}
	std::cout << "OK\n";
}
```