**Министерство образования и науки Российской Федерации**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,   
МЕХАНИКИ И ОПТИКИ**

Факультет программной инженерии и компьютерной техники  
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Aлгоритмы и структуры данных»

**ОТЧЁТ**

по лабораторной работе №7 (Week 7 Openedu)

Студент Луговских Савелий Михайлович Р3218

Преподаватель Муромцев Дмитрий Ильич

Санкт-Петербург

2019 г.

Содержание

[Задача 1 Проверка сбалансированности 3](#_Toc5699561)

[Исходный код к задаче 1 4](#_Toc5699562)

[Бенчмарк к задаче 1 5](#_Toc5699563)

[Задача 2. Делаю я левый поворот… 12](#_Toc5699564)

[Исходный код к задаче 2 13](#_Toc5699565)

[Бенчмарк к задаче 2 20](#_Toc5699566)

[Задача 3 Вставка в АВЛ-дерево 28](#_Toc5699567)

[Исходный код к задаче 3 29](#_Toc5699568)

[Бенчмарк к задаче 3 36](#_Toc5699569)

[Задача 4 Удаление из АВЛ-дерева 43](#_Toc5699570)

[Исходный код к задаче 4 45](#_Toc5699571)

[Бенчмарк к задаче 4 52](#_Toc5699572)

[Задача 5 Упорядоченное множество на АВЛ-дереве 59](#_Toc5699573)

[Исходный код к задаче 5 60](#_Toc5699574)

[Бенчмарк к задаче 5 68](#_Toc5699575)

# Задача 1 Проверка сбалансированности

1.0 из 1.0 балла (оценивается)

|  |  |
| --- | --- |
| Имя входного файла: | input.txt |
| Имя выходного файла: | output.txt |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 мегабайт |

АВЛ-дерево является сбалансированным в следующем смысле: для любой вершины высота ее левого поддерева отличается от высоты ее правого поддерева не больше, чем на единицу.

Введем понятие баланса вершины: для вершины дерева V ее баланс B(V) равен разности высоты правого поддерева и высоты левого поддерева. Таким образом, свойство АВЛ-дерева, приведенное выше, можно сформулировать следующим образом: для любой ее вершины V выполняется следующее неравенство:

**Обратите внимание, что, по историческим причинам, определение баланса в этой и последующих задачах этой недели "зеркально отражено" по сравнению с определением баланса в лекциях!** Надеемся, что этот факт не доставит Вам неудобств. В литературе по алгоритмам — как российской, так и мировой — ситуация, как правило, примерно та же.

Дано двоичное дерево поиска. Для каждой его вершины требуется определить ее баланс.

#### Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число N (1 — число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В (i+1)-ой строке файла (1 находится описание i-ой вершины, состоящее из трех чисел Ki, Li, Ri, разделенных пробелами — ключа в i-ой вершине , номера левого ребенка i-ой вершины ( I < Li или Li = 0, если левого ребенка нет) и номера правого ребенка i-ой вершины (I < R или Ri = 0, если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является деревом поиска.

#### Формат выходного файла

Для i-ой вершины в i-ой строке выведите одно число — баланс данной вершины.

#### Пример

|  |  |
| --- | --- |
| input.txt | output.txt |
| 6 -2 0 2 8 4 3 9 0 0 3 6 5 6 0 0 0 0 0 | 3 -1 0 0 0 0 |

## Исходный код к задаче 1

class Lab7\_1

{

public static void Main(string[] args)

{

var app = new Lab7\_1();

app.DoWork(args);

}

private void DoWork(string[] args)

{

using (var sw = new StreamWriter("output.txt"))

{

var stdin = File.ReadAllLines("input.txt");

var n = long.Parse(stdin[0]);

var tree = new Tree(n);

tree.Parse(stdin);

for (int i = 0; i < n; i++)

sw.WriteLine(TreeNode<long>.GetBalance(tree.Nodes[i]));

}

}

}

public class Tree

{

public TreeNode<long>[] Nodes { get; private set; }

private long \_nodesCount;

List<TreeNode<long>> \_leafs = new List<TreeNode<long>>();

public Tree(long n)

{

\_nodesCount = n;

this.Nodes = new TreeNode<long>[n];

}

//Parsing

public void Parse(string[] stdin)

{

for (int i = 1; i <= \_nodesCount; i++)

{

var temp = stdin[i].Split(' ').Select(x => long.Parse(x)).ToArray();

if (this.Nodes[i - 1] == null)

this.Nodes[i - 1] = new TreeNode<long>();

this.Nodes[i - 1].Key = temp[0];

//Left child

if (temp[1] != 0)

{

if (this.Nodes[temp[1] - 1] == null)

this.Nodes[temp[1] - 1] = new TreeNode<long>()

{

Parent = this.Nodes[i - 1]

};

this.Nodes[i - 1].Left = this.Nodes[temp[1] - 1];

}

//Right child

if (temp[2] != 0)

{

if (temp[2] != 0 && this.Nodes[temp[2] - 1] == null)

this.Nodes[temp[2] - 1] = new TreeNode<long>() { Parent = this.Nodes[i - 1] };

this.Nodes[i - 1].Right = this.Nodes[temp[2] - 1];

}

//Calc Height

if (temp[1] == 0 & temp[2] == 0)

{

TreeNode<long> leaf = this.Nodes[i - 1];

Stack<TreeNode<long>> curr = new Stack<TreeNode<long>>();

while (leaf != null)

{

curr.Push(leaf);

leaf = leaf.Parent;

}

while (curr.Count != 0)

{

leaf = curr.Pop();

if (leaf.Height < curr.Count)

leaf.Height = curr.Count;

}

}

}

}

}

public class TreeNode<T> where T : IComparable<T>

{

public T Key { get; set; }

public TreeNode<T> Parent { get; set; }

public TreeNode<T> Left { get; set; }

public TreeNode<T> Right { get; set; }

public long Height { get; set; }

public static long GetBalance(TreeNode<T> tree)

{

if (tree == null)

throw new ArgumentNullException("Tree does not exist!");

if (tree.Left != null && tree.Right != null)

return tree.Right.Height - tree.Left.Height;

if (tree.Left == null && tree.Right != null)

return tree.Right.Height + 1;

if (tree.Left != null && tree.Right == null)

return -1 - tree.Left.Height;

else

return 0;

}

}

## Бенчмарк к задаче 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ теста** | **Результат** | **Время, с** | **Память** | **Размер входного файла** | **Размер выходного файла** |
| Max |  | 0.500 | 51212288 | 3986010 | 1688889 |
| 1 | OK | 0.046 | 11689984 | 46 | 19 |
| 2 | OK | 0.062 | 11620352 | 10 | 3 |
| 3 | OK | 0.046 | 11653120 | 17 | 6 |
| 4 | OK | 0.031 | 11710464 | 17 | 7 |
| 5 | OK | 0.031 | 11591680 | 24 | 9 |
| 6 | OK | 0.031 | 11681792 | 24 | 10 |
| 7 | OK | 0.031 | 11661312 | 24 | 9 |
| 8 | OK | 0.031 | 11698176 | 24 | 10 |
| 9 | OK | 0.046 | 11624448 | 24 | 11 |
| 10 | OK | 0.046 | 11608064 | 31 | 12 |
| 11 | OK | 0.031 | 11644928 | 31 | 13 |
| 12 | OK | 0.031 | 11431936 | 31 | 12 |
| 13 | OK | 0.031 | 11411456 | 31 | 13 |
| 14 | OK | 0.015 | 11513856 | 31 | 14 |
| 15 | OK | 0.031 | 11431936 | 31 | 12 |
| 16 | OK | 0.031 | 11423744 | 31 | 13 |
| 17 | OK | 0.031 | 11423744 | 31 | 13 |
| 18 | OK | 0.031 | 11407360 | 31 | 14 |
| 19 | OK | 0.031 | 11468800 | 31 | 13 |
| 20 | OK | 0.031 | 11382784 | 31 | 14 |
| 21 | OK | 0.031 | 11407360 | 31 | 13 |
| 22 | OK | 0.031 | 11485184 | 31 | 14 |
| 23 | OK | 0.031 | 11415552 | 31 | 15 |
| 24 | OK | 0.031 | 11374592 | 38 | 15 |
| 25 | OK | 0.031 | 11427840 | 38 | 16 |
| 26 | OK | 0.046 | 11489280 | 38 | 15 |
| 27 | OK | 0.031 | 11431936 | 38 | 16 |
| 28 | OK | 0.031 | 11407360 | 38 | 17 |
| 29 | OK | 0.031 | 11411456 | 38 | 15 |
| 30 | OK | 0.015 | 11456512 | 38 | 16 |
| 31 | OK | 0.031 | 11395072 | 38 | 16 |
| 32 | OK | 0.031 | 11489280 | 38 | 17 |
| 33 | OK | 0.031 | 11419648 | 38 | 16 |
| 34 | OK | 0.046 | 11382784 | 38 | 17 |
| 35 | OK | 0.046 | 11399168 | 38 | 16 |
| 36 | OK | 0.046 | 11403264 | 38 | 17 |
| 37 | OK | 0.046 | 11472896 | 38 | 18 |
| 38 | OK | 0.031 | 11427840 | 38 | 15 |
| 39 | OK | 0.031 | 11399168 | 38 | 16 |
| 40 | OK | 0.031 | 11464704 | 38 | 15 |
| 41 | OK | 0.046 | 11460608 | 38 | 16 |
| 42 | OK | 0.046 | 11419648 | 38 | 17 |
| 43 | OK | 0.031 | 11436032 | 38 | 15 |
| 44 | OK | 0.031 | 11386880 | 38 | 16 |
| 45 | OK | 0.031 | 11411456 | 38 | 16 |
| 46 | OK | 0.031 | 11403264 | 38 | 17 |
| 47 | OK | 0.031 | 11395072 | 38 | 16 |
| 48 | OK | 0.031 | 11427840 | 38 | 17 |
| 49 | OK | 0.031 | 11423744 | 38 | 16 |
| 50 | OK | 0.031 | 11522048 | 38 | 17 |
| 51 | OK | 0.031 | 11538432 | 38 | 18 |
| 52 | OK | 0.031 | 11468800 | 38 | 16 |
| 53 | OK | 0.015 | 11399168 | 38 | 17 |
| 54 | OK | 0.031 | 11481088 | 38 | 16 |
| 55 | OK | 0.031 | 11399168 | 38 | 17 |
| 56 | OK | 0.046 | 11423744 | 38 | 18 |
| 57 | OK | 0.046 | 11423744 | 38 | 16 |
| 58 | OK | 0.031 | 11464704 | 38 | 17 |
| 59 | OK | 0.031 | 11415552 | 38 | 17 |
| 60 | OK | 0.046 | 11436032 | 38 | 18 |
| 61 | OK | 0.031 | 11468800 | 38 | 17 |
| 62 | OK | 0.015 | 11476992 | 38 | 18 |
| 63 | OK | 0.062 | 11423744 | 38 | 17 |
| 64 | OK | 0.031 | 11505664 | 38 | 18 |
| 65 | OK | 0.031 | 11386880 | 38 | 19 |
| 66 | OK | 0.031 | 11411456 | 45 | 18 |
| 67 | OK | 0.031 | 11505664 | 45 | 19 |
| 68 | OK | 0.031 | 11419648 | 45 | 18 |
| 69 | OK | 0.031 | 11436032 | 45 | 19 |
| 70 | OK | 0.046 | 11411456 | 45 | 20 |
| 71 | OK | 0.031 | 11411456 | 45 | 18 |
| 72 | OK | 0.046 | 11472896 | 45 | 19 |
| 73 | OK | 0.031 | 11485184 | 45 | 19 |
| 74 | OK | 0.031 | 11505664 | 45 | 20 |
| 75 | OK | 0.031 | 11464704 | 45 | 19 |
| 76 | OK | 0.015 | 11411456 | 45 | 20 |
| 77 | OK | 0.031 | 11419648 | 45 | 19 |
| 78 | OK | 0.031 | 11423744 | 45 | 20 |
| 79 | OK | 0.031 | 11419648 | 45 | 21 |
| 80 | OK | 0.015 | 11419648 | 45 | 18 |
| 81 | OK | 0.031 | 11427840 | 45 | 19 |
| 82 | OK | 0.031 | 11431936 | 45 | 18 |
| 83 | OK | 0.031 | 11415552 | 45 | 19 |
| 84 | OK | 0.015 | 11485184 | 45 | 20 |
| 85 | OK | 0.046 | 11493376 | 45 | 18 |
| 86 | OK | 0.031 | 11436032 | 45 | 19 |
| 87 | OK | 0.031 | 11419648 | 45 | 19 |
| 88 | OK | 0.031 | 11403264 | 45 | 20 |
| 89 | OK | 0.031 | 11431936 | 45 | 19 |
| 90 | OK | 0.015 | 11423744 | 45 | 20 |
| 91 | OK | 0.031 | 11419648 | 45 | 19 |
| 92 | OK | 0.046 | 11501568 | 45 | 20 |
| 93 | OK | 0.031 | 11407360 | 45 | 21 |
| 94 | OK | 0.031 | 11419648 | 45 | 19 |
| 95 | OK | 0.046 | 11476992 | 45 | 20 |
| 96 | OK | 0.046 | 11436032 | 45 | 19 |
| 97 | OK | 0.046 | 11452416 | 45 | 20 |
| 98 | OK | 0.031 | 11427840 | 45 | 21 |
| 99 | OK | 0.031 | 11452416 | 45 | 19 |
| 100 | OK | 0.046 | 11403264 | 45 | 20 |
| 101 | OK | 0.046 | 11419648 | 45 | 20 |
| 102 | OK | 0.031 | 11415552 | 45 | 21 |
| 103 | OK | 0.031 | 11431936 | 45 | 20 |
| 104 | OK | 0.031 | 11415552 | 45 | 21 |
| 105 | OK | 0.031 | 11403264 | 45 | 20 |
| 106 | OK | 0.046 | 11522048 | 45 | 21 |
| 107 | OK | 0.031 | 11423744 | 45 | 22 |
| 108 | OK | 0.031 | 11485184 | 45 | 18 |
| 109 | OK | 0.031 | 11497472 | 45 | 19 |
| 110 | OK | 0.031 | 11427840 | 45 | 18 |
| 111 | OK | 0.031 | 11423744 | 45 | 19 |
| 112 | OK | 0.031 | 11436032 | 45 | 20 |
| 113 | OK | 0.046 | 11472896 | 45 | 18 |
| 114 | OK | 0.031 | 11485184 | 45 | 19 |
| 115 | OK | 0.015 | 11436032 | 45 | 19 |
| 116 | OK | 0.031 | 11464704 | 45 | 20 |
| 117 | OK | 0.031 | 11530240 | 45 | 19 |
| 118 | OK | 0.046 | 11489280 | 45 | 20 |
| 119 | OK | 0.031 | 11411456 | 45 | 19 |
| 120 | OK | 0.046 | 11493376 | 45 | 20 |
| 121 | OK | 0.031 | 11419648 | 45 | 21 |
| 122 | OK | 0.031 | 11395072 | 45 | 18 |
| 123 | OK | 0.062 | 11427840 | 45 | 19 |
| 124 | OK | 0.031 | 11476992 | 45 | 18 |
| 125 | OK | 0.015 | 11476992 | 45 | 19 |
| 126 | OK | 0.031 | 11419648 | 45 | 20 |
| 127 | OK | 0.031 | 11411456 | 45 | 19 |
| 128 | OK | 0.015 | 11472896 | 45 | 20 |
| 129 | OK | 0.031 | 11493376 | 45 | 19 |
| 130 | OK | 0.031 | 11460608 | 45 | 20 |
| 131 | OK | 0.031 | 11444224 | 45 | 21 |
| 132 | OK | 0.046 | 11419648 | 45 | 19 |
| 133 | OK | 0.031 | 11431936 | 45 | 20 |
| 134 | OK | 0.031 | 11411456 | 45 | 20 |
| 135 | OK | 0.046 | 11419648 | 45 | 21 |
| 136 | OK | 0.031 | 11476992 | 45 | 18 |
| 137 | OK | 0.046 | 11431936 | 45 | 19 |
| 138 | OK | 0.031 | 11546624 | 45 | 20 |
| 139 | OK | 0.031 | 11427840 | 45 | 21 |
| 140 | OK | 0.031 | 11419648 | 45 | 21 |
| 141 | OK | 0.031 | 11427840 | 45 | 22 |
| 142 | OK | 0.031 | 11472896 | 45 | 19 |
| 143 | OK | 0.031 | 11419648 | 45 | 20 |
| 144 | OK | 0.046 | 11415552 | 45 | 19 |
| 145 | OK | 0.031 | 11427840 | 45 | 20 |
| 146 | OK | 0.015 | 11431936 | 45 | 21 |
| 147 | OK | 0.031 | 11489280 | 45 | 19 |
| 148 | OK | 0.046 | 11407360 | 45 | 20 |
| 149 | OK | 0.031 | 11476992 | 45 | 20 |
| 150 | OK | 0.015 | 11460608 | 45 | 21 |
| 151 | OK | 0.031 | 11415552 | 45 | 20 |
| 152 | OK | 0.031 | 11452416 | 45 | 21 |
| 153 | OK | 0.031 | 11436032 | 45 | 20 |
| 154 | OK | 0.031 | 11427840 | 45 | 21 |
| 155 | OK | 0.031 | 11415552 | 45 | 22 |
| 156 | OK | 0.015 | 11423744 | 45 | 19 |
| 157 | OK | 0.015 | 11411456 | 45 | 20 |
| 158 | OK | 0.031 | 11476992 | 45 | 19 |
| 159 | OK | 0.031 | 11395072 | 45 | 20 |
| 160 | OK | 0.031 | 11456512 | 45 | 21 |
| 161 | OK | 0.031 | 11509760 | 45 | 19 |
| 162 | OK | 0.046 | 11411456 | 45 | 20 |
| 163 | OK | 0.031 | 11448320 | 45 | 20 |
| 164 | OK | 0.015 | 11481088 | 45 | 21 |
| 165 | OK | 0.031 | 11395072 | 45 | 20 |
| 166 | OK | 0.031 | 11411456 | 45 | 21 |
| 167 | OK | 0.046 | 11407360 | 45 | 20 |
| 168 | OK | 0.031 | 11411456 | 45 | 21 |
| 169 | OK | 0.046 | 11456512 | 45 | 22 |
| 170 | OK | 0.031 | 11415552 | 45 | 19 |
| 171 | OK | 0.031 | 11419648 | 45 | 20 |
| 172 | OK | 0.031 | 11485184 | 45 | 19 |
| 173 | OK | 0.031 | 11407360 | 45 | 20 |
| 174 | OK | 0.031 | 11509760 | 45 | 21 |
| 175 | OK | 0.031 | 11427840 | 45 | 19 |
| 176 | OK | 0.031 | 11419648 | 45 | 20 |
| 177 | OK | 0.031 | 11436032 | 45 | 20 |
| 178 | OK | 0.046 | 11407360 | 45 | 21 |
| 179 | OK | 0.031 | 11427840 | 45 | 20 |
| 180 | OK | 0.046 | 11472896 | 45 | 21 |
| 181 | OK | 0.046 | 11481088 | 45 | 20 |
| 182 | OK | 0.031 | 11468800 | 45 | 21 |
| 183 | OK | 0.031 | 11460608 | 45 | 22 |
| 184 | OK | 0.031 | 11505664 | 45 | 20 |
| 185 | OK | 0.031 | 11489280 | 45 | 21 |
| 186 | OK | 0.031 | 11481088 | 45 | 20 |
| 187 | OK | 0.031 | 11415552 | 45 | 21 |
| 188 | OK | 0.031 | 11431936 | 45 | 22 |
| 189 | OK | 0.046 | 11403264 | 45 | 20 |
| 190 | OK | 0.046 | 11431936 | 45 | 21 |
| 191 | OK | 0.031 | 11440128 | 45 | 21 |
| 192 | OK | 0.031 | 11476992 | 45 | 22 |
| 193 | OK | 0.031 | 11419648 | 45 | 21 |
| 194 | OK | 0.031 | 11415552 | 45 | 22 |
| 195 | OK | 0.031 | 11481088 | 45 | 21 |
| 196 | OK | 0.031 | 11485184 | 45 | 22 |
| 197 | OK | 0.031 | 11472896 | 45 | 23 |
| 198 | OK | 0.046 | 11411456 | 221 | 55 |
| 199 | OK | 0.031 | 11419648 | 220 | 59 |
| 200 | OK | 0.031 | 11407360 | 220 | 46 |
| 201 | OK | 0.031 | 11415552 | 223 | 48 |
| 202 | OK | 0.031 | 11493376 | 226 | 45 |
| 203 | OK | 0.031 | 11460608 | 1786 | 502 |
| 204 | OK | 0.031 | 11534336 | 1785 | 555 |
| 205 | OK | 0.031 | 11476992 | 1785 | 445 |
| 206 | OK | 0.031 | 11554816 | 1845 | 365 |
| 207 | OK | 0.031 | 11620352 | 1847 | 363 |
| 208 | OK | 0.031 | 11890688 | 9555 | 3006 |
| 209 | OK | 0.031 | 11804672 | 9554 | 3297 |
| 210 | OK | 0.031 | 11812864 | 9554 | 2730 |
| 211 | OK | 0.031 | 11890688 | 9303 | 1888 |
| 212 | OK | 0.031 | 11853824 | 9984 | 1877 |
| 213 | OK | 0.031 | 12644352 | 37691 | 12907 |
| 214 | OK | 0.031 | 12603392 | 37690 | 13974 |
| 215 | OK | 0.078 | 12599296 | 37690 | 11820 |
| 216 | OK | 0.046 | 12541952 | 39602 | 7150 |
| 217 | OK | 0.031 | 12619776 | 38744 | 7125 |
| 218 | OK | 0.062 | 18423808 | 178903 | 63876 |
| 219 | OK | 0.078 | 18452480 | 178902 | 68889 |
| 220 | OK | 0.046 | 18403328 | 178902 | 58890 |
| 221 | OK | 0.046 | 19156992 | 185712 | 33049 |
| 222 | OK | 0.078 | 19148800 | 180580 | 33013 |
| 223 | OK | 0.250 | 35704832 | 1853240 | 724890 |
| 224 | OK | 0.234 | 35717120 | 1853239 | 773873 |
| 225 | OK | 0.250 | 35700736 | 1853239 | 675751 |
| 226 | OK | 0.265 | 35557376 | 1855624 | 324156 |
| 227 | OK | 0.281 | 35532800 | 1856715 | 324455 |
| 228 | OK | 0.453 | 48099328 | 3473125 | 1412256 |
| 229 | OK | 0.406 | 48447488 | 3473124 | 1501788 |
| 230 | OK | 0.421 | 48394240 | 3473124 | 1322578 |
| 231 | OK | 0.453 | 46665728 | 3603994 | 592172 |
| 232 | OK | 0.453 | 46657536 | 3646224 | 592525 |
| 233 | OK | 0.484 | 51212288 | 3888905 | 1589032 |
| 234 | OK | 0.484 | 51130368 | 3888904 | 1688889 |
| 235 | OK | 0.437 | 51187712 | 3888904 | 1488890 |
| 236 | OK | 0.500 | 49221632 | 3890628 | 661024 |
| 237 | OK | 0.484 | 49696768 | 3986010 | 661067 |

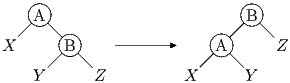
# Задача 2. Делаю я левый поворот…

|  |  |
| --- | --- |
| Имя входного файла: | input.txt |
| Имя выходного файла: | output.txt |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 мегабайт |

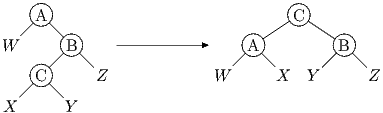
Для балансировки АВЛ-дерева при операциях вставки и удаления производятся левые и правые повороты. Левый поворот в вершине производится, когда баланс этой вершины больше 1, аналогично, правый поворот производится при балансе, меньшем −1.

Существует два разных левых (как, разумеется, и правых) поворота: большой и малый левый поворот.

Малый левый поворот осуществляется следующим образом:



Заметим, что если до выполнения малого левого поворота был нарушен баланс только корня дерева, то после его выполнения все вершины становятся сбалансированными, за исключением случая, когда у правого ребенка корня баланс до поворота равен −1. В этом случае вместо малого левого поворота выполняется большой левый поворот, который осуществляется так:



Дано дерево, в котором баланс корня равен 2. Сделайте левый поворот.

#### Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число N (3 — число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В (i+1)-ой строке файла (1 находится описание i-ой вершины, состоящее из трех чисел Ki, Li, Ri, разделенных пробелами — ключа в i-ой вершине , номера левого ребенка i-ой вершины ( I < Li или Li = 0, если левого ребенка нет) и номера правого ребенка i-ой вершины (I < R или Ri = 0, если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является деревом поиска. Баланс корня дерева (вершины с номером 1) равен 2, баланс всех остальных вершин находится в пределах от -1 до 1.

#### Формат выходного файла

Выведите в том же формате дерево после осуществления левого поворота. Нумерация вершин может быть произвольной при условии соблюдения формата. Так, номер вершины должен быть меньше номера ее детей.

#### Пример

|  |  |
| --- | --- |
| input.txt | output.txt |
| 7 -2 7 2 8 4 3 9 0 0 3 6 5 6 0 0 0 0 0 -7 0 0 | 7 3 2 3 -2 4 5 8 6 7 -7 0 0 0 0 0 6 0 0 9 0 0 |

## Исходный код к задаче 2

class Lab7\_2

{

public static void Main(string[] args)

{

var app = new Lab7\_2();

app.DoWork(args);

}

private void DoWork(string[] args)

{

using (var sw = new StreamWriter("output.txt"))

{

string[] stdin = File.ReadAllLines("input.txt");

TreeNode<long> root = null;

for (int i = 1; i < stdin.Length; i++)

root = TreeNode<long>.Insert(root, new TreeNode<long> { Key = long.Parse(stdin[i].Split(' ')[0]) });

root = TreeNode<long>.Balance(root);

sw.WriteLine(stdin[0]);

TreeNode<long>.PrintTree(sw, root);

}

}

class TreeNode<T> where T : IComparable<T>

{

public T Key { get; set; }

public TreeNode<T> Parent { get; set; }

public TreeNode<T> Left { get; set; }

public TreeNode<T> Right { get; set; }

private long Depth { get; set; }

public long Height { get; private set; }

public static TreeNode<T> Previous(TreeNode<T> node)

{

if (node.Left == null)

return node;

return Maximum(node.Left);

}

public static TreeNode<T> Maximum(TreeNode<T> node)

{

while (node.Right != null)

node = node.Right;

return node;

}

/// <returns>Root of tree after remove</returns>

public static TreeNode<T> Remove(TreeNode<T> item)

{

TreeNode<T> parent = item.Parent;

//Leaf

if (item.Left == null && item.Right == null)

{

if (parent == null)

return null;

if (parent.Left == item)

parent.Left = null;

else

parent.Right = null;

UpdateHeight(parent);

return Balance(parent);

}

//One child

if ((item.Left == null) ^ (item.Right == null))

if (item.Left != null)

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Left;

else

parent.Right = item.Left;

UpdateHeight(parent);

}

item.Left.Parent = parent;

return Balance(item.Left);

}

else

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Right;

else

parent.Right = item.Right;

UpdateHeight(parent);

}

item.Right.Parent = parent;

return Balance(item.Right);

}

//Two child

if ((item.Left != null) && (item.Right != null))

{

TreeNode<T> prev = Previous(item);

Remove(prev);

item.Key = prev.Key;

}

return Balance(item);

}

/// <returns>Root of tree after insert</returns>

public static TreeNode<T> Insert(TreeNode<T> root, TreeNode<T> node)

{

if (root == null)

return node;

TreeNode<T> current = root;

while (true)

{

if (current.Key.CompareTo(node.Key) == 0)

throw new ArgumentException("Not unique key");

if (current.Key.CompareTo(node.Key) < 0)

{

if (current.Right != null)

current = current.Right;

else

{

current.Right = node;

node.Parent = current;

UpdateHeight(node);

return root;

}

}

else

{

if (current.Left != null)

current = current.Left;

else

{

current.Left = node;

node.Parent = current;

UpdateHeight(node);

return root;

}

}

}

}

private static void UpdateHeight(TreeNode<T> node)

{

while (node != null)

{

long rH = node.Right != null ? node.Right.Height : -1;

long lH = node.Left != null ? node.Left.Height : -1;

long currentH = node.Height;

if (rH > lH)

node.Height = rH + 1;

else

node.Height = lH + 1;

node = node.Parent;

}

}

/// <returns>Root of tree after balance</returns>

public static TreeNode<T> Balance(TreeNode<T> leaf)

{

TreeNode<T> current = leaf;

while (current != null)

{

long balance = GetBalance(current);

if (balance > 1)

{

if (GetBalance(current.Right) == -1)

current = BigLeftTurn(current);

else

current = SmallLeftTurn(current);

}

if (balance < -1)

{

if (GetBalance(current.Left) == 1)

current = BigRightTurn(current);

else

current = SmallRightTurn(current);

}

if (current.Parent == null)

return current;

else

current = current.Parent;

}

return current;

}

public static void PrintTree(StreamWriter sw, TreeNode<T> root)

{

if (root == null)

return;

Queue<TreeNode<T>> bfsQueue = new Queue<TreeNode<T>>();

long counter = 1;

bfsQueue.Enqueue(root);

while (bfsQueue.Count != 0)

{

TreeNode<T> current = bfsQueue.Dequeue();

sw.Write(current.Key);

if (current.Left != null)

{

bfsQueue.Enqueue(current.Left);

sw.Write(" " + ++counter);

}

else

sw.Write(" " + 0);

if (current.Right != null)

{

bfsQueue.Enqueue(current.Right);

sw.WriteLine(" " + ++counter);

}

else

sw.WriteLine(" " + 0);

}

}

public static long GetBalance(TreeNode<T> tree)

{

if (tree == null)

return 0;

if (tree.Left != null && tree.Right != null)

return tree.Right.Height - tree.Left.Height;

if (tree.Left == null && tree.Right != null)

return tree.Right.Height + 1;

if (tree.Left != null && tree.Right == null)

return -1 - tree.Left.Height;

else

return 0;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallLeftTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Left;

TreeNode<T> y = root.Right.Left;

TreeNode<T> z = root.Right.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = root;

if (z != null)

z.Parent = child;

//Childs

root.Left = x;

root.Right = y;

child.Left = root;

child.Right = z;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (xH > yH)

root.Height = xH + 1;

else

root.Height = yH + 1;

if (root.Height > zH)

child.Height = root.Height + 1;

else

child.Height = zH + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallRightTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Right;

TreeNode<T> y = root.Left.Left;

TreeNode<T> z = root.Left.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = child;

if (z != null)

z.Parent = root;

//Childs

root.Left = z;

root.Right = x;

child.Left = y;

child.Right = root;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (zH > xH)

root.Height = zH + 1;

else

root.Height = xH + 1;

if (y.Height > root.Height)

child.Height = yH + 1;

else

child.Height = root.Height + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigRightTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Left;

TreeNode<T> c = root.Left.Right;

TreeNode<T> z = b.Left;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = root;

if (x != null)

x.Parent = b;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = b;

c.Right = root;

b.Left = z;

b.Right = x;

root.Left = y;

root.Right = w;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (zH > xH)

b.Height = zH + 1;

else

b.Height = xH + 1;

if (yH > wH)

root.Height = yH + 1;

else

root.Height = wH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigLeftTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Right;

TreeNode<T> c = root.Right.Left;

TreeNode<T> z = b.Right;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = b;

if (x != null)

x.Parent = root;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = root;

c.Right = b;

b.Left = y;

b.Right = z;

root.Left = w;

root.Right = x;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (wH > xH)

root.Height = wH + 1;

else

root.Height = xH + 1;

if (yH > zH)

b.Height = yH + 1;

else

b.Height = zH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

}

}

## Бенчмарк к задаче 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ теста** | **Результат** | **Время, с** | **Память** | **Размер входного файла** | **Размер выходного файла** |
| Max |  | 0.500 | 50868224 | 3986416 | 3986416 |
| 1 | OK | 0.046 | 11198464 | 54 | 54 |
| 2 | OK | 0.031 | 11120640 | 24 | 24 |
| 3 | OK | 0.031 | 11083776 | 24 | 24 |
| 4 | OK | 0.046 | 11153408 | 31 | 31 |
| 5 | OK | 0.031 | 11112448 | 45 | 45 |
| 6 | OK | 0.046 | 11128832 | 45 | 45 |
| 7 | OK | 0.031 | 11124736 | 45 | 45 |
| 8 | OK | 0.031 | 11206656 | 45 | 45 |
| 9 | OK | 0.046 | 11120640 | 52 | 52 |
| 10 | OK | 0.046 | 11169792 | 52 | 52 |
| 11 | OK | 0.031 | 11104256 | 52 | 52 |
| 12 | OK | 0.031 | 10940416 | 52 | 52 |
| 13 | OK | 0.031 | 11022336 | 52 | 52 |
| 14 | OK | 0.031 | 10956800 | 52 | 52 |
| 15 | OK | 0.031 | 10969088 | 59 | 59 |
| 16 | OK | 0.031 | 10899456 | 59 | 59 |
| 17 | OK | 0.046 | 10895360 | 59 | 59 |
| 18 | OK | 0.046 | 10928128 | 59 | 59 |
| 19 | OK | 0.031 | 10956800 | 66 | 66 |
| 20 | OK | 0.031 | 10924032 | 75 | 75 |
| 21 | OK | 0.062 | 10960896 | 75 | 75 |
| 22 | OK | 0.031 | 11010048 | 75 | 75 |
| 23 | OK | 0.031 | 10989568 | 75 | 75 |
| 24 | OK | 0.031 | 10989568 | 75 | 75 |
| 25 | OK | 0.031 | 10924032 | 75 | 75 |
| 26 | OK | 0.031 | 10907648 | 75 | 75 |
| 27 | OK | 0.046 | 10895360 | 75 | 75 |
| 28 | OK | 0.031 | 10919936 | 75 | 75 |
| 29 | OK | 0.046 | 11018240 | 75 | 75 |
| 30 | OK | 0.015 | 10895360 | 75 | 75 |
| 31 | OK | 0.031 | 10903552 | 75 | 75 |
| 32 | OK | 0.031 | 10952704 | 75 | 75 |
| 33 | OK | 0.046 | 10911744 | 75 | 75 |
| 34 | OK | 0.015 | 10985472 | 75 | 75 |
| 35 | OK | 0.031 | 10907648 | 75 | 75 |
| 36 | OK | 0.046 | 10956800 | 75 | 75 |
| 37 | OK | 0.046 | 10899456 | 75 | 75 |
| 38 | OK | 0.015 | 10928128 | 75 | 75 |
| 39 | OK | 0.031 | 10874880 | 75 | 75 |
| 40 | OK | 0.031 | 10960896 | 75 | 75 |
| 41 | OK | 0.031 | 10874880 | 75 | 75 |
| 42 | OK | 0.046 | 11014144 | 75 | 75 |
| 43 | OK | 0.031 | 10924032 | 75 | 75 |
| 44 | OK | 0.031 | 10960896 | 75 | 75 |
| 45 | OK | 0.046 | 10973184 | 75 | 75 |
| 46 | OK | 0.031 | 10915840 | 75 | 75 |
| 47 | OK | 0.031 | 10964992 | 75 | 75 |
| 48 | OK | 0.031 | 10919936 | 75 | 75 |
| 49 | OK | 0.031 | 10874880 | 75 | 75 |
| 50 | OK | 0.031 | 10899456 | 75 | 75 |
| 51 | OK | 0.031 | 10964992 | 75 | 75 |
| 52 | OK | 0.031 | 10928128 | 84 | 84 |
| 53 | OK | 0.031 | 10948608 | 84 | 84 |
| 54 | OK | 0.031 | 10956800 | 84 | 84 |
| 55 | OK | 0.031 | 10969088 | 84 | 84 |
| 56 | OK | 0.015 | 10948608 | 84 | 84 |
| 57 | OK | 0.031 | 10985472 | 84 | 84 |
| 58 | OK | 0.031 | 10932224 | 84 | 84 |
| 59 | OK | 0.031 | 11055104 | 84 | 84 |
| 60 | OK | 0.031 | 10895360 | 84 | 84 |
| 61 | OK | 0.031 | 10907648 | 84 | 84 |
| 62 | OK | 0.015 | 11001856 | 84 | 84 |
| 63 | OK | 0.031 | 10915840 | 84 | 84 |
| 64 | OK | 0.031 | 10899456 | 84 | 84 |
| 65 | OK | 0.031 | 10985472 | 84 | 84 |
| 66 | OK | 0.031 | 10887168 | 84 | 84 |
| 67 | OK | 0.046 | 10936320 | 84 | 84 |
| 68 | OK | 0.031 | 10895360 | 84 | 84 |
| 69 | OK | 0.031 | 10977280 | 84 | 84 |
| 70 | OK | 0.031 | 10977280 | 84 | 84 |
| 71 | OK | 0.031 | 10895360 | 84 | 84 |
| 72 | OK | 0.031 | 10899456 | 84 | 84 |
| 73 | OK | 0.031 | 10924032 | 84 | 84 |
| 74 | OK | 0.031 | 10915840 | 84 | 84 |
| 75 | OK | 0.031 | 10940416 | 84 | 84 |
| 76 | OK | 0.031 | 10977280 | 84 | 84 |
| 77 | OK | 0.031 | 10936320 | 84 | 84 |
| 78 | OK | 0.031 | 10915840 | 84 | 84 |
| 79 | OK | 0.031 | 10977280 | 84 | 84 |
| 80 | OK | 0.031 | 10895360 | 84 | 84 |
| 81 | OK | 0.031 | 10948608 | 84 | 84 |
| 82 | OK | 0.046 | 10899456 | 84 | 84 |
| 83 | OK | 0.031 | 10915840 | 84 | 84 |
| 84 | OK | 0.031 | 10903552 | 84 | 84 |
| 85 | OK | 0.031 | 10899456 | 84 | 84 |
| 86 | OK | 0.031 | 10969088 | 84 | 84 |
| 87 | OK | 0.031 | 10903552 | 84 | 84 |
| 88 | OK | 0.031 | 10932224 | 84 | 84 |
| 89 | OK | 0.046 | 10932224 | 84 | 84 |
| 90 | OK | 0.031 | 10948608 | 84 | 84 |
| 91 | OK | 0.062 | 10936320 | 84 | 84 |
| 92 | OK | 0.031 | 10956800 | 84 | 84 |
| 93 | OK | 0.031 | 10915840 | 84 | 84 |
| 94 | OK | 0.046 | 10932224 | 84 | 84 |
| 95 | OK | 0.031 | 10936320 | 84 | 84 |
| 96 | OK | 0.031 | 10989568 | 84 | 84 |
| 97 | OK | 0.031 | 10944512 | 84 | 84 |
| 98 | OK | 0.031 | 10915840 | 84 | 84 |
| 99 | OK | 0.031 | 10936320 | 84 | 84 |
| 100 | OK | 0.031 | 10907648 | 84 | 84 |
| 101 | OK | 0.046 | 10924032 | 84 | 84 |
| 102 | OK | 0.062 | 10985472 | 84 | 84 |
| 103 | OK | 0.031 | 10964992 | 84 | 84 |
| 104 | OK | 0.046 | 10932224 | 84 | 84 |
| 105 | OK | 0.031 | 10964992 | 84 | 84 |
| 106 | OK | 0.031 | 10907648 | 84 | 84 |
| 107 | OK | 0.031 | 10907648 | 84 | 84 |
| 108 | OK | 0.031 | 10936320 | 84 | 84 |
| 109 | OK | 0.046 | 10899456 | 84 | 84 |
| 110 | OK | 0.031 | 10915840 | 84 | 84 |
| 111 | OK | 0.031 | 10919936 | 84 | 84 |
| 112 | OK | 0.046 | 10919936 | 84 | 84 |
| 113 | OK | 0.046 | 10948608 | 84 | 84 |
| 114 | OK | 0.031 | 10969088 | 84 | 84 |
| 115 | OK | 0.015 | 10911744 | 84 | 84 |
| 116 | OK | 0.015 | 10956800 | 84 | 84 |
| 117 | OK | 0.031 | 10887168 | 84 | 84 |
| 118 | OK | 0.031 | 10891264 | 84 | 84 |
| 119 | OK | 0.046 | 10928128 | 84 | 84 |
| 120 | OK | 0.031 | 10907648 | 84 | 84 |
| 121 | OK | 0.031 | 10964992 | 84 | 84 |
| 122 | OK | 0.046 | 10911744 | 84 | 84 |
| 123 | OK | 0.031 | 10964992 | 84 | 84 |
| 124 | OK | 0.046 | 10952704 | 84 | 84 |
| 125 | OK | 0.031 | 10977280 | 84 | 84 |
| 126 | OK | 0.031 | 10915840 | 84 | 84 |
| 127 | OK | 0.031 | 10985472 | 84 | 84 |
| 128 | OK | 0.031 | 10919936 | 84 | 84 |
| 129 | OK | 0.031 | 10944512 | 84 | 84 |
| 130 | OK | 0.031 | 10948608 | 84 | 84 |
| 131 | OK | 0.031 | 10903552 | 84 | 84 |
| 132 | OK | 0.031 | 10936320 | 93 | 93 |
| 133 | OK | 0.015 | 10993664 | 93 | 93 |
| 134 | OK | 0.031 | 11026432 | 93 | 93 |
| 135 | OK | 0.031 | 10985472 | 93 | 93 |
| 136 | OK | 0.015 | 10964992 | 93 | 93 |
| 137 | OK | 0.015 | 10964992 | 93 | 93 |
| 138 | OK | 0.031 | 10997760 | 93 | 93 |
| 139 | OK | 0.031 | 10895360 | 93 | 93 |
| 140 | OK | 0.015 | 10952704 | 93 | 93 |
| 141 | OK | 0.015 | 10919936 | 93 | 93 |
| 142 | OK | 0.031 | 10944512 | 93 | 93 |
| 143 | OK | 0.031 | 10932224 | 93 | 93 |
| 144 | OK | 0.031 | 10940416 | 93 | 93 |
| 145 | OK | 0.015 | 10977280 | 93 | 93 |
| 146 | OK | 0.031 | 10952704 | 93 | 93 |
| 147 | OK | 0.031 | 10977280 | 93 | 93 |
| 148 | OK | 0.031 | 10899456 | 93 | 93 |
| 149 | OK | 0.031 | 10924032 | 93 | 93 |
| 150 | OK | 0.031 | 10944512 | 93 | 93 |
| 151 | OK | 0.031 | 10887168 | 93 | 93 |
| 152 | OK | 0.015 | 11010048 | 93 | 93 |
| 153 | OK | 0.046 | 10956800 | 93 | 93 |
| 154 | OK | 0.031 | 10948608 | 93 | 93 |
| 155 | OK | 0.031 | 10956800 | 93 | 93 |
| 156 | OK | 0.031 | 11026432 | 93 | 93 |
| 157 | OK | 0.062 | 10940416 | 93 | 93 |
| 158 | OK | 0.031 | 10993664 | 93 | 93 |
| 159 | OK | 0.031 | 10936320 | 93 | 93 |
| 160 | OK | 0.031 | 10956800 | 93 | 93 |
| 161 | OK | 0.031 | 10895360 | 93 | 93 |
| 162 | OK | 0.031 | 10915840 | 93 | 93 |
| 163 | OK | 0.031 | 10940416 | 93 | 93 |
| 164 | OK | 0.031 | 10903552 | 93 | 93 |
| 165 | OK | 0.031 | 10924032 | 93 | 93 |
| 166 | OK | 0.031 | 10964992 | 93 | 93 |
| 167 | OK | 0.031 | 10969088 | 93 | 93 |
| 168 | OK | 0.031 | 10911744 | 93 | 93 |
| 169 | OK | 0.046 | 11014144 | 93 | 93 |
| 170 | OK | 0.031 | 10969088 | 93 | 93 |
| 171 | OK | 0.031 | 10940416 | 93 | 93 |
| 172 | OK | 0.031 | 10915840 | 93 | 93 |
| 173 | OK | 0.031 | 10895360 | 93 | 93 |
| 174 | OK | 0.031 | 10911744 | 93 | 93 |
| 175 | OK | 0.031 | 10977280 | 93 | 93 |
| 176 | OK | 0.046 | 10940416 | 93 | 93 |
| 177 | OK | 0.031 | 10928128 | 93 | 93 |
| 178 | OK | 0.031 | 10948608 | 93 | 93 |
| 179 | OK | 0.046 | 10964992 | 93 | 93 |
| 180 | OK | 0.031 | 10964992 | 93 | 93 |
| 181 | OK | 0.031 | 10956800 | 93 | 93 |
| 182 | OK | 0.046 | 10895360 | 93 | 93 |
| 183 | OK | 0.031 | 10899456 | 93 | 93 |
| 184 | OK | 0.031 | 10919936 | 93 | 93 |
| 185 | OK | 0.031 | 10928128 | 93 | 93 |
| 186 | OK | 0.031 | 10952704 | 93 | 93 |
| 187 | OK | 0.015 | 10895360 | 93 | 93 |
| 188 | OK | 0.031 | 10993664 | 93 | 93 |
| 189 | OK | 0.031 | 10993664 | 93 | 93 |
| 190 | OK | 0.015 | 10907648 | 93 | 93 |
| 191 | OK | 0.015 | 10993664 | 93 | 93 |
| 192 | OK | 0.031 | 10964992 | 93 | 93 |
| 193 | OK | 0.015 | 10924032 | 93 | 93 |
| 194 | OK | 0.031 | 10903552 | 93 | 93 |
| 195 | OK | 0.046 | 10911744 | 93 | 93 |
| 196 | OK | 0.031 | 10928128 | 93 | 93 |
| 197 | OK | 0.031 | 10903552 | 93 | 93 |
| 198 | OK | 0.031 | 10960896 | 93 | 93 |
| 199 | OK | 0.046 | 10960896 | 93 | 93 |
| 200 | OK | 0.046 | 10948608 | 93 | 93 |
| 201 | OK | 0.031 | 10899456 | 93 | 93 |
| 202 | OK | 0.046 | 11001856 | 93 | 93 |
| 203 | OK | 0.031 | 10952704 | 93 | 93 |
| 204 | OK | 0.031 | 10915840 | 93 | 93 |
| 205 | OK | 0.031 | 10924032 | 93 | 93 |
| 206 | OK | 0.031 | 10903552 | 93 | 93 |
| 207 | OK | 0.031 | 10981376 | 93 | 93 |
| 208 | OK | 0.031 | 10952704 | 93 | 93 |
| 209 | OK | 0.031 | 10956800 | 93 | 93 |
| 210 | OK | 0.015 | 10924032 | 93 | 93 |
| 211 | OK | 0.046 | 10997760 | 93 | 93 |
| 212 | OK | 0.046 | 10932224 | 93 | 93 |
| 213 | OK | 0.031 | 11046912 | 93 | 93 |
| 214 | OK | 0.015 | 10993664 | 93 | 93 |
| 215 | OK | 0.031 | 10956800 | 93 | 93 |
| 216 | OK | 0.046 | 10924032 | 93 | 93 |
| 217 | OK | 0.031 | 10883072 | 93 | 93 |
| 218 | OK | 0.031 | 11014144 | 93 | 93 |
| 219 | OK | 0.031 | 10948608 | 93 | 93 |
| 220 | OK | 0.031 | 10993664 | 93 | 93 |
| 221 | OK | 0.031 | 10911744 | 93 | 93 |
| 222 | OK | 0.015 | 10903552 | 93 | 93 |
| 223 | OK | 0.031 | 10981376 | 93 | 93 |
| 224 | OK | 0.046 | 10977280 | 93 | 93 |
| 225 | OK | 0.031 | 10973184 | 93 | 93 |
| 226 | OK | 0.031 | 10928128 | 93 | 93 |
| 227 | OK | 0.015 | 10915840 | 93 | 93 |
| 228 | OK | 0.031 | 10919936 | 93 | 93 |
| 229 | OK | 0.046 | 10944512 | 93 | 93 |
| 230 | OK | 0.031 | 10936320 | 93 | 93 |
| 231 | OK | 0.031 | 10952704 | 93 | 93 |
| 232 | OK | 0.015 | 10919936 | 93 | 93 |
| 233 | OK | 0.015 | 10952704 | 93 | 93 |
| 234 | OK | 0.031 | 10899456 | 93 | 93 |
| 235 | OK | 0.031 | 10964992 | 93 | 93 |
| 236 | OK | 0.031 | 10948608 | 93 | 93 |
| 237 | OK | 0.015 | 10948608 | 93 | 93 |
| 238 | OK | 0.015 | 10948608 | 93 | 93 |
| 239 | OK | 0.031 | 10952704 | 93 | 93 |
| 240 | OK | 0.046 | 10907648 | 93 | 93 |
| 241 | OK | 0.046 | 10936320 | 93 | 93 |
| 242 | OK | 0.031 | 10948608 | 93 | 93 |
| 243 | OK | 0.031 | 10924032 | 93 | 93 |
| 244 | OK | 0.031 | 10928128 | 93 | 93 |
| 245 | OK | 0.031 | 10915840 | 93 | 93 |
| 246 | OK | 0.015 | 10964992 | 93 | 93 |
| 247 | OK | 0.031 | 10911744 | 93 | 93 |
| 248 | OK | 0.031 | 10928128 | 93 | 93 |
| 249 | OK | 0.031 | 10948608 | 93 | 93 |
| 250 | OK | 0.031 | 10952704 | 93 | 93 |
| 251 | OK | 0.046 | 11014144 | 93 | 93 |
| 252 | OK | 0.015 | 10973184 | 220 | 220 |
| 253 | OK | 0.031 | 11034624 | 1798 | 1798 |
| 254 | OK | 0.031 | 11038720 | 1842 | 1842 |
| 255 | OK | 0.031 | 11390976 | 9606 | 9606 |
| 256 | OK | 0.031 | 11325440 | 9569 | 9569 |
| 257 | OK | 0.046 | 11341824 | 9662 | 9662 |
| 258 | OK | 0.046 | 11366400 | 9777 | 9777 |
| 259 | OK | 0.031 | 12046336 | 37717 | 37717 |
| 260 | OK | 0.046 | 12070912 | 37874 | 37874 |
| 261 | OK | 0.031 | 12038144 | 39268 | 39268 |
| 262 | OK | 0.046 | 12095488 | 39470 | 39470 |
| 263 | OK | 0.062 | 18653184 | 181024 | 181024 |
| 264 | OK | 0.046 | 18624512 | 183405 | 183405 |
| 265 | OK | 0.046 | 18579456 | 180784 | 180784 |
| 266 | OK | 0.078 | 18653184 | 183152 | 183152 |
| 267 | OK | 0.046 | 18612224 | 181246 | 181246 |
| 268 | OK | 0.046 | 18673664 | 180886 | 180886 |
| 269 | OK | 0.250 | 33234944 | 1843051 | 1843051 |
| 270 | OK | 0.234 | 33296384 | 1888721 | 1888721 |
| 271 | OK | 0.250 | 33304576 | 1866794 | 1866794 |
| 272 | OK | 0.234 | 33271808 | 1898864 | 1898864 |
| 273 | OK | 0.234 | 33239040 | 1908693 | 1908693 |
| 274 | OK | 0.250 | 33198080 | 1881384 | 1881384 |
| 275 | OK | 0.437 | 48955392 | 3526463 | 3526463 |
| 276 | OK | 0.453 | 49078272 | 3559824 | 3559824 |
| 277 | OK | 0.421 | 49217536 | 3598289 | 3598289 |
| 278 | OK | 0.453 | 49270784 | 3590402 | 3590402 |
| 279 | OK | 0.437 | 49332224 | 3567894 | 3567894 |
| 280 | OK | 0.468 | 48984064 | 3566660 | 3566660 |
| 281 | OK | 0.437 | 48902144 | 3487414 | 3487414 |
| 282 | OK | 0.500 | 50360320 | 3874088 | 3874088 |
| 283 | OK | 0.484 | 50589696 | 3978208 | 3978208 |
| 284 | OK | 0.484 | 50823168 | 3957801 | 3957801 |
| 285 | OK | 0.484 | 50737152 | 3978349 | 3978349 |
| 286 | OK | 0.484 | 50868224 | 3986416 | 3986416 |
| 287 | OK | 0.468 | 50642944 | 3933437 | 3933437 |
| 288 | OK | 0.484 | 50577408 | 3926735 | 3926735 |

# Задача 3 Вставка в АВЛ-дерево

|  |  |
| --- | --- |
| Имя входного файла: | input.txt |
| Имя выходного файла: | output.txt |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 мегабайт |

Вставка в АВЛ-дерево вершины V с ключом X при условии, что такой вершины в этом дереве нет, осуществляется следующим образом:

* находится вершина W, ребенком которой должна стать вершина V
* вершина V делается ребенком вершины W
* производится подъем от вершины W к корню, при этом, если какая-то из вершин несбалансирована, производится, в зависимости от значения баланса, левый или правый поворот.

Первый этап нуждается в пояснении. Спуск до будущего родителя вершины V осуществляется, начиная от корня, следующим образом:

* Пусть ключ текущей вершины равен Y
* Если X < Y и у текущей вершины есть левый ребенок, переходим к левому ребенку.
* Если X < Y и у текущей вершины нет левого ребенка, то останавливаемся, текущая вершина будет родителем новой вершины.
* Если X > Y и у текущей вершины есть правый ребенок, переходим к правому ребенку.
* Если X > Y и у текущей вершины нет правого ребенка, то останавливаемся, текущая вершина будет родителем новой вершины.

Отдельно рассматривается следующий крайний случай — если до вставки дерево было пустым, то вставка новой вершины осуществляется проще: новая вершина становится корнем дерева.

#### Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число   — число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В (i + 1)-ой строке файла (1 ) находится описание i-ой вершины, состоящее из трех чисел Ki, Li, Ri, разделенных пробелами — ключа в i-ой вершине -, номера левого ребенка i-ой вершины ( i < Li < N или Li = 0 , если левого ребенка нет) и номера правого ребенка i-ой вершины ( I < Ri < N или Ri = 0, если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является корректным АВЛ-деревом.

В последней строке содержится число  — ключ вершины, которую требуется вставить в дерево. Гарантируется, что такой вершины в дереве нет.

#### Формат выходного файла

Выведите в том же формате дерево после осуществления операции вставки. Нумерация вершин может быть произвольной при условии соблюдения формата.

#### Пример

|  |  |
| --- | --- |
| input.txt | output.txt |
| 2 3 0 2 4 0 0 5 | 3 4 2 3 3 0 0 5 0 0 |

## Исходный код к задаче 3

class Lab7\_3

{

public static void Main(string[] args)

{

var app = new Lab7\_3();

app.DoWork(args);

}

private void DoWork(string[] args)

{

using (var sw = new StreamWriter("output.txt"))

{

string[] stdin = File.ReadAllLines("input.txt");

TreeNode<long> root = null;

for (int i = 1; i <= long.Parse(stdin[0]); i++)

root = TreeNode<long>.Insert(root, new TreeNode<long> { Key = long.Parse(stdin[i].Split(' ')[0]) });

for (int i = int.Parse(stdin[0]) + 1; i < stdin.Length; i++)

{

TreeNode<long> node = new TreeNode<long> { Key = long.Parse(stdin[i].Split(' ')[0]) };

root = TreeNode<long>.Insert(root, node);

root = TreeNode<long>.Balance(node);

}

sw.WriteLine(stdin.Length - 1);

TreeNode<long>.PrintTree(sw, root);

}

}

class TreeNode<T> where T : IComparable<T>

{

public T Key { get; set; }

public TreeNode<T> Parent { get; set; }

public TreeNode<T> Left { get; set; }

public TreeNode<T> Right { get; set; }

private long Depth { get; set; }

public long Height { get; private set; }

public static TreeNode<T> Previous(TreeNode<T> node)

{

if (node.Left == null)

return node;

return Maximum(node.Left);

}

public static TreeNode<T> Maximum(TreeNode<T> node)

{

while (node.Right != null)

node = node.Right;

return node;

}

/// <returns>Root of tree after remove</returns>

public static TreeNode<T> Remove(TreeNode<T> item)

{

TreeNode<T> parent = item.Parent;

//Leaf

if (item.Left == null && item.Right == null)

{

if (parent == null)

return null;

if (parent.Left == item)

parent.Left = null;

else

parent.Right = null;

UpdateHeight(parent);

return Balance(parent);

}

//One child

if ((item.Left == null) ^ (item.Right == null))

if (item.Left != null)

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Left;

else

parent.Right = item.Left;

UpdateHeight(parent);

}

item.Left.Parent = parent;

return Balance(item.Left);

}

else

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Right;

else

parent.Right = item.Right;

UpdateHeight(parent);

}

item.Right.Parent = parent;

return Balance(item.Right);

}

//Two child

if ((item.Left != null) && (item.Right != null))

{

TreeNode<T> prev = Previous(item);

Remove(prev);

item.Key = prev.Key;

}

return Balance(item);

}

/// <returns>Root of tree after insert</returns>

public static TreeNode<T> Insert(TreeNode<T> root, TreeNode<T> node)

{

if (root == null)

return node;

TreeNode<T> current = root;

while (true)

{

if (current.Key.CompareTo(node.Key) == 0)

throw new ArgumentException("Not unique key");

if (current.Key.CompareTo(node.Key) < 0)

{

if (current.Right != null)

current = current.Right;

else

{

current.Right = node;

node.Parent = current;

UpdateHeight(node);

return root;

//return Balance(node);

}

}

else

{

if (current.Left != null)

current = current.Left;

else

{

current.Left = node;

node.Parent = current;

UpdateHeight(node);

return root;

//return Balance(node);

}

}

}

}

private static void UpdateHeight(TreeNode<T> node)

{

while (node != null)

{

long rH = node.Right != null ? node.Right.Height : -1;

long lH = node.Left != null ? node.Left.Height : -1;

long currentH = node.Height;

if (rH > lH)

node.Height = rH + 1;

else

node.Height = lH + 1;

node = node.Parent;

}

}

/// <returns>Root of tree after balance</returns>

public static TreeNode<T> Balance(TreeNode<T> leaf)

{

TreeNode<T> current = leaf;

while (current != null)

{

long balance = GetBalance(current);

if (balance > 1)

{

if (GetBalance(current.Right) == -1)

current = BigLeftTurn(current);

else

current = SmallLeftTurn(current);

}

if (balance < -1)

{

if (GetBalance(current.Left) == 1)

current = BigRightTurn(current);

else

current = SmallRightTurn(current);

}

if (current.Parent == null)

return current;

else

current = current.Parent;

}

return current;

}

public static void PrintTree(StreamWriter sw, TreeNode<T> root)

{

if (root == null)

return;

Queue<TreeNode<T>> bfsQueue = new Queue<TreeNode<T>>();

long counter = 1;

bfsQueue.Enqueue(root);

while (bfsQueue.Count != 0)

{

TreeNode<T> current = bfsQueue.Dequeue();

sw.Write(current.Key);

if (current.Left != null)

{

bfsQueue.Enqueue(current.Left);

sw.Write(" " + ++counter);

}

else

sw.Write(" " + 0);

if (current.Right != null)

{

bfsQueue.Enqueue(current.Right);

sw.WriteLine(" " + ++counter);

}

else

sw.WriteLine(" " + 0);

}

}

public static long GetBalance(TreeNode<T> tree)

{

if (tree == null)

return 0;

if (tree.Left != null && tree.Right != null)

return tree.Right.Height - tree.Left.Height;

if (tree.Left == null && tree.Right != null)

return tree.Right.Height + 1;

if (tree.Left != null && tree.Right == null)

return -1 - tree.Left.Height;

else

return 0;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallLeftTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Left;

TreeNode<T> y = root.Right.Left;

TreeNode<T> z = root.Right.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = root;

if (z != null)

z.Parent = child;

//Childs

root.Left = x;

root.Right = y;

child.Left = root;

child.Right = z;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (xH > yH)

root.Height = xH + 1;

else

root.Height = yH + 1;

if (root.Height > zH)

child.Height = root.Height + 1;

else

child.Height = zH + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallRightTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Right;

TreeNode<T> y = root.Left.Left;

TreeNode<T> z = root.Left.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = child;

if (z != null)

z.Parent = root;

//Childs

root.Left = z;

root.Right = x;

child.Left = y;

child.Right = root;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (zH > xH)

root.Height = zH + 1;

else

root.Height = xH + 1;

if (y.Height > root.Height)

child.Height = yH + 1;

else

child.Height = root.Height + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigRightTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Left;

TreeNode<T> c = root.Left.Right;

TreeNode<T> z = b.Left;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = root;

if (x != null)

x.Parent = b;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = b;

c.Right = root;

b.Left = z;

b.Right = x;

root.Left = y;

root.Right = w;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (zH > xH)

b.Height = zH + 1;

else

b.Height = xH + 1;

if (yH > wH)

root.Height = yH + 1;

else

root.Height = wH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigLeftTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Right;

TreeNode<T> c = root.Right.Left;

TreeNode<T> z = b.Right;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = b;

if (x != null)

x.Parent = root;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = root;

c.Right = b;

b.Left = y;

b.Right = z;

root.Left = w;

root.Right = x;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (wH > xH)

root.Height = wH + 1;

else

root.Height = xH + 1;

if (yH > zH)

b.Height = yH + 1;

else

b.Height = zH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

}

}

## Бенчмарк к задаче 3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ теста** | **Результат** | **Время, с** | **Память** | **Размер входного файла** | **Размер выходного файла** |
| Max |  | 0.546 | 50647040 | 4011957 | 4011966 |
| 1 | OK | 0.031 | 11149312 | 20 | 24 |
| 2 | OK | 0.046 | 11264000 | 6 | 10 |
| 3 | OK | 0.031 | 11210752 | 14 | 18 |
| 4 | OK | 0.031 | 11120640 | 13 | 17 |
| 5 | OK | 0.046 | 11210752 | 21 | 25 |
| 6 | OK | 0.031 | 11173888 | 20 | 24 |
| 7 | OK | 0.031 | 11145216 | 20 | 24 |
| 8 | OK | 0.031 | 11120640 | 21 | 25 |
| 9 | OK | 0.046 | 11145216 | 20 | 24 |
| 10 | OK | 0.062 | 11128832 | 20 | 24 |
| 11 | OK | 0.046 | 11116544 | 28 | 32 |
| 12 | OK | 0.062 | 10948608 | 27 | 31 |
| 13 | OK | 0.031 | 10936320 | 27 | 31 |
| 14 | OK | 0.031 | 10989568 | 27 | 31 |
| 15 | OK | 0.015 | 10948608 | 35 | 39 |
| 16 | OK | 0.031 | 10952704 | 34 | 38 |
| 17 | OK | 0.031 | 10952704 | 34 | 38 |
| 18 | OK | 0.031 | 10973184 | 34 | 38 |
| 19 | OK | 0.031 | 10936320 | 34 | 38 |
| 20 | OK | 0.031 | 10915840 | 35 | 39 |
| 21 | OK | 0.031 | 10924032 | 34 | 38 |
| 22 | OK | 0.031 | 10993664 | 34 | 38 |
| 23 | OK | 0.031 | 10948608 | 34 | 38 |
| 24 | OK | 0.015 | 10928128 | 34 | 38 |
| 25 | OK | 0.031 | 10993664 | 35 | 39 |
| 26 | OK | 0.031 | 10964992 | 34 | 38 |
| 27 | OK | 0.031 | 10969088 | 34 | 38 |
| 28 | OK | 0.031 | 10960896 | 34 | 38 |
| 29 | OK | 0.046 | 10907648 | 34 | 38 |
| 30 | OK | 0.031 | 10964992 | 35 | 39 |
| 31 | OK | 0.031 | 10948608 | 34 | 38 |
| 32 | OK | 0.031 | 10899456 | 34 | 38 |
| 33 | OK | 0.031 | 10977280 | 34 | 38 |
| 34 | OK | 0.015 | 10960896 | 34 | 38 |
| 35 | OK | 0.031 | 10928128 | 42 | 46 |
| 36 | OK | 0.062 | 11001856 | 41 | 45 |
| 37 | OK | 0.015 | 10919936 | 41 | 45 |
| 38 | OK | 0.031 | 10997760 | 41 | 45 |
| 39 | OK | 0.031 | 10919936 | 41 | 45 |
| 40 | OK | 0.031 | 10919936 | 41 | 45 |
| 41 | OK | 0.031 | 11026432 | 42 | 46 |
| 42 | OK | 0.046 | 10936320 | 41 | 45 |
| 43 | OK | 0.031 | 10903552 | 41 | 45 |
| 44 | OK | 0.031 | 10981376 | 41 | 45 |
| 45 | OK | 0.031 | 10973184 | 41 | 45 |
| 46 | OK | 0.031 | 10911744 | 41 | 45 |
| 47 | OK | 0.031 | 11026432 | 42 | 46 |
| 48 | OK | 0.031 | 10940416 | 41 | 45 |
| 49 | OK | 0.031 | 10964992 | 41 | 45 |
| 50 | OK | 0.031 | 10936320 | 41 | 45 |
| 51 | OK | 0.031 | 10924032 | 41 | 45 |
| 52 | OK | 0.046 | 10989568 | 41 | 45 |
| 53 | OK | 0.031 | 10952704 | 42 | 46 |
| 54 | OK | 0.031 | 10960896 | 41 | 45 |
| 55 | OK | 0.015 | 10928128 | 41 | 45 |
| 56 | OK | 0.062 | 10932224 | 41 | 45 |
| 57 | OK | 0.031 | 10924032 | 41 | 45 |
| 58 | OK | 0.031 | 10928128 | 41 | 45 |
| 59 | OK | 0.031 | 10981376 | 42 | 46 |
| 60 | OK | 0.031 | 10956800 | 41 | 45 |
| 61 | OK | 0.031 | 10928128 | 41 | 45 |
| 62 | OK | 0.031 | 10932224 | 41 | 45 |
| 63 | OK | 0.031 | 10948608 | 41 | 45 |
| 64 | OK | 0.031 | 10915840 | 41 | 45 |
| 65 | OK | 0.046 | 10989568 | 42 | 46 |
| 66 | OK | 0.031 | 10924032 | 41 | 45 |
| 67 | OK | 0.031 | 10960896 | 41 | 45 |
| 68 | OK | 0.031 | 10956800 | 41 | 45 |
| 69 | OK | 0.031 | 10993664 | 41 | 45 |
| 70 | OK | 0.046 | 10956800 | 41 | 45 |
| 71 | OK | 0.031 | 10960896 | 50 | 54 |
| 72 | OK | 0.046 | 10928128 | 49 | 53 |
| 73 | OK | 0.031 | 10911744 | 49 | 53 |
| 74 | OK | 0.046 | 10932224 | 49 | 53 |
| 75 | OK | 0.031 | 10989568 | 49 | 53 |
| 76 | OK | 0.031 | 11042816 | 49 | 53 |
| 77 | OK | 0.031 | 10944512 | 50 | 54 |
| 78 | OK | 0.062 | 10911744 | 50 | 54 |
| 79 | OK | 0.046 | 10989568 | 49 | 53 |
| 80 | OK | 0.046 | 10993664 | 49 | 53 |
| 81 | OK | 0.031 | 10956800 | 49 | 53 |
| 82 | OK | 0.031 | 10907648 | 49 | 53 |
| 83 | OK | 0.031 | 10895360 | 49 | 53 |
| 84 | OK | 0.015 | 10944512 | 50 | 54 |
| 85 | OK | 0.031 | 10924032 | 50 | 54 |
| 86 | OK | 0.015 | 10973184 | 49 | 53 |
| 87 | OK | 0.031 | 10903552 | 49 | 53 |
| 88 | OK | 0.031 | 10915840 | 49 | 53 |
| 89 | OK | 0.031 | 10960896 | 49 | 53 |
| 90 | OK | 0.031 | 10948608 | 49 | 53 |
| 91 | OK | 0.031 | 10985472 | 50 | 54 |
| 92 | OK | 0.031 | 10936320 | 50 | 54 |
| 93 | OK | 0.031 | 10924032 | 49 | 53 |
| 94 | OK | 0.078 | 10919936 | 49 | 53 |
| 95 | OK | 0.046 | 10981376 | 49 | 53 |
| 96 | OK | 0.046 | 10924032 | 49 | 53 |
| 97 | OK | 0.031 | 10924032 | 49 | 53 |
| 98 | OK | 0.031 | 10989568 | 50 | 54 |
| 99 | OK | 0.031 | 10932224 | 58 | 62 |
| 100 | OK | 0.031 | 11018240 | 57 | 61 |
| 101 | OK | 0.031 | 10973184 | 57 | 61 |
| 102 | OK | 0.015 | 10960896 | 57 | 61 |
| 103 | OK | 0.046 | 10956800 | 57 | 61 |
| 104 | OK | 0.031 | 10960896 | 57 | 61 |
| 105 | OK | 0.031 | 10907648 | 58 | 62 |
| 106 | OK | 0.046 | 10903552 | 58 | 62 |
| 107 | OK | 0.031 | 10956800 | 58 | 62 |
| 108 | OK | 0.031 | 10932224 | 57 | 61 |
| 109 | OK | 0.031 | 10969088 | 57 | 61 |
| 110 | OK | 0.031 | 10911744 | 57 | 61 |
| 111 | OK | 0.031 | 10997760 | 57 | 61 |
| 112 | OK | 0.046 | 10936320 | 57 | 61 |
| 113 | OK | 0.031 | 10969088 | 58 | 62 |
| 114 | OK | 0.031 | 10915840 | 58 | 62 |
| 115 | OK | 0.046 | 10944512 | 58 | 62 |
| 116 | OK | 0.031 | 10940416 | 57 | 61 |
| 117 | OK | 0.031 | 10915840 | 57 | 61 |
| 118 | OK | 0.031 | 10969088 | 57 | 61 |
| 119 | OK | 0.031 | 10944512 | 57 | 61 |
| 120 | OK | 0.031 | 10960896 | 57 | 61 |
| 121 | OK | 0.031 | 10924032 | 58 | 62 |
| 122 | OK | 0.031 | 10985472 | 58 | 62 |
| 123 | OK | 0.031 | 10997760 | 58 | 62 |
| 124 | OK | 0.031 | 10915840 | 57 | 61 |
| 125 | OK | 0.031 | 10903552 | 57 | 61 |
| 126 | OK | 0.015 | 10952704 | 57 | 61 |
| 127 | OK | 0.031 | 10936320 | 57 | 61 |
| 128 | OK | 0.015 | 10924032 | 57 | 61 |
| 129 | OK | 0.031 | 10915840 | 58 | 62 |
| 130 | OK | 0.031 | 10907648 | 58 | 62 |
| 131 | OK | 0.031 | 10948608 | 58 | 62 |
| 132 | OK | 0.015 | 10969088 | 57 | 61 |
| 133 | OK | 0.031 | 10919936 | 57 | 61 |
| 134 | OK | 0.015 | 11014144 | 57 | 61 |
| 135 | OK | 0.031 | 10952704 | 57 | 61 |
| 136 | OK | 0.031 | 10907648 | 57 | 61 |
| 137 | OK | 0.093 | 10948608 | 58 | 62 |
| 138 | OK | 0.031 | 10952704 | 58 | 62 |
| 139 | OK | 0.046 | 10956800 | 58 | 62 |
| 140 | OK | 0.031 | 10928128 | 57 | 61 |
| 141 | OK | 0.031 | 10907648 | 57 | 61 |
| 142 | OK | 0.031 | 11005952 | 57 | 61 |
| 143 | OK | 0.031 | 11079680 | 57 | 61 |
| 144 | OK | 0.031 | 10919936 | 57 | 61 |
| 145 | OK | 0.015 | 11001856 | 58 | 62 |
| 146 | OK | 0.031 | 10964992 | 58 | 62 |
| 147 | OK | 0.015 | 10928128 | 58 | 62 |
| 148 | OK | 0.031 | 10981376 | 57 | 61 |
| 149 | OK | 0.015 | 10981376 | 57 | 61 |
| 150 | OK | 0.031 | 10964992 | 57 | 61 |
| 151 | OK | 0.015 | 10964992 | 57 | 61 |
| 152 | OK | 0.046 | 10907648 | 57 | 61 |
| 153 | OK | 0.031 | 10993664 | 58 | 62 |
| 154 | OK | 0.031 | 10956800 | 58 | 62 |
| 155 | OK | 0.046 | 10911744 | 58 | 62 |
| 156 | OK | 0.031 | 11030528 | 57 | 61 |
| 157 | OK | 0.031 | 10944512 | 57 | 61 |
| 158 | OK | 0.046 | 10944512 | 57 | 61 |
| 159 | OK | 0.031 | 10989568 | 57 | 61 |
| 160 | OK | 0.015 | 10924032 | 57 | 61 |
| 161 | OK | 0.031 | 11014144 | 58 | 62 |
| 162 | OK | 0.031 | 10932224 | 58 | 62 |
| 163 | OK | 0.031 | 10936320 | 58 | 62 |
| 164 | OK | 0.078 | 11026432 | 57 | 61 |
| 165 | OK | 0.031 | 10960896 | 57 | 61 |
| 166 | OK | 0.031 | 10924032 | 57 | 61 |
| 167 | OK | 0.031 | 10944512 | 57 | 61 |
| 168 | OK | 0.015 | 10960896 | 57 | 61 |
| 169 | OK | 0.031 | 11026432 | 58 | 62 |
| 170 | OK | 0.031 | 10899456 | 58 | 62 |
| 171 | OK | 0.031 | 10985472 | 58 | 62 |
| 172 | OK | 0.031 | 10948608 | 57 | 61 |
| 173 | OK | 0.031 | 10952704 | 57 | 61 |
| 174 | OK | 0.031 | 10936320 | 57 | 61 |
| 175 | OK | 0.031 | 10981376 | 57 | 61 |
| 176 | OK | 0.015 | 10956800 | 57 | 61 |
| 177 | OK | 0.031 | 10944512 | 58 | 62 |
| 178 | OK | 0.031 | 10895360 | 58 | 62 |
| 179 | OK | 0.031 | 10940416 | 58 | 62 |
| 180 | OK | 0.046 | 10993664 | 57 | 61 |
| 181 | OK | 0.031 | 10919936 | 57 | 61 |
| 182 | OK | 0.031 | 10981376 | 57 | 61 |
| 183 | OK | 0.031 | 10981376 | 57 | 61 |
| 184 | OK | 0.031 | 10973184 | 57 | 61 |
| 185 | OK | 0.031 | 10977280 | 58 | 62 |
| 186 | OK | 0.031 | 10969088 | 58 | 62 |
| 187 | OK | 0.031 | 10989568 | 58 | 62 |
| 188 | OK | 0.031 | 10932224 | 57 | 61 |
| 189 | OK | 0.015 | 10936320 | 57 | 61 |
| 190 | OK | 0.015 | 10936320 | 57 | 61 |
| 191 | OK | 0.031 | 10952704 | 57 | 61 |
| 192 | OK | 0.015 | 11067392 | 57 | 61 |
| 193 | OK | 0.046 | 10956800 | 58 | 62 |
| 194 | OK | 0.031 | 11026432 | 58 | 62 |
| 195 | OK | 0.031 | 10932224 | 58 | 62 |
| 196 | OK | 0.031 | 10932224 | 57 | 61 |
| 197 | OK | 0.015 | 10989568 | 57 | 61 |
| 198 | OK | 0.046 | 10964992 | 57 | 61 |
| 199 | OK | 0.015 | 10936320 | 57 | 61 |
| 200 | OK | 0.015 | 10952704 | 57 | 61 |
| 201 | OK | 0.015 | 10915840 | 58 | 62 |
| 202 | OK | 0.031 | 11001856 | 58 | 62 |
| 203 | OK | 0.031 | 10993664 | 58 | 62 |
| 204 | OK | 0.031 | 10911744 | 57 | 61 |
| 205 | OK | 0.031 | 10944512 | 57 | 61 |
| 206 | OK | 0.031 | 10977280 | 57 | 61 |
| 207 | OK | 0.062 | 10915840 | 57 | 61 |
| 208 | OK | 0.031 | 10960896 | 57 | 61 |
| 209 | OK | 0.015 | 10997760 | 58 | 62 |
| 210 | OK | 0.015 | 10932224 | 58 | 62 |
| 211 | OK | 0.046 | 10924032 | 58 | 62 |
| 212 | OK | 0.031 | 10907648 | 57 | 61 |
| 213 | OK | 0.046 | 10981376 | 57 | 61 |
| 214 | OK | 0.031 | 10944512 | 57 | 61 |
| 215 | OK | 0.031 | 10952704 | 57 | 61 |
| 216 | OK | 0.031 | 10944512 | 57 | 61 |
| 217 | OK | 0.031 | 10940416 | 58 | 62 |
| 218 | OK | 0.031 | 10907648 | 58 | 62 |
| 219 | OK | 0.031 | 10973184 | 58 | 62 |
| 220 | OK | 0.031 | 10969088 | 57 | 61 |
| 221 | OK | 0.031 | 10989568 | 57 | 61 |
| 222 | OK | 0.031 | 10915840 | 57 | 61 |
| 223 | OK | 0.031 | 10911744 | 57 | 61 |
| 224 | OK | 0.015 | 10993664 | 57 | 61 |
| 225 | OK | 0.031 | 10940416 | 58 | 62 |
| 226 | OK | 0.046 | 10940416 | 58 | 62 |
| 227 | OK | 0.046 | 10919936 | 58 | 62 |
| 228 | OK | 0.031 | 10960896 | 57 | 61 |
| 229 | OK | 0.031 | 10977280 | 57 | 61 |
| 230 | OK | 0.031 | 10960896 | 57 | 61 |
| 231 | OK | 0.031 | 10940416 | 57 | 61 |
| 232 | OK | 0.031 | 10989568 | 57 | 61 |
| 233 | OK | 0.031 | 10973184 | 58 | 62 |
| 234 | OK | 0.031 | 10932224 | 58 | 62 |
| 235 | OK | 0.031 | 11001856 | 240 | 245 |
| 236 | OK | 0.031 | 10944512 | 243 | 248 |
| 237 | OK | 0.031 | 11096064 | 1835 | 1841 |
| 238 | OK | 0.031 | 11055104 | 1815 | 1821 |
| 239 | OK | 0.031 | 11018240 | 1834 | 1840 |
| 240 | OK | 0.031 | 11403264 | 9951 | 9957 |
| 241 | OK | 0.015 | 11354112 | 9831 | 9837 |
| 242 | OK | 0.046 | 11382784 | 9854 | 9860 |
| 243 | OK | 0.031 | 12140544 | 38301 | 38308 |
| 244 | OK | 0.031 | 12083200 | 37664 | 37671 |
| 245 | OK | 0.046 | 12083200 | 39108 | 39115 |
| 246 | OK | 0.046 | 12136448 | 39190 | 39197 |
| 247 | OK | 0.046 | 18604032 | 183695 | 183703 |
| 248 | OK | 0.062 | 18657280 | 184258 | 184266 |
| 249 | OK | 0.046 | 18612224 | 185065 | 185073 |
| 250 | OK | 0.062 | 18616320 | 185428 | 185436 |
| 251 | OK | 0.046 | 18702336 | 185741 | 185749 |
| 252 | OK | 0.250 | 33198080 | 1900094 | 1900102 |
| 253 | OK | 0.265 | 33271808 | 1860225 | 1860233 |
| 254 | OK | 0.265 | 33341440 | 1899455 | 1899463 |
| 255 | OK | 0.265 | 33325056 | 1861088 | 1861096 |
| 256 | OK | 0.265 | 33202176 | 1942127 | 1942135 |
| 257 | OK | 0.265 | 33370112 | 1930200 | 1930208 |
| 258 | OK | 0.265 | 33304576 | 1861244 | 1861252 |
| 259 | OK | 0.468 | 48869376 | 3510448 | 3510457 |
| 260 | OK | 0.453 | 49311744 | 3650901 | 3650910 |
| 261 | OK | 0.437 | 49225728 | 3552374 | 3552383 |
| 262 | OK | 0.453 | 48939008 | 3435983 | 3435992 |
| 263 | OK | 0.468 | 49324032 | 3562689 | 3562698 |
| 264 | OK | 0.453 | 49086464 | 3521159 | 3521168 |
| 265 | OK | 0.453 | 49156096 | 3539149 | 3539158 |
| 266 | OK | 0.531 | 50618368 | 3985264 | 3985273 |
| 267 | OK | 0.500 | 50536448 | 3866892 | 3866901 |
| 268 | OK | 0.500 | 50647040 | 3942753 | 3942762 |
| 269 | OK | 0.484 | 50626560 | 3824263 | 3824272 |
| 270 | OK | 0.500 | 50585600 | 4011957 | 4011966 |
| 271 | OK | 0.531 | 50511872 | 3955420 | 3955429 |
| 272 | OK | 0.546 | 50606080 | 3946583 | 3946592 |
| 273 | OK | 0.515 | 50638848 | 3891536 | 3891545 |

# Задача 4 Удаление из АВЛ-дерева

|  |  |
| --- | --- |
| Имя входного файла: | input.txt |
| Имя выходного файла: | output.txt |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 мегабайт |

Удаление из АВЛ-дерева вершины с ключом X, при условии ее наличия, осуществляется следующим образом:

* путем спуска от корня и проверки ключей находится V — удаляемая вершина;
* если вершина  V — лист (то есть, у нее нет детей)
* удаляем вершину;
* поднимаемся к корню, начиная с бывшего родителя вершины V, при этом если встречается несбалансированная вершина, то производим поворот.
* если у вершины V не существует левого ребенка:
  + следовательно, баланс вершины равен единице и ее правый ребенок — лист;
  + заменяем вершину V ее правым ребенком;
  + поднимаемся к корню, производя, где необходимо, балансировку.

 иначе:

 находим R — самую правую вершину в левом поддереве;

 переносим ключ вершины К в вершину V ;

 удаляем вершину R (у нее нет правого ребенка, поэтому она либо лист, либо имеет левого ребенка, являющегося листом);

 поднимаемся к корню, начиная с бывшего родителя вершины R, производя балансировку.

Исключением является случай, когда производится удаление из дерева, состоящего из одной вершины — корня. Результатом удаления в этом случае будет пустое дерево.

Указанный алгоритм не является единственно возможным, но мы просим Вас реализовать именно его, так как тестирующая система проверяет точное равенство получающихся деревьев.

#### Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число   — число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В (i + 1)-ой строке файла (1 ) находится описание i-ой вершины, состоящее из трех чисел Ki, Li, Ri, разделенных пробелами — ключа в i-ой вершине -, номера левого ребенка i-ой вершины ( i < Li < N или Li = 0 , если левого ребенка нет) и номера правого ребенка i-ой вершины ( I < Ri < N или Ri = 0, если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является корректным АВЛ-деревом.

В последней строке содержится число  — ключ вершины, которую требуется вставить в дерево. Гарантируется, что такой вершины в дереве нет.

#### Формат выходного файла

Выведите в том же формате дерево после осуществления операции удаления. Нумерация вершин может быть произвольной при условии соблюдения формата.

#### Пример

|  |  |
| --- | --- |
| input.txt | output.txt |
| 3 4 2 3 3 0 0 5 0 0 4 | 2 3 0 2 5 0 0 |

## Исходный код к задаче 4

class Lab7\_4

{

public static void Main(string[] args)

{

var app = new Lab7\_4();

app.DoWork(args);

}

private void DoWork(string[] args)

{

using (var sw = new StreamWriter("output.txt"))

{

string[] stdin = File.ReadAllLines("input.txt");

int n = int.Parse(stdin[0]);

TreeNode<long> root = null;

for (int i = 1; i <= n; i++)

root = TreeNode<long>.Insert(root, new TreeNode<long> { Key = long.Parse(stdin[i].Split(' ')[0]) });

for (int i = n + 1; i < stdin.Length; i++)

{

TreeNode<long> node = TreeNode<long>.Search(root, long.Parse(stdin[i]));

if (node != null)

root = TreeNode<long>.Remove(node);

}

sw.WriteLine(n - (stdin.Length - 1 - n));

TreeNode<long>.PrintTree(sw, root);

}

}

class TreeNode<T> where T : IComparable<T>

{

public T Key { get; set; }

public TreeNode<T> Parent { get; set; }

public TreeNode<T> Left { get; set; }

public TreeNode<T> Right { get; set; }

private long Depth { get; set; }

public long Height { get; private set; }

public static TreeNode<T> Search(TreeNode<T> root, T key)

{

while (root != null && root.Key.CompareTo(key) != 0)

if (root.Key.CompareTo(key) > 0)

root = root.Left;

else

root = root.Right;

return root;

}

public static TreeNode<T> Previous(TreeNode<T> node)

{

if (node.Left == null)

return node;

return Maximum(node.Left);

}

public static TreeNode<T> Maximum(TreeNode<T> node)

{

while (node.Right != null)

node = node.Right;

return node;

}

/// <returns>Root of tree after remove</returns>

public static TreeNode<T> Remove(TreeNode<T> item)

{

TreeNode<T> parent = item.Parent;

//Leaf

if (item.Left == null && item.Right == null)

{

if (parent == null)

return null;

if (parent.Left == item)

parent.Left = null;

else

parent.Right = null;

UpdateHeight(parent);

return Balance(parent);

}

//One child

if ((item.Left == null) ^ (item.Right == null))

if (item.Left != null)

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Left;

else

parent.Right = item.Left;

UpdateHeight(parent);

}

item.Left.Parent = parent;

return Balance(item.Left);

}

else

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Right;

else

parent.Right = item.Right;

UpdateHeight(parent);

}

item.Right.Parent = parent;

return Balance(item.Right);

}

//Two child

if ((item.Left != null) && (item.Right != null))

{

TreeNode<T> prev = Previous(item);

Remove(prev);

item.Key = prev.Key;

}

return Balance(item);

}

/// <returns>Root of tree after insert</returns>

public static TreeNode<T> Insert(TreeNode<T> root, TreeNode<T> node)

{

if (root == null)

return node;

TreeNode<T> current = root;

while (true)

{

if (current.Key.CompareTo(node.Key) == 0)

throw new ArgumentException("Not unique key");

if (current.Key.CompareTo(node.Key) < 0)

{

if (current.Right != null)

current = current.Right;

else

{

current.Right = node;

node.Parent = current;

UpdateHeight(node);

return root;

//return Balance(node);

}

}

else

{

if (current.Left != null)

current = current.Left;

else

{

current.Left = node;

node.Parent = current;

UpdateHeight(node);

return root;

//return Balance(node);

}

}

}

}

private static void UpdateHeight(TreeNode<T> node)

{

while (node != null)

{

long rH = node.Right != null ? node.Right.Height : -1;

long lH = node.Left != null ? node.Left.Height : -1;

long currentH = node.Height;

if (rH > lH)

node.Height = rH + 1;

else

node.Height = lH + 1;

node = node.Parent;

}

}

/// <returns>Root of tree after balance</returns>

public static TreeNode<T> Balance(TreeNode<T> leaf)

{

TreeNode<T> current = leaf;

while (current != null)

{

long balance = GetBalance(current);

if (balance > 1)

{

if (GetBalance(current.Right) == -1)

current = BigLeftTurn(current);

else

current = SmallLeftTurn(current);

}

if (balance < -1)

{

if (GetBalance(current.Left) == 1)

current = BigRightTurn(current);

else

current = SmallRightTurn(current);

}

if (current.Parent == null)

return current;

else

current = current.Parent;

}

return current;

}

public static void PrintTree(StreamWriter sw, TreeNode<T> root)

{

if (root == null)

return;

Queue<TreeNode<T>> bfsQueue = new Queue<TreeNode<T>>();

long counter = 1;

bfsQueue.Enqueue(root);

while (bfsQueue.Count != 0)

{

TreeNode<T> current = bfsQueue.Dequeue();

sw.Write(current.Key);

if (current.Left != null)

{

bfsQueue.Enqueue(current.Left);

sw.Write(" " + ++counter);

}

else

sw.Write(" " + 0);

if (current.Right != null)

{

bfsQueue.Enqueue(current.Right);

sw.WriteLine(" " + ++counter);

}

else

sw.WriteLine(" " + 0);

}

}

public static long GetBalance(TreeNode<T> tree)

{

if (tree == null)

return 0;

if (tree.Left != null && tree.Right != null)

return tree.Right.Height - tree.Left.Height;

if (tree.Left == null && tree.Right != null)

return tree.Right.Height + 1;

if (tree.Left != null && tree.Right == null)

return -1 - tree.Left.Height;

else

return 0;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallLeftTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Left;

TreeNode<T> y = root.Right.Left;

TreeNode<T> z = root.Right.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = root;

if (z != null)

z.Parent = child;

//Childs

root.Left = x;

root.Right = y;

child.Left = root;

child.Right = z;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (xH > yH)

root.Height = xH + 1;

else

root.Height = yH + 1;

if (root.Height > zH)

child.Height = root.Height + 1;

else

child.Height = zH + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallRightTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Right;

TreeNode<T> y = root.Left.Left;

TreeNode<T> z = root.Left.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = child;

if (z != null)

z.Parent = root;

//Childs

root.Left = z;

root.Right = x;

child.Left = y;

child.Right = root;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (zH > xH)

root.Height = zH + 1;

else

root.Height = xH + 1;

if (y.Height > root.Height)

child.Height = yH + 1;

else

child.Height = root.Height + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigRightTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Left;

TreeNode<T> c = root.Left.Right;

TreeNode<T> z = b.Left;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = root;

if (x != null)

x.Parent = b;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = b;

c.Right = root;

b.Left = z;

b.Right = x;

root.Left = y;

root.Right = w;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (zH > xH)

b.Height = zH + 1;

else

b.Height = xH + 1;

if (yH > wH)

root.Height = yH + 1;

else

root.Height = wH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigLeftTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Right;

TreeNode<T> c = root.Right.Left;

TreeNode<T> z = b.Right;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = b;

if (x != null)

x.Parent = root;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = root;

c.Right = b;

b.Left = y;

b.Right = z;

root.Left = w;

root.Right = x;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (wH > xH)

root.Height = wH + 1;

else

root.Height = xH + 1;

if (yH > zH)

b.Height = yH + 1;

else

b.Height = zH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

}

}

## Бенчмарк к задаче 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ теста** | **Результат** | **Время, с** | **Память** | **Размер входного файла** | **Размер выходного файла** |
| Max |  | 0.515 | 50823168 | 4077288 | 4077255 |
| 1 | OK | 0.046 | 11120640 | 27 | 17 |
| 2 | OK | 0.031 | 11096064 | 13 | 3 |
| 3 | OK | 0.031 | 11161600 | 20 | 10 |
| 4 | OK | 0.031 | 11190272 | 20 | 10 |
| 5 | OK | 0.031 | 11161600 | 20 | 10 |
| 6 | OK | 0.031 | 11153408 | 20 | 10 |
| 7 | OK | 0.031 | 11132928 | 27 | 17 |
| 8 | OK | 0.046 | 11153408 | 27 | 17 |
| 9 | OK | 0.031 | 11182080 | 27 | 17 |
| 10 | OK | 0.031 | 11153408 | 34 | 24 |
| 11 | OK | 0.046 | 11145216 | 34 | 24 |
| 12 | OK | 0.031 | 10928128 | 34 | 24 |
| 13 | OK | 0.046 | 10928128 | 34 | 24 |
| 14 | OK | 0.031 | 10948608 | 34 | 24 |
| 15 | OK | 0.046 | 10964992 | 34 | 24 |
| 16 | OK | 0.031 | 10973184 | 34 | 24 |
| 17 | OK | 0.046 | 10940416 | 34 | 24 |
| 18 | OK | 0.031 | 10940416 | 34 | 24 |
| 19 | OK | 0.031 | 10952704 | 34 | 24 |
| 20 | OK | 0.031 | 10981376 | 34 | 24 |
| 21 | OK | 0.031 | 10903552 | 34 | 24 |
| 22 | OK | 0.031 | 10924032 | 34 | 24 |
| 23 | OK | 0.031 | 10956800 | 34 | 24 |
| 24 | OK | 0.031 | 10907648 | 34 | 24 |
| 25 | OK | 0.046 | 10985472 | 34 | 24 |
| 26 | OK | 0.031 | 10940416 | 41 | 31 |
| 27 | OK | 0.031 | 10936320 | 41 | 31 |
| 28 | OK | 0.015 | 10899456 | 41 | 31 |
| 29 | OK | 0.046 | 10915840 | 41 | 31 |
| 30 | OK | 0.046 | 10997760 | 41 | 31 |
| 31 | OK | 0.031 | 10960896 | 41 | 31 |
| 32 | OK | 0.031 | 10924032 | 41 | 31 |
| 33 | OK | 0.031 | 10919936 | 41 | 31 |
| 34 | OK | 0.031 | 10997760 | 41 | 31 |
| 35 | OK | 0.031 | 10977280 | 41 | 31 |
| 36 | OK | 0.031 | 10915840 | 41 | 31 |
| 37 | OK | 0.031 | 10952704 | 41 | 31 |
| 38 | OK | 0.031 | 10948608 | 41 | 31 |
| 39 | OK | 0.031 | 10919936 | 41 | 31 |
| 40 | OK | 0.031 | 10924032 | 41 | 31 |
| 41 | OK | 0.046 | 10964992 | 41 | 31 |
| 42 | OK | 0.031 | 10956800 | 41 | 31 |
| 43 | OK | 0.031 | 10952704 | 41 | 31 |
| 44 | OK | 0.015 | 10891264 | 41 | 31 |
| 45 | OK | 0.046 | 10964992 | 41 | 31 |
| 46 | OK | 0.031 | 10952704 | 41 | 31 |
| 47 | OK | 0.046 | 10907648 | 41 | 31 |
| 48 | OK | 0.031 | 10952704 | 41 | 31 |
| 49 | OK | 0.031 | 10948608 | 41 | 31 |
| 50 | OK | 0.031 | 10940416 | 41 | 31 |
| 51 | OK | 0.046 | 10928128 | 41 | 31 |
| 52 | OK | 0.031 | 11034624 | 41 | 31 |
| 53 | OK | 0.031 | 10919936 | 41 | 31 |
| 54 | OK | 0.031 | 10989568 | 41 | 31 |
| 55 | OK | 0.031 | 10924032 | 41 | 31 |
| 56 | OK | 0.031 | 10940416 | 48 | 38 |
| 57 | OK | 0.031 | 10903552 | 48 | 38 |
| 58 | OK | 0.046 | 10977280 | 48 | 38 |
| 59 | OK | 0.031 | 10993664 | 48 | 38 |
| 60 | OK | 0.031 | 10919936 | 48 | 38 |
| 61 | OK | 0.015 | 10956800 | 48 | 38 |
| 62 | OK | 0.031 | 11018240 | 48 | 38 |
| 63 | OK | 0.031 | 10928128 | 48 | 38 |
| 64 | OK | 0.031 | 10989568 | 48 | 38 |
| 65 | OK | 0.046 | 10899456 | 48 | 38 |
| 66 | OK | 0.031 | 10969088 | 48 | 38 |
| 67 | OK | 0.031 | 10964992 | 48 | 38 |
| 68 | OK | 0.046 | 11010048 | 48 | 38 |
| 69 | OK | 0.031 | 10964992 | 48 | 38 |
| 70 | OK | 0.046 | 10944512 | 48 | 38 |
| 71 | OK | 0.031 | 10936320 | 48 | 38 |
| 72 | OK | 0.031 | 10960896 | 48 | 38 |
| 73 | OK | 0.031 | 10964992 | 48 | 38 |
| 74 | OK | 0.046 | 10932224 | 48 | 38 |
| 75 | OK | 0.031 | 10964992 | 48 | 38 |
| 76 | OK | 0.031 | 10903552 | 48 | 38 |
| 77 | OK | 0.031 | 10891264 | 48 | 38 |
| 78 | OK | 0.031 | 10956800 | 48 | 38 |
| 79 | OK | 0.031 | 10940416 | 48 | 38 |
| 80 | OK | 0.031 | 10989568 | 55 | 45 |
| 81 | OK | 0.031 | 10907648 | 55 | 45 |
| 82 | OK | 0.031 | 10932224 | 55 | 45 |
| 83 | OK | 0.046 | 10956800 | 55 | 45 |
| 84 | OK | 0.031 | 10956800 | 55 | 45 |
| 85 | OK | 0.031 | 10919936 | 55 | 45 |
| 86 | OK | 0.046 | 10952704 | 55 | 45 |
| 87 | OK | 0.031 | 10940416 | 55 | 45 |
| 88 | OK | 0.031 | 10981376 | 55 | 45 |
| 89 | OK | 0.031 | 11042816 | 55 | 45 |
| 90 | OK | 0.031 | 10919936 | 55 | 45 |
| 91 | OK | 0.031 | 10956800 | 55 | 45 |
| 92 | OK | 0.031 | 10964992 | 55 | 45 |
| 93 | OK | 0.062 | 10948608 | 55 | 45 |
| 94 | OK | 0.031 | 10985472 | 55 | 45 |
| 95 | OK | 0.046 | 10919936 | 55 | 45 |
| 96 | OK | 0.062 | 10928128 | 55 | 45 |
| 97 | OK | 0.031 | 10915840 | 55 | 45 |
| 98 | OK | 0.031 | 10907648 | 55 | 45 |
| 99 | OK | 0.031 | 10981376 | 55 | 45 |
| 100 | OK | 0.031 | 10973184 | 55 | 45 |
| 101 | OK | 0.031 | 10932224 | 55 | 45 |
| 102 | OK | 0.046 | 10936320 | 55 | 45 |
| 103 | OK | 0.031 | 10919936 | 55 | 45 |
| 104 | OK | 0.046 | 10964992 | 55 | 45 |
| 105 | OK | 0.031 | 10944512 | 55 | 45 |
| 106 | OK | 0.031 | 10977280 | 55 | 45 |
| 107 | OK | 0.078 | 10948608 | 55 | 45 |
| 108 | OK | 0.031 | 10928128 | 55 | 45 |
| 109 | OK | 0.031 | 10944512 | 55 | 45 |
| 110 | OK | 0.031 | 10989568 | 55 | 45 |
| 111 | OK | 0.031 | 10981376 | 55 | 45 |
| 112 | OK | 0.031 | 10919936 | 55 | 45 |
| 113 | OK | 0.031 | 10973184 | 55 | 45 |
| 114 | OK | 0.031 | 10989568 | 55 | 45 |
| 115 | OK | 0.031 | 10964992 | 55 | 45 |
| 116 | OK | 0.031 | 11005952 | 55 | 45 |
| 117 | OK | 0.031 | 10940416 | 55 | 45 |
| 118 | OK | 0.031 | 10948608 | 55 | 45 |
| 119 | OK | 0.031 | 10928128 | 55 | 45 |
| 120 | OK | 0.031 | 10977280 | 55 | 45 |
| 121 | OK | 0.031 | 10936320 | 55 | 45 |
| 122 | OK | 0.031 | 11005952 | 55 | 45 |
| 123 | OK | 0.031 | 10944512 | 55 | 45 |
| 124 | OK | 0.031 | 10969088 | 55 | 45 |
| 125 | OK | 0.031 | 10981376 | 55 | 45 |
| 126 | OK | 0.031 | 10940416 | 55 | 45 |
| 127 | OK | 0.031 | 10985472 | 55 | 45 |
| 128 | OK | 0.031 | 10964992 | 55 | 45 |
| 129 | OK | 0.031 | 10919936 | 55 | 45 |
| 130 | OK | 0.046 | 10911744 | 55 | 45 |
| 131 | OK | 0.031 | 10969088 | 55 | 45 |
| 132 | OK | 0.046 | 10928128 | 55 | 45 |
| 133 | OK | 0.031 | 10952704 | 55 | 45 |
| 134 | OK | 0.031 | 10919936 | 55 | 45 |
| 135 | OK | 0.031 | 10969088 | 55 | 45 |
| 136 | OK | 0.031 | 10948608 | 55 | 45 |
| 137 | OK | 0.031 | 10944512 | 55 | 45 |
| 138 | OK | 0.031 | 10997760 | 55 | 45 |
| 139 | OK | 0.031 | 10915840 | 55 | 45 |
| 140 | OK | 0.046 | 10911744 | 55 | 45 |
| 141 | OK | 0.031 | 10969088 | 55 | 45 |
| 142 | OK | 0.031 | 10985472 | 55 | 45 |
| 143 | OK | 0.031 | 10964992 | 55 | 45 |
| 144 | OK | 0.031 | 10969088 | 55 | 45 |
| 145 | OK | 0.031 | 10940416 | 55 | 45 |
| 146 | OK | 0.031 | 10964992 | 55 | 45 |
| 147 | OK | 0.015 | 10948608 | 55 | 45 |
| 148 | OK | 0.031 | 10948608 | 55 | 45 |
| 149 | OK | 0.031 | 10952704 | 55 | 45 |
| 150 | OK | 0.031 | 10948608 | 55 | 45 |
| 151 | OK | 0.031 | 10924032 | 55 | 45 |
| 152 | OK | 0.031 | 10928128 | 55 | 45 |
| 153 | OK | 0.031 | 10952704 | 55 | 45 |
| 154 | OK | 0.031 | 10981376 | 55 | 45 |
| 155 | OK | 0.031 | 10936320 | 55 | 45 |
| 156 | OK | 0.031 | 10964992 | 55 | 45 |
| 157 | OK | 0.046 | 10956800 | 55 | 45 |
| 158 | OK | 0.031 | 10940416 | 55 | 45 |
| 159 | OK | 0.046 | 10944512 | 55 | 45 |
| 160 | OK | 0.031 | 11001856 | 55 | 45 |
| 161 | OK | 0.031 | 10936320 | 55 | 45 |
| 162 | OK | 0.031 | 10919936 | 55 | 45 |
| 163 | OK | 0.031 | 10919936 | 55 | 45 |
| 164 | OK | 0.015 | 10936320 | 55 | 45 |
| 165 | OK | 0.031 | 10919936 | 55 | 45 |
| 166 | OK | 0.031 | 10952704 | 55 | 45 |
| 167 | OK | 0.031 | 10969088 | 55 | 45 |
| 168 | OK | 0.031 | 10981376 | 55 | 45 |
| 169 | OK | 0.031 | 10915840 | 55 | 45 |
| 170 | OK | 0.046 | 10944512 | 55 | 45 |
| 171 | OK | 0.015 | 10956800 | 55 | 45 |
| 172 | OK | 0.015 | 10948608 | 55 | 45 |
| 173 | OK | 0.031 | 10895360 | 55 | 45 |
| 174 | OK | 0.031 | 10944512 | 55 | 45 |
| 175 | OK | 0.015 | 10973184 | 55 | 45 |
| 176 | OK | 0.031 | 10932224 | 55 | 45 |
| 177 | OK | 0.031 | 11005952 | 55 | 45 |
| 178 | OK | 0.031 | 10936320 | 55 | 45 |
| 179 | OK | 0.031 | 10924032 | 55 | 45 |
| 180 | OK | 0.015 | 10940416 | 55 | 45 |
| 181 | OK | 0.031 | 10956800 | 55 | 45 |
| 182 | OK | 0.046 | 10964992 | 55 | 45 |
| 183 | OK | 0.031 | 10924032 | 55 | 45 |
| 184 | OK | 0.046 | 10956800 | 55 | 45 |
| 185 | OK | 0.046 | 10907648 | 55 | 45 |
| 186 | OK | 0.031 | 10940416 | 55 | 45 |
| 187 | OK | 0.015 | 10928128 | 55 | 45 |
| 188 | OK | 0.031 | 10956800 | 55 | 45 |
| 189 | OK | 0.046 | 10907648 | 55 | 45 |
| 190 | OK | 0.031 | 10960896 | 55 | 45 |
| 191 | OK | 0.031 | 10919936 | 55 | 45 |
| 192 | OK | 0.046 | 11010048 | 55 | 45 |
| 193 | OK | 0.015 | 10981376 | 55 | 45 |
| 194 | OK | 0.031 | 10936320 | 55 | 45 |
| 195 | OK | 0.046 | 10940416 | 55 | 45 |
| 196 | OK | 0.031 | 10948608 | 55 | 45 |
| 197 | OK | 0.031 | 10960896 | 55 | 45 |
| 198 | OK | 0.031 | 10977280 | 55 | 45 |
| 199 | OK | 0.031 | 11014144 | 239 | 210 |
| 200 | OK | 0.031 | 10960896 | 235 | 208 |
| 201 | OK | 0.031 | 11059200 | 1797 | 1769 |
| 202 | OK | 0.031 | 11083776 | 1809 | 1781 |
| 203 | OK | 0.046 | 11104256 | 1831 | 1803 |
| 204 | OK | 0.031 | 11448320 | 9625 | 9597 |
| 205 | OK | 0.046 | 11354112 | 10026 | 9996 |
| 206 | OK | 0.031 | 11431936 | 9672 | 9642 |
| 207 | OK | 0.031 | 12058624 | 39459 | 39428 |
| 208 | OK | 0.031 | 12099584 | 39672 | 39641 |
| 209 | OK | 0.031 | 12066816 | 38780 | 38749 |
| 210 | OK | 0.046 | 12214272 | 38392 | 38361 |
| 211 | OK | 0.046 | 18624512 | 179425 | 179394 |
| 212 | OK | 0.062 | 18673664 | 182878 | 182845 |
| 213 | OK | 0.062 | 18706432 | 185986 | 185953 |
| 214 | OK | 0.062 | 18636800 | 186275 | 186244 |
| 215 | OK | 0.046 | 18624512 | 179082 | 179051 |
| 216 | OK | 0.250 | 33349632 | 1912349 | 1912319 |
| 217 | OK | 0.250 | 33280000 | 1864033 | 1864003 |
| 218 | OK | 0.265 | 33243136 | 1913940 | 1913908 |
| 219 | OK | 0.250 | 33353728 | 1879988 | 1879958 |
| 220 | OK | 0.234 | 33292288 | 1887512 | 1887482 |
| 221 | OK | 0.234 | 33378304 | 1932558 | 1932526 |
| 222 | OK | 0.250 | 33280000 | 1875036 | 1875006 |
| 223 | OK | 0.437 | 49283072 | 3597394 | 3597361 |
| 224 | OK | 0.437 | 49090560 | 3569973 | 3569940 |
| 225 | OK | 0.437 | 49139712 | 3512224 | 3512193 |
| 226 | OK | 0.437 | 49209344 | 3624329 | 3624296 |
| 227 | OK | 0.421 | 45637632 | 3523352 | 3523321 |
| 228 | OK | 0.453 | 49262592 | 3638077 | 3638044 |
| 229 | OK | 0.453 | 48955392 | 3512844 | 3512813 |
| 230 | OK | 0.468 | 50806784 | 4001320 | 4001287 |
| 231 | OK | 0.468 | 50536448 | 3954282 | 3954249 |
| 232 | OK | 0.484 | 50589696 | 3907814 | 3907783 |
| 233 | OK | 0.484 | 50774016 | 3983014 | 3982983 |
| 234 | OK | 0.484 | 50823168 | 4029895 | 4029862 |
| 235 | OK | 0.515 | 50683904 | 4046188 | 4046157 |
| 236 | OK | 0.484 | 50823168 | 4077288 | 4077255 |
| 237 | OK | 0.500 | 50507776 | 3902092 | 3902061 |

# Задача 5 Упорядоченное множество на АВЛ-дереве

|  |  |
| --- | --- |
| Имя входного файла: | input.txt |
| Имя выходного файла: | output.txt |
| Ограничение по времени: | 2 секунды |
| Ограничение по памяти: | 256 мегабайт |

Если Вы сдали все предыдущие задачи, Вы уже можете написать эффективную реализацию упорядоченного множества на АВЛ-дереве. Сделайте это.

Для проверки того, что множество реализовано именно на АВЛ-дереве, мы просим Вас выводить баланс корня после каждой операции вставки и удаления.

Операции вставки и удаления требуется реализовать точно так же, как это было сделано в предыдущих двух задачах, потому что в ином случае баланс корня может отличаться от требуемого.

#### Формат входного файла

В первой строке файла находится число   — число операций над множеством. Изначально множество пусто. В каждой из последующих N строк файла находится описание операции.

Операции бывают следующих видов:

* A r — вставить число в множество. Если число x там уже содержится, множество изменять не следует.
* D x — удалить число из множества. Если числа x нет в множестве, множество изменять не следует.
* C x — проверить, есть ли число в множестве.

#### Формат выходного файла

Для каждой операции вида C x выведите Y, если число x содержится в множестве, и N, если не содержится.

Для каждой операции вида A x или D x выведите баланс корня дерева после выполнения операции. Если дерево пустое (в нем нет вершин), выведите 0.

Вывод для каждой операции должен содержаться на отдельной строке.

#### Пример

|  |  |
| --- | --- |
| input.txt | output.txt |
| 6 A 3 A 4 A 5 C 4 C 6 D 5 | 0 1 0 Y N -1 |

## Исходный код к задаче 5

class Lab7\_5

{

public static void Main(string[] args)

{

var app = new Lab7\_5();

app.DoWork(args);

}

private void DoWork(string[] args)

{

using (var sw = new StreamWriter("output.txt"))

{

string[] stdin = File.ReadAllLines("input.txt");

TreeNode<long> root = null;

for (int i = 1; i < stdin.Length; i++)

{

string[] temp = stdin[i].Split(' ');

switch (temp[0])

{

case "A":

TreeNode<long> s = TreeNode<long>.Search(root, long.Parse(temp[1]));

if (s == null)

root = TreeNode<long>.Insert(root, new TreeNode<long> { Key = long.Parse(temp[1]) });

sw.WriteLine(TreeNode<long>.GetBalance(root));

break;

case "D":

TreeNode<long> t = TreeNode<long>.Search(root, long.Parse(temp[1]));

if (t != null)

root = TreeNode<long>.Remove(t);

sw.WriteLine(TreeNode<long>.GetBalance(root));

break;

case "C":

TreeNode<long> x = TreeNode<long>.Search(root, long.Parse(temp[1]));

if (x != null)

sw.WriteLine("Y");

else

sw.WriteLine("N");

break;

}

}

}

}

class TreeNode<T> where T : IComparable<T>

{

public T Key { get; set; }

public TreeNode<T> Parent { get; set; }

public TreeNode<T> Left { get; set; }

public TreeNode<T> Right { get; set; }

public long Height { get; set; }

public static TreeNode<T> Next(TreeNode<T> node)

{

if (node.Right == null)

return node;

return Minimum(node.Right);

}

public static TreeNode<T> Previous(TreeNode<T> node)

{

if (node.Left == null)

return node;

return Maximum(node.Left);

}

public static TreeNode<T> Maximum(TreeNode<T> node)

{

while (node.Right != null)

node = node.Right;

return node;

}

public static TreeNode<T> Minimum(TreeNode<T> node)

{

while (node.Left != null)

node = node.Left;

return node;

}

/// <returns>Root of tree after remove</returns>

public static TreeNode<T> Remove(TreeNode<T> item)

{

TreeNode<T> parent = item.Parent;

//Leaf

if (item.Left == null && item.Right == null)

{

if (parent == null)

return null;

if (parent.Left == item)

parent.Left = null;

else

parent.Right = null;

UpdateHeight(parent);

return Balance(parent);

}

//One child

if ((item.Left == null) ^ (item.Right == null))

if (item.Left != null)

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Left;

else

parent.Right = item.Left;

UpdateHeight(parent);

}

item.Left.Parent = parent;

return Balance(item.Left);

}

else

{

if (parent != null)

{

if (parent.Left == item)

parent.Left = item.Right;

else

parent.Right = item.Right;

UpdateHeight(parent);

}

item.Right.Parent = parent;

return Balance(item.Right);

}

//Two child

if ((item.Left != null) && (item.Right != null))

{

TreeNode<T> prev = Previous(item);

Remove(prev);

item.Key = prev.Key;

}

return Balance(item);

}

/// <returns>Root of tree after insert</returns>

public static TreeNode<T> Insert(TreeNode<T> root, TreeNode<T> node)

{

if (root == null)

return node;

TreeNode<T> current = root;

while (true)

{

if (current.Key.CompareTo(node.Key) == 0)

throw new ArgumentException("Not unique key");

if (current.Key.CompareTo(node.Key) < 0)

{

if (current.Right != null)

current = current.Right;

else

{

current.Right = node;

node.Parent = current;

UpdateHeight(node);

return Balance(node);

}

}

else

{

if (current.Left != null)

current = current.Left;

else

{

current.Left = node;

node.Parent = current;

UpdateHeight(node);

return Balance(node);

}

}

}

}

private static void UpdateHeight(TreeNode<T> node)

{

while (node != null)

{

long rH = node.Right != null ? node.Right.Height : -1;

long lH = node.Left != null ? node.Left.Height : -1;

long currentH = node.Height;

if (rH > lH)

node.Height = rH + 1;

else

node.Height = lH + 1;

node = node.Parent;

}

}

public static TreeNode<T> Search(TreeNode<T> root, T key)

{

while (root != null && root.Key.CompareTo(key) != 0)

if (root.Key.CompareTo(key) > 0)

root = root.Left;

else

root = root.Right;

return root;

}

/// <returns>Root of tree after balance</returns>

public static TreeNode<T> Balance(TreeNode<T> leaf)

{

TreeNode<T> current = leaf;

while (current != null)

{

long balance = GetBalance(current);

if (balance > 1)

{

if (GetBalance(current.Right) == -1)

current = BigLeftTurn(current);

else

current = SmallLeftTurn(current);

}

if (balance < -1)

{

if (GetBalance(current.Left) == 1)

current = BigRightTurn(current);

else

current = SmallRightTurn(current);

}

if (current.Parent == null)

return current;

else

current = current.Parent;

}

return current;

}

public static void PrintTree(TreeNode<T> root)

{

if (root == null)

return;

Queue<TreeNode<T>> bfsQueue = new Queue<TreeNode<T>>();

long counter = 1;

bfsQueue.Enqueue(root);

while (bfsQueue.Count != 0)

{

TreeNode<T> current = bfsQueue.Dequeue();

Console.Write(current.Key);

if (current.Left != null)

{

bfsQueue.Enqueue(current.Left);

Console.Write(" " + ++counter);

}

else

Console.Write(" " + 0);

if (current.Right != null)

{

bfsQueue.Enqueue(current.Right);

Console.WriteLine(" " + ++counter);

}

else

Console.WriteLine(" " + 0);

}

}

public static long GetBalance(TreeNode<T> tree)

{

if (tree == null)

return 0;

if (tree.Left != null && tree.Right != null)

return tree.Right.Height - tree.Left.Height;

if (tree.Left == null && tree.Right != null)

return tree.Right.Height + 1;

if (tree.Left != null && tree.Right == null)

return -1 - tree.Left.Height;

else

return 0;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallLeftTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Left;

TreeNode<T> y = root.Right.Left;

TreeNode<T> z = root.Right.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = root;

if (z != null)

z.Parent = child;

//Childs

root.Left = x;

root.Right = y;

child.Left = root;

child.Right = z;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (xH > yH)

root.Height = xH + 1;

else

root.Height = yH + 1;

if (root.Height > zH)

child.Height = root.Height + 1;

else

child.Height = zH + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> SmallRightTurn(TreeNode<T> root)

{

TreeNode<T> child = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> x = root.Right;

TreeNode<T> y = root.Left.Left;

TreeNode<T> z = root.Left.Right;

//Parents

child.Parent = parent;

root.Parent = child;

if (x != null)

x.Parent = root;

if (y != null)

y.Parent = child;

if (z != null)

z.Parent = root;

//Childs

root.Left = z;

root.Right = x;

child.Left = y;

child.Right = root;

if (parent != null)

if (parent.Right == root)

parent.Right = child;

else

parent.Left = child;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

if (zH > xH)

root.Height = zH + 1;

else

root.Height = xH + 1;

if (y.Height > root.Height)

child.Height = yH + 1;

else

child.Height = root.Height + 1;

UpdateHeight(child);

return child;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigRightTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Right;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Left;

TreeNode<T> c = root.Left.Right;

TreeNode<T> z = b.Left;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = root;

if (x != null)

x.Parent = b;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = b;

c.Right = root;

b.Left = z;

b.Right = x;

root.Left = y;

root.Right = w;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (zH > xH)

b.Height = zH + 1;

else

b.Height = xH + 1;

if (yH > wH)

root.Height = yH + 1;

else

root.Height = wH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

/// <returns>Root of tree after turn</returns>

public static TreeNode<T> BigLeftTurn(TreeNode<T> root)

{

TreeNode<T> w = root.Left;

TreeNode<T> parent = root.Parent;

TreeNode<T> b = root.Right;

TreeNode<T> c = root.Right.Left;

TreeNode<T> z = b.Right;

TreeNode<T> x = c.Left;

TreeNode<T> y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

root.Parent = c;

if (w != null)

w.Parent = root;

if (z != null)

z.Parent = b;

if (y != null)

y.Parent = b;

if (x != null)

x.Parent = root;

//Childs

if (parent != null)

if (parent.Right == root)

parent.Right = c;

else

parent.Left = c;

c.Left = root;

c.Right = b;

b.Left = y;

b.Right = z;

root.Left = w;

root.Right = x;

//Heights

long xH = x != null ? x.Height : -1;

long yH = y != null ? y.Height : -1;

long zH = z != null ? z.Height : -1;

long wH = w != null ? w.Height : -1;

if (wH > xH)

root.Height = wH + 1;

else

root.Height = xH + 1;

if (yH > zH)

b.Height = yH + 1;

else

b.Height = zH + 1;

if (b.Height > root.Height)

c.Height = b.Height + 1;

else

c.Height = root.Height + 1;

UpdateHeight(c);

return c;

}

}

}

## Бенчмарк к задаче 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№ теста** | **Результат** | **Время, с** | **Память** | **Размер входного файла** | **Размер выходного файла** |
| Max |  | 0.703 | 46481408 | 2678110 | 731071 |
| 1 | OK | 0.031 | 10629120 | 33 | 19 |
| 2 | OK | 0.031 | 10592256 | 114 | 66 |
| 3 | OK | 0.062 | 10555392 | 154 | 90 |
| 4 | OK | 0.031 | 10645504 | 154 | 91 |
| 5 | OK | 0.015 | 10706944 | 154 | 90 |
| 6 | OK | 0.031 | 10543104 | 154 | 95 |
| 7 | OK | 0.031 | 10575872 | 154 | 91 |
| 8 | OK | 0.031 | 10637312 | 154 | 94 |
| 9 | OK | 0.031 | 10596352 | 154 | 95 |
| 10 | OK | 0.015 | 10539008 | 154 | 90 |
| 11 | OK | 0.031 | 10559488 | 154 | 90 |
| 12 | OK | 0.015 | 10387456 | 154 | 90 |
| 13 | OK | 0.031 | 10375168 | 154 | 95 |
| 14 | OK | 0.031 | 10346496 | 154 | 97 |
| 15 | OK | 0.031 | 10522624 | 154 | 94 |
| 16 | OK | 0.015 | 10416128 | 154 | 93 |
| 17 | OK | 0.046 | 10342400 | 154 | 90 |
| 18 | OK | 0.031 | 10321920 | 154 | 90 |
| 19 | OK | 0.031 | 10424320 | 154 | 98 |
| 20 | OK | 0.031 | 10362880 | 154 | 93 |
| 21 | OK | 0.015 | 10358784 | 154 | 92 |
| 22 | OK | 0.015 | 10452992 | 154 | 98 |
| 23 | OK | 0.171 | 28426240 | 1000008 | 616458 |
| 24 | OK | 0.171 | 28397568 | 1000008 | 622272 |
| 25 | OK | 0.156 | 28405760 | 1000008 | 625335 |
| 26 | OK | 0.156 | 28479488 | 1000008 | 628546 |
| 27 | OK | 0.156 | 28418048 | 1000008 | 631472 |
| 28 | OK | 0.203 | 28401664 | 1000008 | 632217 |
| 29 | OK | 0.171 | 28442624 | 1000008 | 631772 |
| 30 | OK | 0.156 | 28401664 | 1000008 | 631071 |
| 31 | OK | 0.156 | 28364800 | 1000008 | 630132 |
| 32 | OK | 0.156 | 28553216 | 1017957 | 630451 |
| 33 | OK | 0.156 | 28495872 | 1000008 | 616595 |
| 34 | OK | 0.156 | 28356608 | 1000008 | 622199 |
| 35 | OK | 0.140 | 28372992 | 1000008 | 625057 |
| 36 | OK | 0.140 | 28385280 | 1000008 | 628040 |
| 37 | OK | 0.156 | 28442624 | 1000008 | 631495 |
| 38 | OK | 0.156 | 28401664 | 1000008 | 632086 |
| 39 | OK | 0.171 | 28479488 | 1000008 | 631753 |
| 40 | OK | 0.156 | 28389376 | 1000008 | 630849 |
| 41 | OK | 0.171 | 28446720 | 1000008 | 630110 |
| 42 | OK | 0.156 | 28594176 | 1018151 | 630800 |
| 43 | OK | 0.031 | 10366976 | 756 | 369 |
| 44 | OK | 0.031 | 10354688 | 758 | 432 |
| 45 | OK | 0.031 | 10428416 | 1659 | 408 |
| 46 | OK | 0.031 | 10366976 | 723 | 383 |
| 47 | OK | 0.031 | 10375168 | 723 | 385 |
| 48 | OK | 0.031 | 10473472 | 723 | 415 |
| 49 | OK | 0.031 | 10452992 | 723 | 415 |
| 50 | OK | 0.031 | 10448896 | 1668 | 377 |
| 51 | OK | 0.031 | 10366976 | 1660 | 396 |
| 52 | OK | 0.031 | 10674176 | 5348 | 2337 |
| 53 | OK | 0.031 | 10600448 | 5350 | 2848 |
| 54 | OK | 0.046 | 10657792 | 10439 | 2648 |
| 55 | OK | 0.031 | 10616832 | 5238 | 2343 |
| 56 | OK | 0.015 | 10665984 | 5238 | 2465 |
| 57 | OK | 0.031 | 10588160 | 5238 | 2719 |
| 58 | OK | 0.046 | 10592256 | 5238 | 2719 |
| 59 | OK | 0.046 | 10715136 | 10450 | 2421 |
| 60 | OK | 0.031 | 10706944 | 10439 | 2405 |
| 61 | OK | 0.031 | 11542528 | 32784 | 12708 |
| 62 | OK | 0.031 | 11522048 | 32787 | 14896 |
| 63 | OK | 0.046 | 11599872 | 56716 | 12715 |
| 64 | OK | 0.031 | 11616256 | 31674 | 12778 |
| 65 | OK | 0.046 | 11546624 | 31674 | 13220 |
| 66 | OK | 0.031 | 11530240 | 31674 | 14383 |
| 67 | OK | 0.031 | 11591680 | 31674 | 14825 |
| 68 | OK | 0.046 | 11563008 | 56748 | 13671 |
| 69 | OK | 0.031 | 11595776 | 56716 | 13193 |
| 70 | OK | 0.062 | 17539072 | 162462 | 57855 |
| 71 | OK | 0.062 | 17477632 | 162466 | 68948 |
| 72 | OK | 0.062 | 18395136 | 258205 | 71756 |
| 73 | OK | 0.046 | 16871424 | 152067 | 59306 |
| 74 | OK | 0.078 | 16855040 | 152067 | 59903 |
| 75 | OK | 0.062 | 16834560 | 152067 | 66900 |
| 76 | OK | 0.046 | 16834560 | 152067 | 67497 |
| 77 | OK | 0.078 | 17768448 | 258312 | 70001 |
| 78 | OK | 0.062 | 17584128 | 258332 | 58111 |
| 79 | OK | 0.203 | 27082752 | 811002 | 274035 |
| 80 | OK | 0.218 | 27099136 | 811006 | 332612 |
| 81 | OK | 0.281 | 28688384 | 1222794 | 299942 |
| 82 | OK | 0.171 | 27275264 | 799892 | 286940 |
| 83 | OK | 0.171 | 27267072 | 799892 | 282227 |
| 84 | OK | 0.156 | 27332608 | 799892 | 324420 |
| 85 | OK | 0.187 | 27303936 | 799892 | 319707 |
| 86 | OK | 0.218 | 27226112 | 1222871 | 284516 |
| 87 | OK | 0.187 | 27176960 | 1223246 | 288111 |
| 88 | OK | 0.437 | 44290048 | 1888898 | 600000 |
| 89 | OK | 0.421 | 44371968 | 1888903 | 731071 |
| 90 | OK | 0.703 | 46481408 | 2677526 | 600067 |
| 91 | OK | 0.328 | 37707776 | 1777788 | 601696 |
| 92 | OK | 0.312 | 37699584 | 1777788 | 632768 |
| 93 | OK | 0.328 | 37670912 | 1777788 | 698302 |
| 94 | OK | 0.312 | 37728256 | 1777788 | 698303 |
| 95 | OK | 0.531 | 41881600 | 2678110 | 611713 |
| 96 | OK | 0.390 | 41930752 | 2677266 | 600286 |