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.

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Цель работы.

Реализовать программу, производящую вычисление и вывод высоты дерева, заданного как последовательность чисел, и тестирование к ней.

Задание.

Вычисление высоты дерева. C++

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 $0 \le i \le n - 1$, 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.

Примеры

Вход:

5

4 -1 4 1 1

Выход:

3

Выполнение работы.

Основную задачу вычисления высоты дерева выполняет рекурсивная функция find_tree_height(tree, root, height, result, ind). Она принимает на вход

список (дерево, заданное последовательностью чисел), вершина дерева (актуальное состояние вычисления) — в начале программы (первый цикл рекурсии) это корень, высота ветви, по которой идёт рекурсия и наибольшая высота на данный момент времени.

Дети вершины вычисляются с помощью метода index(). В случае их отсутствия с помощью обработки исключений (try — except) ошибки не возникает. Если детей нет, то эта вершина — лист, ветвь рекурсии завершается и возвращает максимальную высоту дерева на данный момент вычислений. При наличии детей на каждую развилку производится новый вызов функции *find_tree_heigh* — запускаются новые пути рекурсии. Перемещаясь по вершинам, высота ветви начисляется счётчиком *height*. Когда все ветви будут прочитаны рекурсией до конца, программа откатится к первому циклу рекурсии и вернёт высоту дерева.

Производится ввод клавиатуры количество вершин (по условию задачи, однако в данном алгоритме не используется) и последовательность чисел (считывается в список). На экран выводится результат работы рекурсивной функции. Затем идут функции тестирования программы.

Тестирование.

Написаны модульные тесты. Для этого реализован шаблон теста *pattern_test_for_tree(test_tree, answer),* который принимает на вход список (дерево, заданное последовательностью чисел) и верный ответ теста. Для отладки использована инструкция *assert,* параметром которой является текстовое сообщение, выводящее пользователю необходимую информацию об ошибке в случае её возникновения.

Далее реализована функция *test_for_tree(),* вызываемая в коде программы один раз, производящая все заданные тестовые случаи. В случае успешного прохождения всех тестов программа также выводит соответствующее сообщение.

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	Программа работает верно
2.	10 1 2 3 4 5 -1 4 3 3 6	6	Программа работает верно
3.	14 0 0 0 -1 2 3 4 5 5 5 5 5 8 9	4	Программа работает верно
4.	5 -1 -99 139 0 11	2	Программа работает верно

Выводы.

Были освоены навыки работы с деревьями и принципы тестирования. Разработана программа, вычисляющая и выводящая на экран высоту дерева, заданного с клавиатуры в виде последовательности чисел. Реализовано осуществлено модульное тестирование.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb1.py

```
def find_tree_height(tree, root, height, result, ind):
    height += 1
    try:
        child = tree.index(root, ind)
    except ValueError:
        result = max(result, height)
        return result
    try:
        brother = tree.index(root, child+1)
    except ValueError:
        brother = -1
    if brother >= 0:
        result = max(result, find_tree_height(tree, root, height-1,
result, child+1))
    result = max(result, find_tree_height(tree, child, height, result,
0))
    return result
count = int(input())
tree = [int(i) for i in input().split()]
print(find_tree_height(tree, int(tree.index(-1)), 0, 0, 0))

def pattern_test_for_tree(test_tree, answer):
    assert find_tree_height(test_tree,int(test_tree.index(-1)), 0, 0,
0) == answer,\
        "\nTest: {}\nGot: {}\nExpected: {}"\
        .format(test_tree,
find_tree_height(test_tree,int(test_tree.index(-1)), 0, 0, 0),
                answer)
def test_for_tree():
    test_tree = [4, -1, 4, 1, 1]
    answer = 3
    pattern_test_for_tree(test_tree, answer)
```

```
    test_tree = [1, 2, 3, 4, 5, -1, 4, 3, 3, 6]
    answer = 6
    pattern_test_for_tree(test_tree, answer)

    test_tree = [0, 0, 0, -1, 2, 3, 4, 5, 5, 5, 5, 5, 8, 9]
    answer = 4
    pattern_test_for_tree(test_tree, answer)

    test_tree = [-1, -99, 139, 0, 11]
    answer = 2
    pattern_test_for_tree(test_tree, answer)

test_for_tree()

print("All tests were successfully passed")
```