**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  
неделя седьмая

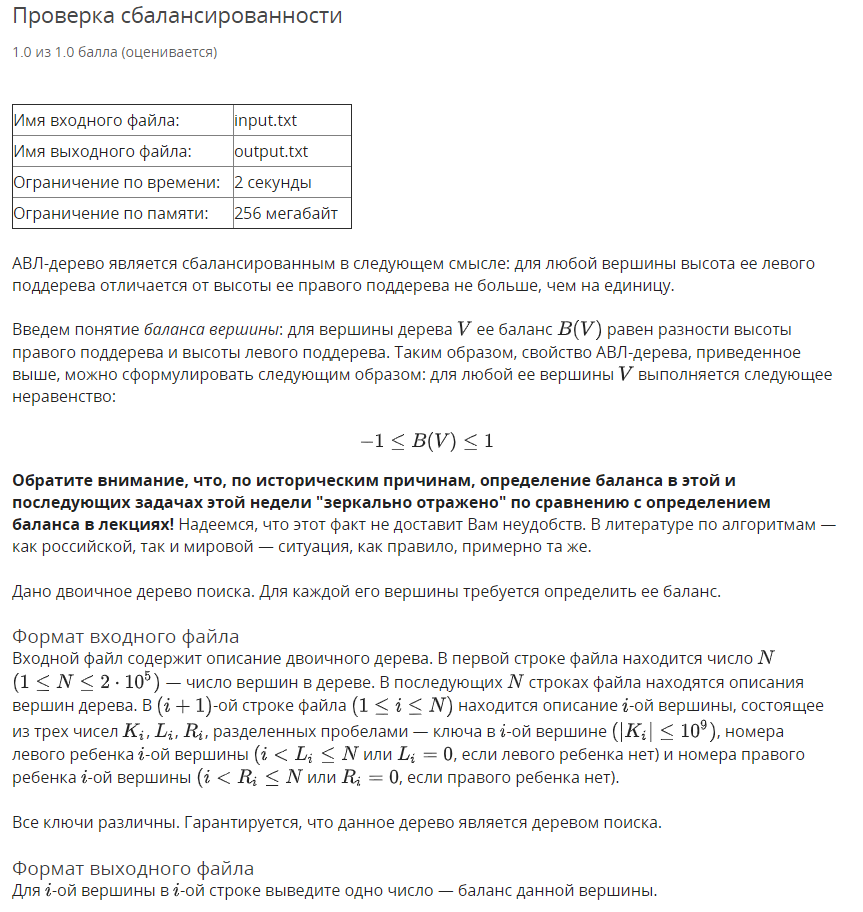
Выполнил:  
Айгузин Иван Олегович   
P3218

Преподаватели:

Романов Алексей Андреевич  
Волчек Дмитрий Геннадьевич

Санкт-Петербург

2018





using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading;

namespace Week07.Task01 {

public sealed class TreeNode {

private int \_depth = int.MinValue;

public int Key { get; set; }

public TreeNode Left { get; set; }

public TreeNode Right { get; set; }

public TreeNode Parent { get; set; }

public int Depth {

get {

if (\_depth == int.MinValue) {

\_depth = Math.Max(Left?.Depth ?? 0, Right?.Depth ?? 0) + 1;

}

return \_depth;

}

set { \_depth = value; }

}

public int Balance {

get { return (Right?.Depth ?? 0) - (Left?.Depth ?? 0); }

}

private void UpdateDepth() {

var node = this;

while (node != null) {

var rH = node.Right?.Depth ?? -1;

var lH = node.Left?.Depth ?? -1;

node.Depth = rH > lH ? rH + 1 : lH + 1;

node = node.Parent;

}

}

public override string ToString() {

return $"{nameof(Key)}: {Key}, {nameof(Depth)}: {Depth}, {nameof(Balance)}: {Balance}";

}

public static List<TreeNode> ReadTree(int n) {

var tree = Enumerable.Range(0, n)

.Select(x => new TreeNode())

.ToList();

for (var i = 0; i < n; i++) {

var a = ReadIntList();

var node = tree[i];

var l = a[1] - 1;

var r = a[2] - 1;

(node.Key, node.Left, node.Right) =

(a[0], l != -1 ? tree[l] : null, r != -1 ? tree[r] : null);

if (node.Left != null) {

node.Left.Parent = node;

}

if (node.Right != null) {

node.Right.Parent = node;

}

}

return tree;

}

private static int[] ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToArray();

}

}

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

var thread = new Thread(Run, int.MaxValue / 10);

thread.Start();

thread.Join();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var n = ReadIntList()[0];

var tree = TreeNode.ReadTree(n);

tree.ForEach(x => Console.WriteLine(x.Balance));

}

private static int[] ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToArray();

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

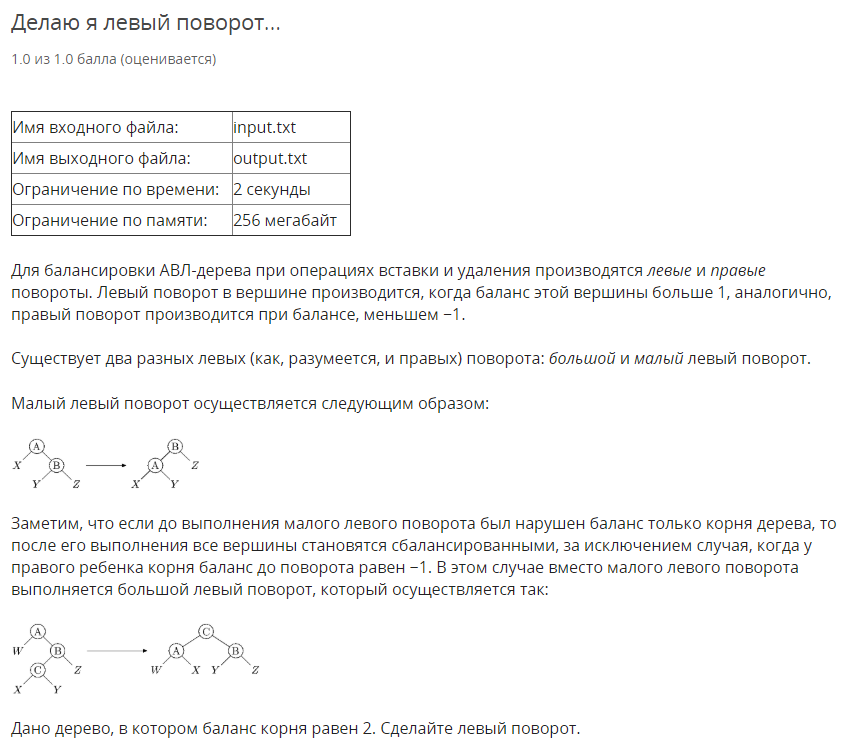
\_in?.Dispose();

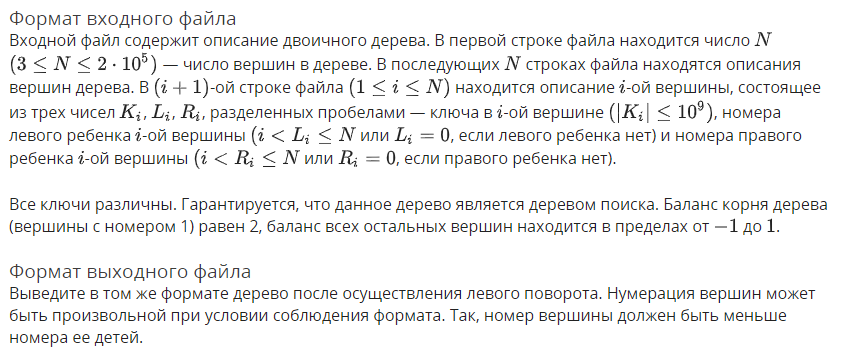
\_out?.Dispose();

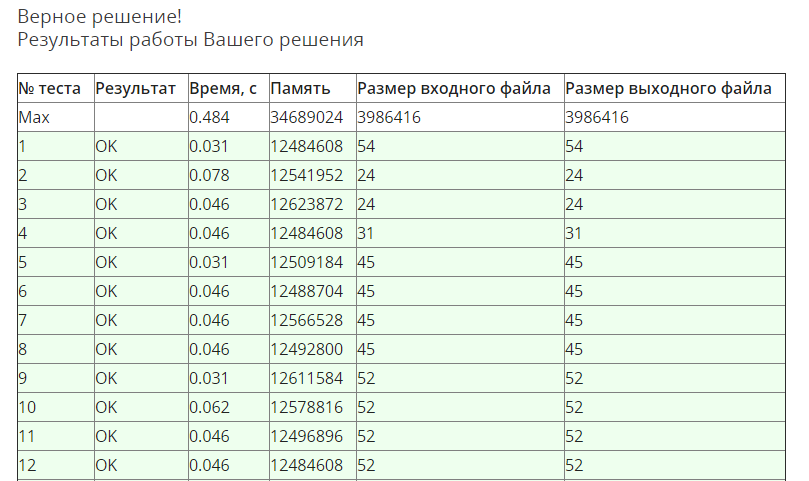
}

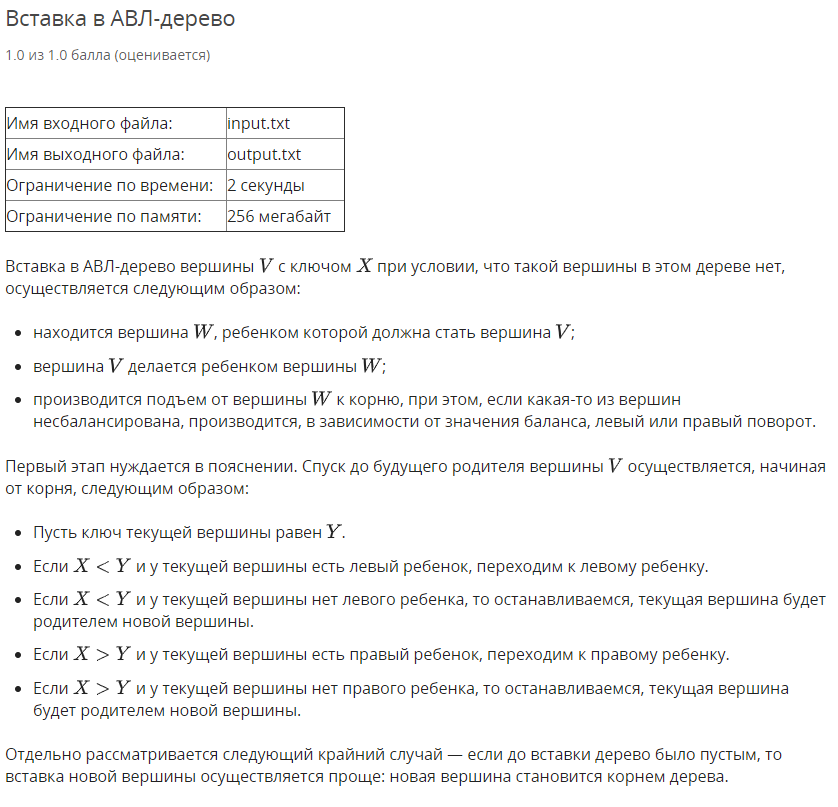
}

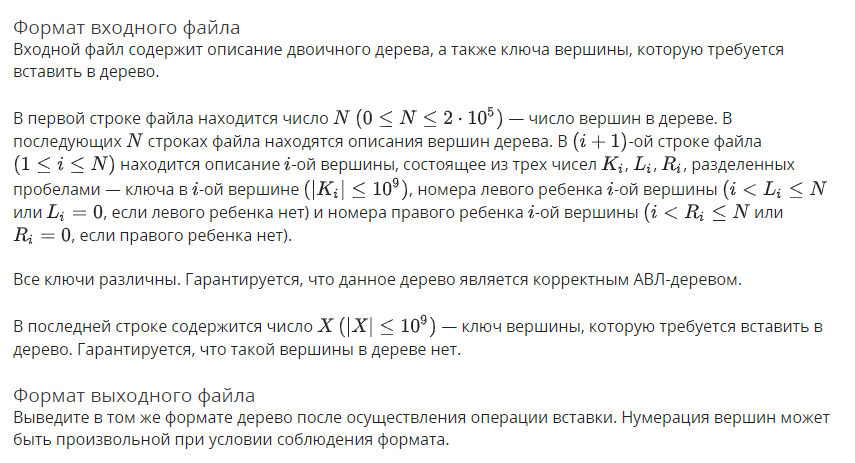
}











using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading;

namespace Week07.Task03 {

public sealed class TreeNode {

public int Key { get; set; }

public TreeNode Left { get; set; }

public TreeNode Right { get; set; }

public TreeNode Parent { get; set; }

public int Height { get; set; }

public int Balance {

get { return (Right?.Height ?? -1) - (Left?.Height ?? -1); }

}

private void UpdateHeight() {

var rH = Right?.Height ?? -1;

var lH = Left?.Height ?? -1;

Height = rH > lH ? rH + 1 : lH + 1;

Parent?.UpdateHeight();

}

public TreeNode Insert(int value) {

if (value < Key) {

if (Left != null) {

return Left.Insert(value);

}

Left = new TreeNode {

Parent = this,

Key = value

};

Left.UpdateHeight();

return Left;

}

if (Right != null) {

return Right.Insert(value);

}

Right = new TreeNode {

Parent = this,

Key = value

};

Right.UpdateHeight();

return Right;

}

public TreeNode BalanceTree() {

var current = this;

var balance = Balance;

if (balance > 1) {

current = Right?.Balance == -1 ? BigLeftTurn() : SmallLeftTurn();

}

if (balance < -1) {

current = Left?.Balance == 1 ? BigRightTurn() : SmallRightTurn();

}

return current.Parent?.BalanceTree() ?? current;

}

private TreeNode SmallLeftTurn() {

var child = Right;

var parent = Parent;

var x = Left;

var y = Right.Left;

var z = Right.Right;

//Parents

child.Parent = parent;

Parent = child;

if (x != null) {

x.Parent = this;

}

if (y != null) {

y.Parent = this;

}

if (z != null) {

z.Parent = child;

}

//Childs

Left = x;

Right = y;

child.Left = this;

child.Right = z;

if (parent != null) {

if (parent.Right == this) {

parent.Right = child;

}

else {

parent.Left = child;

}

}

//Heights

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

Height = xH > yH ? xH + 1 : yH + 1;

child.Height = Height > zH ? Height + 1 : zH + 1;

child.UpdateHeight();

return child;

}

private TreeNode SmallRightTurn() {

var child = Left;

var parent = Parent;

var x = Right;

var y = Left.Left;

var z = Left.Right;

//Parents

child.Parent = parent;

Parent = child;

if (x != null) {

x.Parent = this;

}

if (y != null) {

y.Parent = child;

}

if (z != null) {

z.Parent = this;

}

//Childs

Left = z;

Right = x;

child.Left = y;

child.Right = this;

if (parent != null) {

if (parent.Right == this) {

parent.Right = child;

}

else {

parent.Left = child;

}

}

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

Height = zH > xH ? zH + 1 : xH + 1;

child.Height = y.Height > Height ? yH + 1 : Height + 1;

child.UpdateHeight();

return child;

}

public TreeNode BigRightTurn() {

var w = Right;

var parent = Parent;

var b = Left;

var c = Left.Right;

var z = b.Left;

var x = c.Left;

var y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

Parent = c;

if (w != null) {

w.Parent = this;

}

if (z != null) {

z.Parent = b;

}

if (y != null) {

y.Parent = this;

}

if (x != null) {

x.Parent = b;

}

//Childs

if (parent != null) {

if (parent.Right == this) {

parent.Right = c;

}

else {

parent.Left = c;

}

}

c.Left = b;

c.Right = this;

b.Left = z;

b.Right = x;

Left = y;

Right = w;

//Depths

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

var wH = w?.Height ?? -1;

b.Height = zH > xH ? zH + 1 : xH + 1;

Height = yH > wH ? yH + 1 : wH + 1;

c.Height = b.Height > Height ? b.Height + 1 : Height + 1;

c.UpdateHeight();

return c;

}

public TreeNode BigLeftTurn() {

var w = Left;

var parent = Parent;

var b = Right;

var c = Right.Left;

var z = b.Right;

var x = c.Left;

var y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

Parent = c;

if (w != null) {

w.Parent = this;

}

if (z != null) {

z.Parent = b;

}

if (y != null) {

y.Parent = b;

}

if (x != null) {

x.Parent = this;

}

//Childs

if (parent != null) {

if (parent.Right == this) {

parent.Right = c;

}

else {

parent.Left = c;

}

}

c.Left = this;

c.Right = b;

b.Left = y;

b.Right = z;

Left = w;

Right = x;

//Depths

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

var wH = w?.Height ?? -1;

Height = wH > xH ? wH + 1 : xH + 1;

b.Height = yH > zH ? yH + 1 : zH + 1;

c.Height = b.Height > Height ? b.Height + 1 : Height + 1;

c.UpdateHeight();

return c;

}

public override string ToString() {

return $"{nameof(Key)}: {Key}, {nameof(Height)}: {Height}, {nameof(Balance)}: {Balance}";

}

public static TreeNode ReadTree(int n) {

var root = new TreeNode {

Key = ReadIntList()[0]

};

for (var i = 0; i < n - 1; i++) {

root.Insert(ReadIntList()[0]);

}

return root;

}

private static int[] ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToArray();

}

}

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

var thread = new Thread(Run, int.MaxValue / 10);

thread.Start();

thread.Join();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var n = ReadIntList()[0];

if (n == 0) {

Console.WriteLine(1);

Console.WriteLine($"{ReadIntList()[0]} 0 0");

return;

}

var tree = TreeNode.ReadTree(n);

var newNode = tree.Insert(ReadIntList()[0]);

var root = newNode.BalanceTree();

Console.WriteLine(n + 1);

PrintTree(root);

void PrintTree(TreeNode node) {

var queue = new Queue<TreeNode>();

var counter = 1;

queue.Enqueue(node);

while (queue.Count > 0) {

var current = queue.Dequeue();

var l = 0;

var r = 0;

if (current.Left != null) {

queue.Enqueue(current.Left);

l = ++counter;

}

if (current.Right != null) {

queue.Enqueue(current.Right);

r = ++counter;

}

Console.WriteLine($"{current.Key} {l} {r}");

}

}

}

private static int[] ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToArray();

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

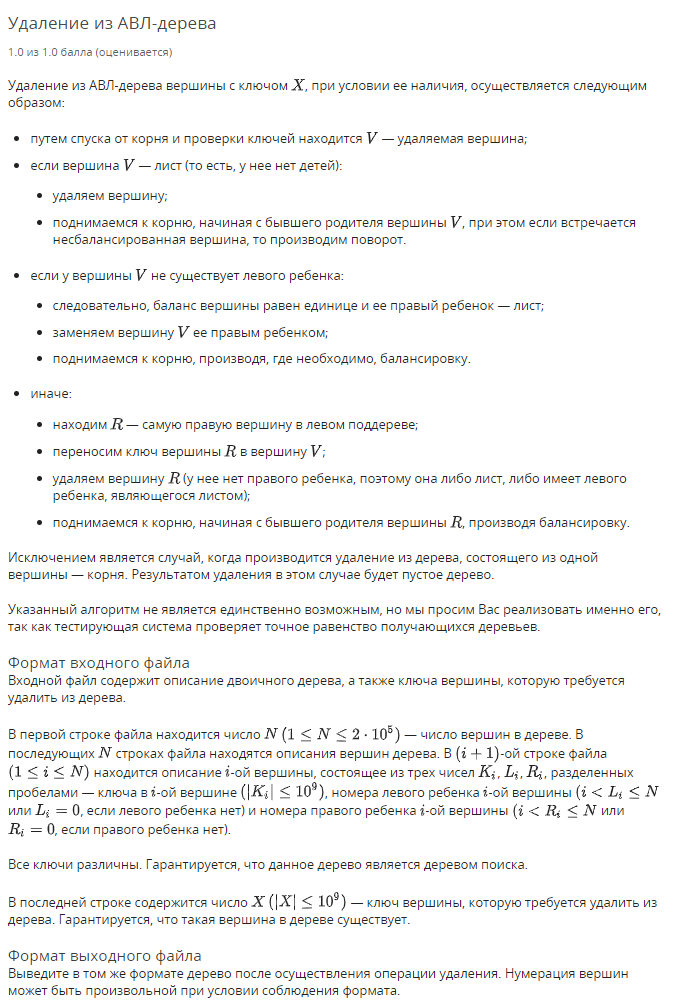
\_in?.Dispose();

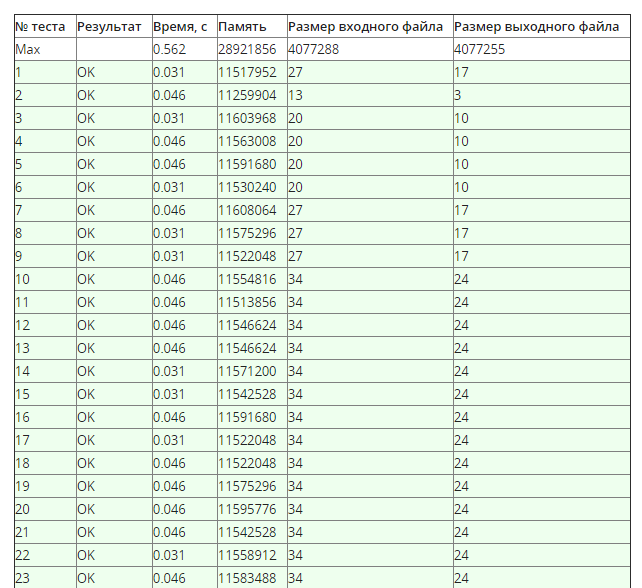
\_out?.Dispose();

}

}

}





using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading;

namespace Week07.Task04 {

public sealed class TreeNode {

public int Key { get; set; }

public TreeNode Left { get; set; }

public TreeNode Right { get; set; }

public TreeNode Parent { get; set; }

public int Height { get; set; }

public int Balance {

get { return (Right?.Height ?? -1) - (Left?.Height ?? -1); }

}

private void UpdateHeight() {

var rH = Right?.Height ?? -1;

var lH = Left?.Height ?? -1;

Height = rH > lH ? rH + 1 : lH + 1;

Parent?.UpdateHeight();

}

public TreeNode Insert(int value) {

if (value < Key) {

if (Left != null) {

return Left.Insert(value);

}

Left = new TreeNode {

Parent = this,

Key = value

};

Left.UpdateHeight();

return Left;

}

if (Right != null) {

return Right.Insert(value);

}

Right = new TreeNode {

Parent = this,

Key = value

};

Right.UpdateHeight();

return Right;

}

public TreeNode BalanceTree() {

var current = this;

var balance = Balance;

if (balance > 1) {

current = Right?.Balance == -1 ? BigLeftTurn() : SmallLeftTurn();

}

if (balance < -1) {

current = Left?.Balance == 1 ? BigRightTurn() : SmallRightTurn();

}

return current.Parent?.BalanceTree() ?? current;

}

private TreeNode SmallLeftTurn() {

var child = Right;

var parent = Parent;

var x = Left;

var y = Right.Left;

var z = Right.Right;

//Parents

child.Parent = parent;

Parent = child;

if (x != null) {

x.Parent = this;

}

if (y != null) {

y.Parent = this;

}

if (z != null) {

z.Parent = child;

}

//Childs

Left = x;

Right = y;

child.Left = this;

child.Right = z;

if (parent != null) {

if (parent.Right == this) {

parent.Right = child;

}

else {

parent.Left = child;

}

}

//Heights

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

Height = xH > yH ? xH + 1 : yH + 1;

child.Height = Height > zH ? Height + 1 : zH + 1;

child.UpdateHeight();

return child;

}

private TreeNode SmallRightTurn() {

var child = Left;

var parent = Parent;

var x = Right;

var y = Left.Left;

var z = Left.Right;

//Parents

child.Parent = parent;

Parent = child;

if (x != null) {

x.Parent = this;

}

if (y != null) {

y.Parent = child;

}

if (z != null) {

z.Parent = this;

}

//Childs

Left = z;

Right = x;

child.Left = y;

child.Right = this;

if (parent != null) {

if (parent.Right == this) {

parent.Right = child;

}

else {

parent.Left = child;

}

}

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

Height = zH > xH ? zH + 1 : xH + 1;

child.Height = y.Height > Height ? yH + 1 : Height + 1;

child.UpdateHeight();

return child;

}

public TreeNode BigRightTurn() {

var w = Right;

var parent = Parent;

var b = Left;

var c = Left.Right;

var z = b.Left;

var x = c.Left;

var y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

Parent = c;

if (w != null) {

w.Parent = this;

}

if (z != null) {

z.Parent = b;

}

if (y != null) {

y.Parent = this;

}

if (x != null) {

x.Parent = b;

}

//Childs

if (parent != null) {

if (parent.Right == this) {

parent.Right = c;

}

else {

parent.Left = c;

}

}

c.Left = b;

c.Right = this;

b.Left = z;

b.Right = x;

Left = y;

Right = w;

//Depths

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

var wH = w?.Height ?? -1;

b.Height = zH > xH ? zH + 1 : xH + 1;

Height = yH > wH ? yH + 1 : wH + 1;

c.Height = b.Height > Height ? b.Height + 1 : Height + 1;

c.UpdateHeight();

return c;

}

public TreeNode BigLeftTurn() {

var w = Left;

var parent = Parent;

var b = Right;

var c = Right.Left;

var z = b.Right;

var x = c.Left;

var y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

Parent = c;

if (w != null) {

w.Parent = this;

}

if (z != null) {

z.Parent = b;

}

if (y != null) {

y.Parent = b;

}

if (x != null) {

x.Parent = this;

}

//Childs

if (parent != null) {

if (parent.Right == this) {

parent.Right = c;

}

else {

parent.Left = c;

}

}

c.Left = this;

c.Right = b;

b.Left = y;

b.Right = z;

Left = w;

Right = x;

//Depths

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

var wH = w?.Height ?? -1;

Height = wH > xH ? wH + 1 : xH + 1;

b.Height = yH > zH ? yH + 1 : zH + 1;

c.Height = b.Height > Height ? b.Height + 1 : Height + 1;

c.UpdateHeight();

return c;

}

public TreeNode Search(int value) {

if (value == Key) {

return this;

}

return value < Key ? Left?.Search(value) : Right?.Search(value);

}

public TreeNode Previous() {

return Left?.Maximum() ?? this;

}

public TreeNode Maximum() {

return Right?.Maximum() ?? this;

}

public TreeNode Remove() {

if (Left == null && Right == null) {

if (Parent == null) {

return null;

}

if (Parent.Left == this) {

Parent.Left = null;

}

else {

Parent.Right = null;

}

Parent.UpdateHeight();

return Parent.BalanceTree();

}

if ((Left == null) ^ (Right == null)) {

if (Left != null) {

if (Parent != null) {

if (Parent.Left == this) {

Parent.Left = Left;

}

else {

Parent.Right = Left;

}

Parent.UpdateHeight();

}

Left.Parent = Parent;

return Left.BalanceTree();

}

if (Parent != null) {

if (Parent.Left == this) {

Parent.Left = Right;

}

else {

Parent.Right = Right;

}

Parent.UpdateHeight();

}

Right.Parent = Parent;

return Right.BalanceTree();

}

//Two child

if (Left != null && Right != null) {

var prev = Previous();

prev.Remove();

Key = prev.Key;

}

return BalanceTree();

}

public override string ToString() {

return $"{nameof(Key)}: {Key}, {nameof(Height)}: {Height}, {nameof(Balance)}: {Balance}";

}

public static TreeNode ReadTree(int n) {

var root = new TreeNode {

Key = ReadIntList()[0]

};

for (var i = 0; i < n - 1; i++) {

root.Insert(ReadIntList()[0]);

}

return root;

}

private static int[] ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToArray();

}

}

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

var thread = new Thread(Run, int.MaxValue / 10);

thread.Start();

thread.Join();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var n = ReadIntList()[0];

var tree = TreeNode.ReadTree(n);

var root = tree.Search(ReadIntList()[0]).Remove();

Console.WriteLine(n - 1);

if (n - 1 != 0) {

PrintTree(root);

}

void PrintTree(TreeNode node) {

var queue = new Queue<TreeNode>();

var counter = 1;

queue.Enqueue(node);

while (queue.Count > 0) {

var current = queue.Dequeue();

var l = 0;

var r = 0;

if (current.Left != null) {

queue.Enqueue(current.Left);

l = ++counter;

}

if (current.Right != null) {

queue.Enqueue(current.Right);

r = ++counter;

}

Console.WriteLine($"{current.Key} {l} {r}");

}

}

}

private static int[] ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToArray();

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

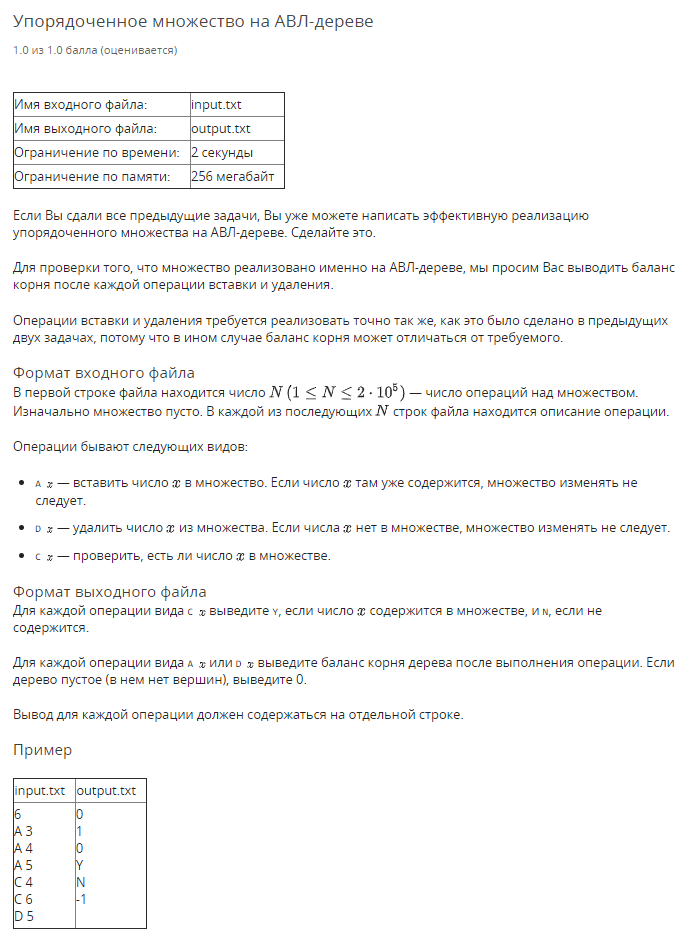
\_in?.Dispose();

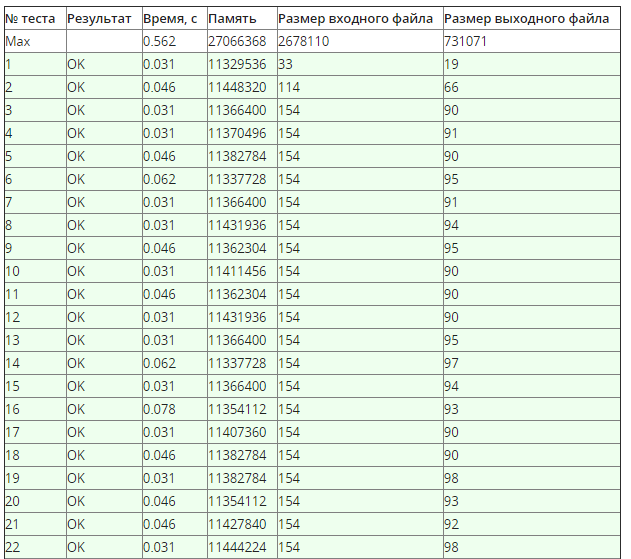
\_out?.Dispose();

}

}

}





using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading;

namespace Week07.Task05 {

public sealed class TreeNode {

public int Key { get; set; }

public TreeNode Left { get; set; }

public TreeNode Right { get; set; }

public TreeNode Parent { get; set; }

public int Height { get; set; }

public int Balance {

get { return (Right?.Height ?? -1) - (Left?.Height ?? -1); }

}

private void UpdateHeight() {

var rH = Right?.Height ?? -1;

var lH = Left?.Height ?? -1;

Height = rH > lH ? rH + 1 : lH + 1;

Parent?.UpdateHeight();

}

public TreeNode Insert(int value) {

if (value < Key) {

if (Left != null) {

return Left.Insert(value);

}

Left = new TreeNode {

Parent = this,

Key = value

};

Left.UpdateHeight();

return Left.BalanceTree();

}

if (Right != null) {

return Right.Insert(value);

}

Right = new TreeNode {

Parent = this,

Key = value

};

Right.UpdateHeight();

return Right.BalanceTree();

}

public TreeNode BalanceTree() {

var current = this;

while (current != null) {

var balance = current.Balance;

if (balance > 1) {

current = current.Right.Balance == -1 ? current.BigLeftTurn() : current.SmallLeftTurn();

}

if (balance < -1) {

current = current.Left.Balance == 1 ? current.BigRightTurn() : current.SmallRightTurn();

}

if (current.Parent == null)

return current;

current = current.Parent;

}

return current;

}

private TreeNode SmallLeftTurn() {

var child = Right;

var parent = Parent;

var x = Left;

var y = Right.Left;

var z = Right.Right;

//Parents

child.Parent = parent;

Parent = child;

if (x != null) {

x.Parent = this;

}

if (y != null) {

y.Parent = this;

}

if (z != null) {

z.Parent = child;

}

//Childs

Left = x;

Right = y;

child.Left = this;

child.Right = z;

if (parent != null) {

if (parent.Right == this) {

parent.Right = child;

} else {

parent.Left = child;

}

}

//Heights

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

Height = xH > yH ? xH + 1 : yH + 1;

child.Height = Height > zH ? Height + 1 : zH + 1;

child.UpdateHeight();

return child;

}

private TreeNode SmallRightTurn() {

var child = Left;

var parent = Parent;

var x = Right;

var y = Left.Left;

var z = Left.Right;

//Parents

child.Parent = parent;

Parent = child;

if (x != null) {

x.Parent = this;

}

if (y != null) {

y.Parent = child;

}

if (z != null) {

z.Parent = this;

}

//Childs

Left = z;

Right = x;

child.Left = y;

child.Right = this;

if (parent != null) {

if (parent.Right == this) {

parent.Right = child;

} else {

parent.Left = child;

}

}

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

Height = zH > xH ? zH + 1 : xH + 1;

child.Height = y.Height > Height ? yH + 1 : Height + 1;

child.UpdateHeight();

return child;

}

public TreeNode BigRightTurn() {

var w = Right;

var parent = Parent;

var b = Left;

var c = Left.Right;

var z = b.Left;

var x = c.Left;

var y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

Parent = c;

if (w != null) {

w.Parent = this;

}

if (z != null) {

z.Parent = b;

}

if (y != null) {

y.Parent = this;

}

if (x != null) {

x.Parent = b;

}

//Childs

if (parent != null) {

if (parent.Right == this) {

parent.Right = c;

} else {

parent.Left = c;

}

}

c.Left = b;

c.Right = this;

b.Left = z;

b.Right = x;

Left = y;

Right = w;

//Depths

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

var wH = w?.Height ?? -1;

b.Height = zH > xH ? zH + 1 : xH + 1;

Height = yH > wH ? yH + 1 : wH + 1;

c.Height = b.Height > Height ? b.Height + 1 : Height + 1;

c.UpdateHeight();

return c;

}

public TreeNode BigLeftTurn() {

var w = Left;

var parent = Parent;

var b = Right;

var c = Right.Left;

var z = b.Right;

var x = c.Left;

var y = c.Right;

//Parents

c.Parent = parent;

b.Parent = c;

Parent = c;

if (w != null) {

w.Parent = this;

}

if (z != null) {

z.Parent = b;

}

if (y != null) {

y.Parent = b;

}

if (x != null) {

x.Parent = this;

}

//Childs

if (parent != null) {

if (parent.Right == this) {

parent.Right = c;

} else {

parent.Left = c;

}

}

c.Left = this;

c.Right = b;

b.Left = y;

b.Right = z;

Left = w;

Right = x;

//Depths

var xH = x?.Height ?? -1;

var yH = y?.Height ?? -1;

var zH = z?.Height ?? -1;

var wH = w?.Height ?? -1;

Height = wH > xH ? wH + 1 : xH + 1;

b.Height = yH > zH ? yH + 1 : zH + 1;

c.Height = b.Height > Height ? b.Height + 1 : Height + 1;

c.UpdateHeight();

return c;

}

public TreeNode Search(int value) {

if (value == Key) {

return this;

}

return value < Key ? Left?.Search(value) : Right?.Search(value);

}

public TreeNode Previous() {

return Left?.Maximum() ?? this;

}

public TreeNode Maximum() {

return Right?.Maximum() ?? this;

}

public TreeNode Remove() {

if (Left == null && Right == null) {

if (Parent == null) {

return null;

}

if (Parent.Left == this) {

Parent.Left = null;

} else {

Parent.Right = null;

}

Parent.UpdateHeight();

return Parent.BalanceTree();

}

if ((Left == null) ^ (Right == null)) {

if (Left != null) {

if (Parent != null) {

if (Parent.Left == this) {

Parent.Left = Left;

} else {

Parent.Right = Left;

}

Parent.UpdateHeight();

}

Left.Parent = Parent;

return Left.BalanceTree();

}

if (Parent != null) {

if (Parent.Left == this) {

Parent.Left = Right;

} else {

Parent.Right = Right;

}

Parent.UpdateHeight();

}

Right.Parent = Parent;

return Right.BalanceTree();

}

//Two child

if (Left != null && Right != null) {

var prev = Previous();

prev.Remove();

Key = prev.Key;

}

return BalanceTree();

}

public override string ToString() {

return $"{nameof(Key)}: {Key}, {nameof(Height)}: {Height}, {nameof(Balance)}: {Balance}";

}

public static TreeNode ReadTree(int n) {

var root = new TreeNode {

Key = ReadIntList()[0]

};

for (var i = 0; i < n - 1; i++) {

root.Insert(ReadIntList()[0]);

}

return root;

}

private static int[] ReadIntList() {

return Console.ReadLine()

.Split(' ')

.Select(int.Parse)

.ToArray();

}

}

public sealed class Program {

private static StreamReader \_in;

private static StreamWriter \_out;

private static void Main(string[] args) {

if (!args.Contains("console")) {

SetupIO();

}

var thread = new Thread(Run, int.MaxValue / 10);

thread.Start();

thread.Join();

if (args.Contains("console")) {

Console.ReadLine();

}

DisposeIO();

}

private static void Run() {

var n = int.Parse(ReadStringList()[0]);

var root = default(TreeNode);

for (var i = 0; i < n; i++) {

var (command, x) = ReadCommand();

switch (command) {

case "A":

if (root == null) {

root = new TreeNode {

Key = x

};

Console.WriteLine(root.Balance);

continue;

}

if (root.Search(x) == null) {

root = root.Insert(x);

}

Console.WriteLine(root.Balance);

break;

case "D":

var node = root?.Search(x);

if (node != null) {

root = node.Remove();

}

Console.WriteLine(root?.Balance ?? 0);

break;

case "C":

Console.WriteLine(root?.Search(x) != null ? "Y" : "N");

break;

}

}

}

private static (string Command, int Value) ReadCommand() {

var a = ReadStringList();

return (a[0], int.Parse(a[1]));

}

private static string[] ReadStringList() {

return Console.ReadLine()

.Split(' ')

.ToArray();

}

private static void SetupIO() {

\_in = new StreamReader("input.txt");

\_out = new StreamWriter("output.txt");

Console.SetIn(\_in);

Console.SetOut(\_out);

}

private static void DisposeIO() {

\_in?.Dispose();

\_out?.Dispose();

}

}

}