**Министерство образования и науки Российской Федерации**

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

Факультет программной инженерии и компьютерной техники

Кафедра информатики и прикладной математики   
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Алгоритмы и структуры данных»

**ОТЧЁТ**

по лабораторной работе №7

Студент Кочарян С.А. группы P3218

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2019

## Проверка сбалансированности



using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Week7

{

public class main

{

public static void Main(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;

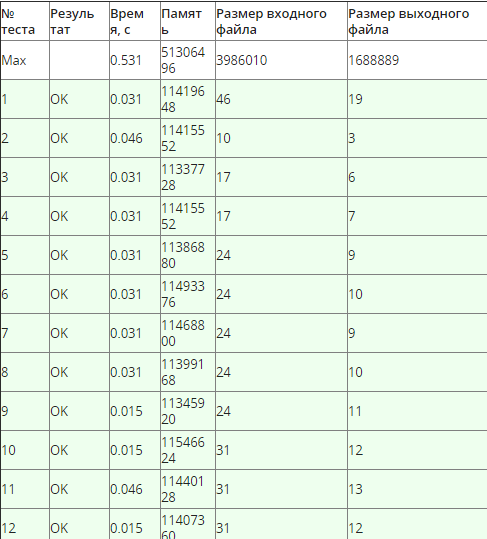
else

return 0;

}

}

}



## Делаю я левый поворот...



using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Week7

{

public class main

{

public static void Main(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;

//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;

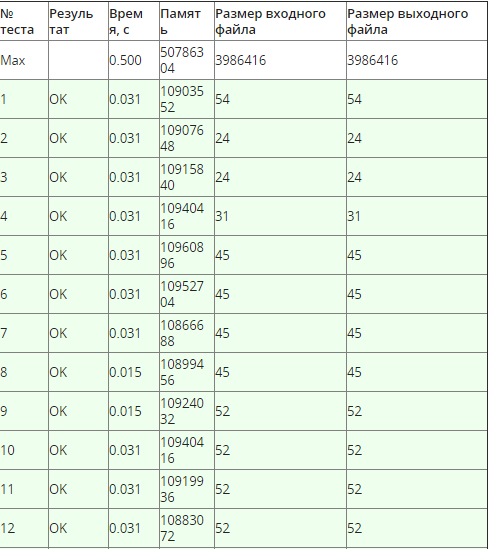
UpdateHeight(c);

return c;

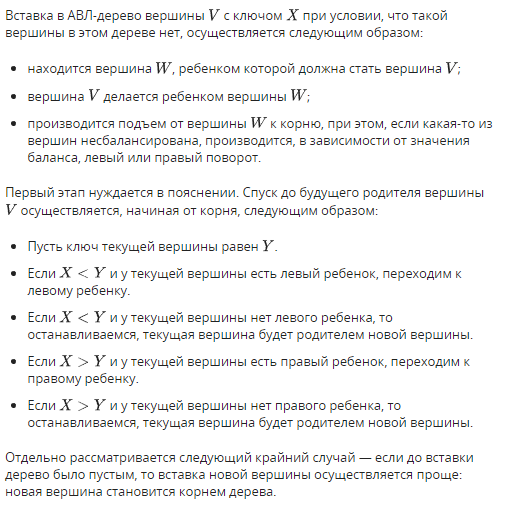
}

}

}}



## Вставка в АВЛ-дерево



using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Week7

{

public class main

{

public static void Main(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;

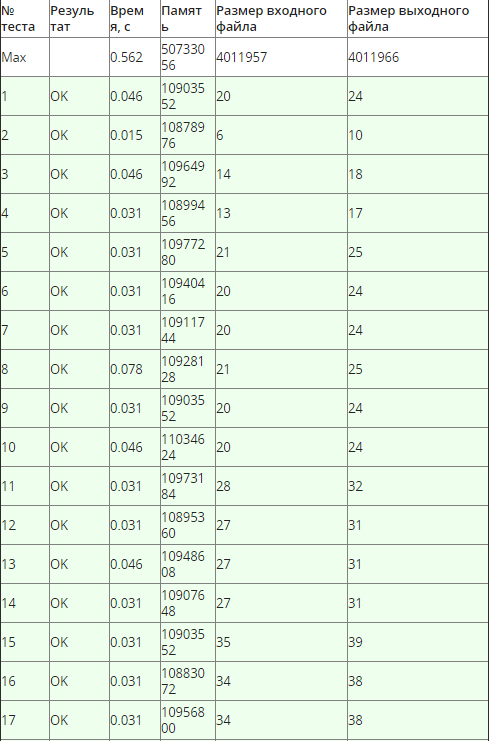
UpdateHeight(c);

return c;

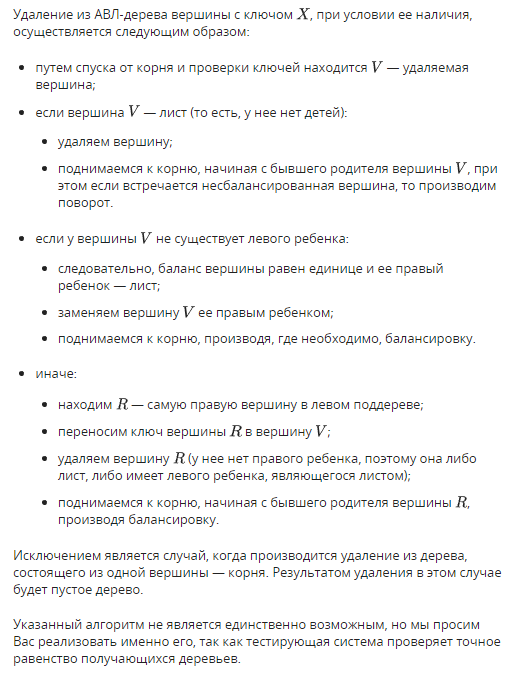
}

}

}



## Удаление из АВЛ-дерева



using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Week7

{

public class main

{

public static void Main(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;

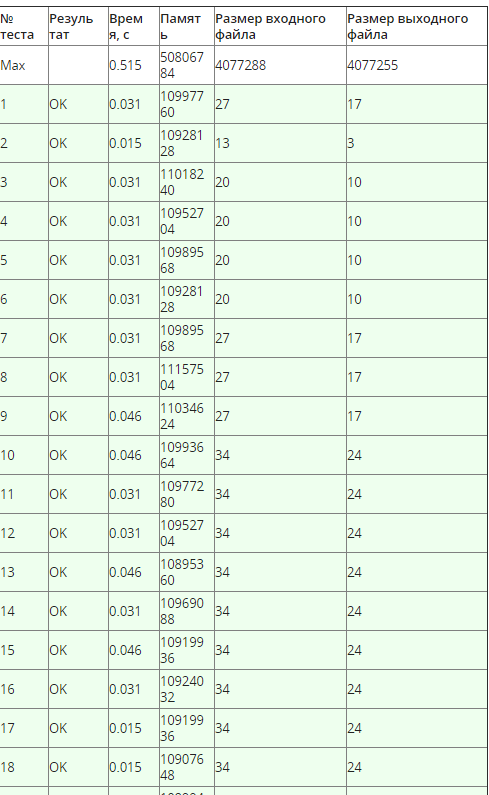
UpdateHeight(c);

return c;

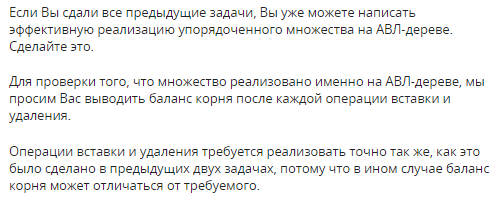
}

}

}



## Упорядоченное множество на АВЛ-дереве



using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Week7

{

public class main

{

public static void Main(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;

}

}

}

}

