**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**

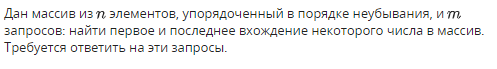
по лабораторной работе №6

Студент Кочарян С.А. группы P3218

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

2019

## Двоичный поиск



#include "edx-io.hpp"

#include <fstream>

#include <string>

#include <iostream>

using namespace std;

int first(long arr[], int low, int high, long x, int n)

{

if (high >= low)

{

int mid = low + (high - low) / 2;

if ((mid == 0 || x > arr[mid - 1]) && arr[mid] == x)

return mid;

else if (x > arr[mid])

return first(arr, (mid + 1), high, x, n);

else

return first(arr, low, (mid - 1), x, n);

}

return -1;

}

int last(long arr[], int low, int high, long x, int n)

{

if (high >= low)

{

int mid = low + (high - low) / 2;

if ((mid == n - 1 || x < arr[mid + 1]) && arr[mid] == x)

return mid;

else if (x < arr[mid])

return last(arr, low, (mid - 1), x, n);

else

return last(arr, (mid + 1), high, x, n);

}

return -1;

}

int main()

{

int elements\_quantity;

io >> elements\_quantity;

long \*elements = new long[elements\_quantity];

for (int i = 0; i < elements\_quantity; i++) {

io >> elements[i];

}

int commands\_quantity;

io >> commands\_quantity;

long curr\_elem;

for (int i = 0; i < commands\_quantity; i++) {

io >> curr\_elem;

int first\_occurance = first(elements, 0, elements\_quantity - 1, curr\_elem, elements\_quantity);

if (first\_occurance != -1)

first\_occurance++;

int last\_occurance = last(elements, 0, elements\_quantity - 1, curr\_elem, elements\_quantity);

if (last\_occurance != -1)

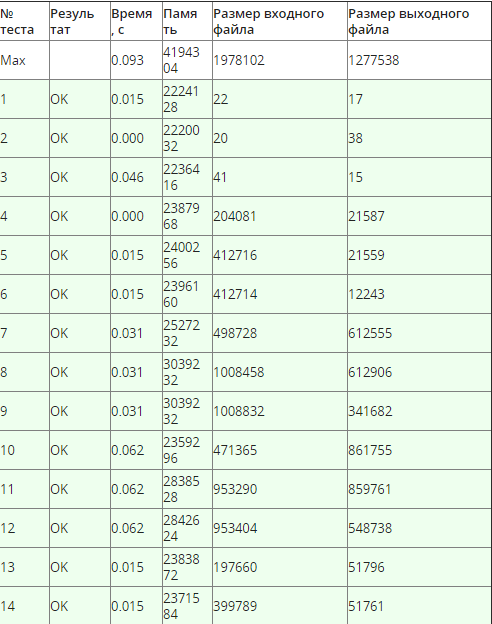
last\_occurance++;

io << first\_occurance << " " << last\_occurance << "\n";

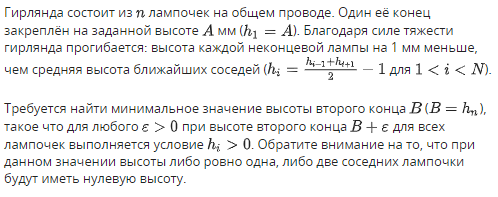
}

return 0;

}



## Гирлянда



#include <string>

using namespace std;

int main() {

FILE \*in;

in = fopen("input.txt","r");

int n;

fscanf(in,"%i",&n);

auto \*garland = new double[n];

fscanf(in,"%lf",&(garland[0]));

fclose(in);

const double precision = 1e-10;

double lowestEdge = 0;

double highestEdge = garland[0];

while (highestEdge - lowestEdge > precision){

garland[1] = (lowestEdge + highestEdge) / 2;

int i = 2;

do{

double value = 2 \* garland[i-1] - garland[i-2] + 2;

garland[i] = value;

i++;

} while (garland[i - 1] >= precision && i < n);

if (i == n && garland[i-1] >= 0){

highestEdge = garland[1];

} else {

lowestEdge = garland[1];

}

}

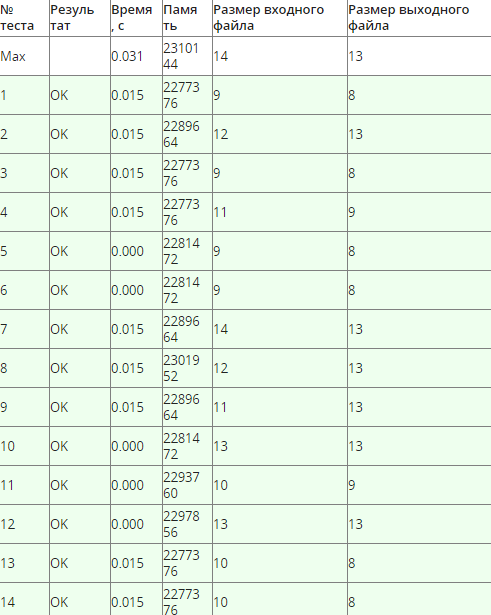
FILE \*out;

out = fopen("output.txt","w");

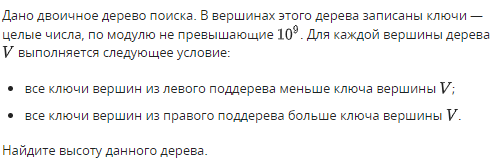
fprintf(out,"%lf", garland[n-1]);

fclose(out);

return 0;

}

## Высота дерева



#include <iostream>

#include <string>

#include <queue>

#include <deque>

#include "edx-io.hpp"

using namespace std;

struct t\_node {

int left, right;

} \*tree;

int depth(int i) {

int d = 1;

if (tree[i].left) {

d = max(depth(tree[i].left - 1) + 1, d);

}

if (tree[i].right) {

d = max(depth(tree[i].right - 1) + 1, d);

}

return d;

}

int main() {

int n, k, l, r;

io >> n;

if (n) {

tree = new t\_node[n];

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

io >> k >> tree[i].left >> tree[i].right;

}

io << depth(0);

}

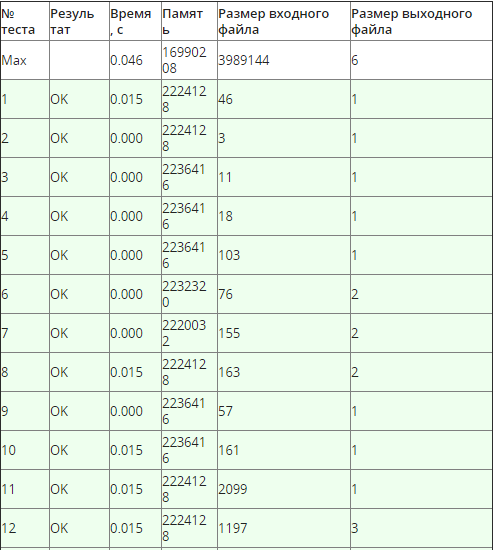
else {

io << 0;

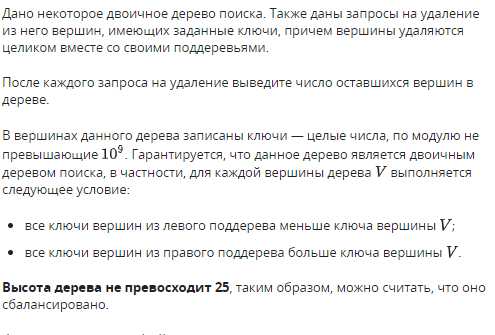
}

return 0;

}}



## Удаление поддеревьев



#include "edx-io.hpp"

#include <fstream>

#include <string>

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <algorithm>

using namespace std;

typedef struct Node{

Node\* left;

Node\* right;

Node\* parent;

int value;

} Node;

int getCount(Node\* node){

int depth = 1;

int depthLeft = 0;

if (node->left != nullptr) depthLeft = getCount(node->left);

int depthRight = 0;

if (node->right != nullptr) depthRight = getCount(node->right);

return depth + depthLeft + depthRight;

}

int deleteNode(Node \*node, int value, bool isRight){

if (node->value == value){

if (isRight){

node->parent->right = nullptr;

} else {

node->parent->left = nullptr;

}

return getCount(node);

} else{

if (value < node->value){

if (node->left != nullptr) return deleteNode(node->left,value, false);

} else {

if (node->right != nullptr) return deleteNode(node->right,value, true);

}

}

return 0;

}

int main() {

int n;

io >> n;

Node \*nodes = new Node[n];

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

nodes[i].right = nullptr;

nodes[i].left = nullptr;

nodes[i].parent = nullptr;

}

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

int k,l,r;

io >> k >> l >> r;

nodes[i].value = k;

if (l != 0){

nodes[i].left = &(nodes[l - 1]);

nodes[l-1].parent = &(nodes[i]);

}

if (r != 0){

nodes[i].right = &(nodes[r - 1]);

nodes[r-1].parent = &(nodes[i]);

}

}

int n2;

io >> n2;

int count = n;

for (int i = 0; i < n2; ++i) {

int value;

io >> value;

count -= deleteNode(nodes,value,true);

io << count << '\n';

}

return 0;

}

