ВШЭ АиСД 2021. Деревья (noSTL)

4 дек 2021, 18:44:25
старт: 22 окт 2021, 12:00:00
финиш: 31 окт 2021, 23:59:59
длительность: 9д. 11ч.
начало: 22 окт 2021, 12:00:00
конец: 31 окт 2021, 23:59:59

A. Высота дерева (0.25)

Ограничение времени	2 секунды
Ограничение памяти	64.0 Мб
Ввод	стандартный ввод
Вывод	стандартный вывод

Реализуйте **бинарное дерево поиска** для **целых** чисел. Программа получает на вход последовательность целых чисел и строит из них дерево. Элементы в деревья добавляются в соответствии с результатом поиска их места. Если элемент уже существует в дереве, добавлять его не надо. Балансировка дерева не производится.

Формат ввода

На вход программа получает последовательность натуральных чисел (пусть n - длина последовательности, $0 \le n \le 2 \cdot 10^4$; числа $\in [1; 10^9]$). Последовательность завершается числом 0 (оно не входит в последовательность), которое означает конец ввода, и добавлять его в дерево не надо.

Формат вывода

Выведите единственное число – высоту получившегося дерева.

Система оценки

Группа	Баллы	Доп. ограничения	Необх. группы	Комментарий
		n		
0	1	–	–	Тесты из условия.
1	0,5	$n \le 50$	0	
2	1,5	$n \le 10^3$	0 – 1	
3	3	$n \le 10^4$	0 – 2	
4	1	$n \le 11000$	0 – 3	**Offline-проверка**
5	1	$n \le 2 \cdot 10^4$	0 – 4	**Offline-проверка**

Пример

Ввод	Вывод
7 3 2 1 9 5 4 6 8 0	4

Набрать здесь | Отправить файл

```cpp
#include <iostream>

struct TreeNode {
    int value;
    TreeNode *leftChildren;
    TreeNode *rightChildren;

    explicit TreeNode(int new_value) {
        value = new_value;
        leftChildren = nullptr;
        rightChildren = nullptr;
    }
};

class Tree {
private:
    TreeNode *root_;

    TreeNode *insertChildren(TreeNode *tree_node, int value) {
        if (tree_node == nullptr) {
            tree_node = new TreeNode(value);
        } else if (value < tree_node->value) {
            tree_node->leftChildren = insertChildren(tree_node->leftChildren, value);
        } else if (value > tree_node->value) {
            tree_node->rightChildren = insertChildren(tree_node->rightChildren, value);
        }

        return tree_node;
    }

    void deleteNode(TreeNode *tree_node) {
        if (tree_node) {
            deleteNode(tree_node->leftChildren);
            deleteNode(tree_node->rightChildren);
            delete tree_node;
        }
    }

```

Отправить

Следующая