Санкт-Петербургский Национальный Исследовательский
Университет Информационных Технологий, Механики и Оптики
Мегафакультет компьютерных технологий и управления

Дисциплина
«Алгоритмы и структуры данных»

Лабораторная работа №6

Выполнил:
Студент группы P3218
Рябов Сергей Витальевич
Преподаватель:
Муромцев Дмитрий Ильич

Санкт-Петербург,
2018

1. Двоичный поиск

Дан массив из n элементов, упорядоченный в порядке неубывания, и m запросов: найти первое и последнее вхождение некоторого числа в массив. Требуется ответить на эти запросы.

Формат входного файла

В первой строке входного файла содержится одно число n — размер массива ($1 \le n \le 10^5$). Во второй строке находятся n чисел в порядке неубывания — элементы массива. В третьей строке находится число m — число запросов ($1 \le m \le 10^5$). В следующей строке находятся m чисел — запросы. Элементы массива и запросы являются целыми числами, неотрицательны и не превышают 10^9.

Формат выходного файла

Для каждого запроса выведите в отдельной строке номер (индекс) первого и последнего вхождения этого числа в массив. Если числа в массиве нет, выведите два раза −1.

Исходный код (C#):

```
using System;
using System.IO;

namespace ADS.Week6
{
    public class Task1
    {
        public static int BinarySearch(int[] array, int value, int left, int right)
        {
            left--;
            while (left != right - 1)
            {
                int middle = (right + left) / 2;
                if (value > array[middle])
                {
                    left = middle;
                }
                else
                {
                    right = middle;
                }
            }
            return right;
        }

        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                int n = int.Parse(streamReader.ReadLine());
                int[] array = new int[n];
                string[] str = streamReader.ReadLine().Split(' ');
                for (int i = 0; i < n; i++)
                {
                    array[i] = int.Parse(str[i]);
                }
                int m = int.Parse(streamReader.ReadLine());
                str = streamReader.ReadLine().Split(' ');
                for (int i = 0; i < m; i++)
                {
                    int value = int.Parse(str[i]);
                    int leftIndex = BinarySearch(array, value, 0, array.Length - 1);
```

```
                    if (array[leftIndex] == value)
                    {
                        int rightIndex = BinarySearch(array, value + 1, leftIndex, array.Length - 1);
                        if (array[rightIndex] != value)
                        {
                            rightIndex--;
                        }
                        streamWriter.WriteLine("{0} {1}", leftIndex + 1, rightIndex + 1);
                    }
                    else
                    {
                        streamWriter.WriteLine("-1 -1");
                    }
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.203	38109184	1978102	1277538
1	OK	0.015	10158080	22	17
2	OK	0.031	10113024	20	38
3	OK	0.031	10137600	41	15
4	OK	0.031	13934592	204081	21587
5	OK	0.031	14311424	412716	21559
6	OK	0.046	13946880	412714	12243
7	OK	0.093	20713472	498728	612555
8	OK	0.125	26861568	1008458	612906
9	OK	0.062	22097920	1008832	341682
10	OK	0.109	22249472	471365	861755
11	OK	0.109	25038848	953290	859761
12	OK	0.093	21700608	953404	548738
13	OK	0.031	14442496	197660	51796
14	OK	0.046	14655488	399789	51761
15	OK	0.031	13897728	399826	29610
16	OK	0.125	22827008	511344	947660
17	OK	0.125	25837568	1034328	951787
18	OK	0.062	22986752	1034511	608920

2. Гирлянда

Гирлянда состоит из n лампочек на общем проводе. Один её конец закреплён на заданной высоте A мм (h1=A). Благодаря силе тяжести гирлянда прогибается: высота каждой неконцевой лампы на 1 мм меньше, чем средняя высота ближайших соседей ($h_i=h_i-1+h_i+12-1$ для $1<i<N$).

Требуется найти минимальное значение высоты второго конца B ($B=h_n$), такое что для любого $\varepsilon>0$ при высоте второго конца $B+\varepsilon$ для всех лампочек выполняется условие $h_i>0$. Обратите внимание на

то, что при данном значении высоты либо ровно одна, либо две соседних лампочки будут иметь нулевую высоту.

Формат входного файла

В первой строке входного файла содержится два числа n и A ($3 \le n \le 1000$, n — целое, $10 \le A \le 1000$, A — вещественное и дано не более чем с тремя знаками после десятичной точки).

Формат выходного файла

Выведите одно вещественное число B — минимальную высоту второго конца. Ваш ответ будет засчитан, если он будет отличаться от правильного не более, чем на 10^{-6}.

Исходный код (C#):

```
using System;
using System.IO;
using System.Globalization;

namespace ADS.Week6
{
    public class Task2
    {
        private const double precision = 0.000001d;
        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                string[] str = streamReader.ReadLine().Split(' ');
                int n = int.Parse(str[0]);
                double first = double.Parse(str[1], CultureInfo.InvariantCulture);
                double min = 0;
                double max = first;
                double answer = 0d;
                bool precisionReached = false;
                while (!precisionReached)
                {
                    double second = (max + min) / 2d;
                    double a = first;
                    double b = second;
                    bool result = true;

                    for (int i = 2; i < n; i++)
                    {
                        double c = 2 * b - a + 2;
                        a = b;
                        b = c;
                        if (c < 0d)
                            result = false;
                    }
                    if (result)
                        max = second;
                    else
                        min = second;
                    if (Math.Abs(answer - b) <= precision)
                    {
                        precisionReached = true;
                    }
                    answer = b;
                }
                streamWriter.WriteLine(answer);
            }
```

```
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.078	10326016	14	23
1	OK	0.031	10256384	9	18
2	OK	0.015	10182656	12	18
3	OK	0.031	10178560	9	22
4	OK	0.015	10276864	11	21
5	OK	0.046	10248192	9	3
6	OK	0.015	10170368	9	18
7	OK	0.031	10166272	14	18
8	OK	0.015	10211328	12	18
9	OK	0.031	10207232	11	18
10	OK	0.015	10264576	13	18
11	OK	0.031	10231808	10	22
12	OK	0.031	10182656	13	18
13	OK	0.015	10231808	10	22
14	OK	0.015	10211328	10	22
15	OK	0.015	10240000	12	18
16	OK	0.015	10158080	9	18
17	OK	0.031	10158080	12	18
18	OK	0.031	10170368	12	18

3. Высота дерева

Высотой дерева называется максимальное число вершин дерева в цепочке, начинающейся в корне дерева, заканчивающейся в одном из его листьев, и не содержащей никакую вершину дважды.

Так, высота дерева, состоящего из единственной вершины, равна единице. Высота пустого дерева (да, бывает и такое!) равна нулю. Высота дерева, изображенного на рисунке, равна четырем.

Дано двоичное дерево поиска. В вершинах этого дерева записаны ключи — целые числа, по модулю не превышающие 10^9. Для каждой вершины дерева V выполняется следующее условие:

- все ключи вершин из левого поддерева меньше ключа вершины V;
- все ключи вершин из правого поддерева больше ключа вершины V.

Найдите высоту данного дерева.

Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число N ($0 \le N \le 2 \cdot 10^5$) — число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В (i+1)-ой строке файла ($1 \le i \le N$) находится описание i-ой вершины, состоящее из трех чисел K_i, L_i, R_i, разделенных пробелами — ключа в i-ой вершине ($|K_i| \le 10^9$),

номера левого ребенка i-ой вершины ($i<L_i \le N$ или $L_i=0$, если левого ребенка нет) и номера правого ребенка i-ой вершины ($i<R_i \le N$ или $R_i=0$, если правого ребенка нет).

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

Формат выходного файла

Выведите одно целое число — высоту дерева.

Исходный код (C#):

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Globalization;

namespace ADS.Week6
{

    public class BinaryTree
    {
        private struct Node
        {
            public int Key { get; set; }
            public int Left { get; set; }
            public int Right { get; set; }

            public Node(int key, int left, int right)
            {
                Key = key;
                Left = left;
                Right = right;
            }
        }
        private Node[] nodes;
        private int root;
        private int size;

        public BinaryTree(int size)
        {
            nodes = new Node[size];
            root = 0;
            size = 0;
        }
        public void Add(int key, int left, int right)
        {
            nodes[size] = new Node(key, left, right);
            if (nodes[size].Left == root || nodes[size].Right == root)
                root = size;
            size++;
        }

        public int GetDepth()
        {
            List<int> childs = new List<int>(nodes.Length);
            List<int> newChilds = new List<int>(nodes.Length);
            newChilds.Add(root);
            int depth = 0;
            while (newChilds.Count > 0)
            {
                List<int> temp = childs;
                childs = newChilds;
                newChilds = temp;
```

```
                newChilds.Clear();
                for (int i = 0; i < childs.Count; i++)
                {
                    if (nodes[childs[i]].Left != -1)
                        newChilds.Add(nodes[childs[i]].Left);
                    if (nodes[childs[i]].Right != -1)
                        newChilds.Add(nodes[childs[i]].Right);
                }
                depth++;
            }
            return depth;
        }
    }
    public class Task3
    {
        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                int n = int.Parse(streamReader.ReadLine());
                BinaryTree tree = new BinaryTree(n);
                for (int i = 0; i < n; i++)
                {
                    string[] str = streamReader.ReadLine().Split(' ');
                    tree.Add(int.Parse(str[0]), int.Parse(str[1]) - 1, int.Parse(str[2]) - 1);
                }
                streamWriter.WriteLine(n == 0 ? 0 : tree.GetDepth());
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.234	14045184	3989144	8
1	OK	0.031	10252288	46	3
2	OK	0.015	10104832	3	3
3	OK	0.015	10276864	11	3
4	OK	0.031	10240000	18	3
5	OK	0.031	10215424	103	3
6	OK	0.031	10309632	76	4
7	OK	0.031	10223616	155	4
8	OK	0.046	10231808	163	4
9	OK	0.031	10330112	57	3
10	OK	0.015	10280960	161	3
11	OK	0.031	10289152	2099	3
12	OK	0.031	10301440	1197	5
13	OK	0.031	10289152	2073	5
14	OK	0.031	10321920	2139	5
15	OK	0.046	10256384	686	3

4. Знакомство с жителями Сортлэнда

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими подданными. Число жителей в графстве нечетно и составляет n, где n может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем.

Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до n. Информация о размере денежных накоплений жителей хранится в массиве M таким образом, что сумма денежных накоплений жителя, обладающего идентификационным номером i, содержится в ячейке M[i]. Помогите секретарю графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

Формат входного файла

Первая строка входного файла содержит число жителей n ($3 \le n \le 9999$, n нечетно). Вторая строка содержит описание массива M, состоящее из n положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива M различны, а их значения имеют точность не более двух знаков после запятой и не превышают 106.

Формат выходного файла

В выходной файл выведите три целых положительных числа, разделенных пробелами — идентификационные номера беднейшего, среднего и самого богатого жителей Сортлэнда.

Исходный код (C#):

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Globalization;

namespace ADS.Week6
{

    public class BinaryTree
    {
        private struct Node
        {
            public int Key { get; set; }
            public int Left { get; set; }
            public int Right { get; set; }
            public bool Exists { get; set; }

            public Node(int key, int left, int right)
            {
                Key = key;
                Left = left;
                Right = right;
                Exists = true;
            }
        }
        private Node[] nodes;
        private int root;
        private int capacity;
```

```
List<int> childs;
List<int> newChilds;

public int Size { get; private set; }

public BinaryTree(int capacity)
{
    nodes = new Node[capacity];
    childs = new List<int>(0);
    newChilds = new List<int>(0);
}
public void Add(int key, int left, int right)
{
    nodes[capacity] = new Node(key, left, right);
    if (nodes[capacity].Left == root || nodes[capacity].Right == root)
        root = capacity;
    capacity++;
    Size++;
}

public int GetSize(int root)
{
    childs.Clear();
    newChilds.Clear();
    newChilds.Add(root);
    int count = 0;
    while (newChilds.Count > 0)
    {
        count += newChilds.Count;
        List<int> temp = childs;
        childs = newChilds;
        newChilds = temp;
        newChilds.Clear();
        for (int i = 0; i < childs.Count; i++)
        {
            if (nodes[childs[i]].Left != -1 && nodes[nodes[childs[i]].Left].Exists)
                newChilds.Add(nodes[childs[i]].Left);
            if (nodes[childs[i]].Right != -1 && nodes[nodes[childs[i]].Right].Exists)
                newChilds.Add(nodes[childs[i]].Right);
        }
    }
    return count;
}
public void Delete(int key)
{
    int node = Find(key);
    if (node != -1)
    {
        Size -= GetSize(node);
        nodes[node].Exists = false;
    }
}
public int Find(int key)
{
    int node = root;
    while (node != -1 && nodes[node].Key != key && nodes[node].Exists)
    {
        if (key <= nodes[node].Key)
            node = nodes[node].Left;
        else
            node = nodes[node].Right;
    }
    if (node != -1 && (nodes[node].Key != key || !nodes[node].Exists))
```

```
                node = -1;
            return node;
        }
    }
    public class Task4
    {
        public static void Main(string[] args)
        {
            using (StreamReader streamReader = new StreamReader("input.txt"))
            using (StreamWriter streamWriter = new StreamWriter("output.txt"))
            {
                int n = int.Parse(streamReader.ReadLine());
                BinaryTree tree = new BinaryTree(n);
                string[] str;
                for (int i = 0; i < n; i++)
                {
                    str = streamReader.ReadLine().Split(' ');
                    tree.Add(int.Parse(str[0]), int.Parse(str[1]) - 1, int.Parse(str[2]) - 1);
                }
                int m = int.Parse(streamReader.ReadLine());
                str = streamReader.ReadLine().Split(' ');
                for (int i = 0; i < m; i++)
                {
                    tree.Delete(int.Parse(str[i]));
                    streamWriter.WriteLine(tree.Size);
                }
            }
        }
    }
}
```

Результат:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.312	38653952	6029382	1077960
1	OK	0.031	10285056	58	12
2	OK	0.031	10178560	27	12
3	OK	0.031	10301440	34	15
4	OK	0.015	10362880	211	30
5	OK	0.031	10334208	246	30
6	OK	0.031	10358784	3437	457
7	OK	0.031	10391552	3363	483
8	OK	0.031	10665984	18842	4247
9	OK	0.031	10817536	25683	3739
10	OK	0.015	11485184	69351	14791
11	OK	0.046	11505664	88936	11629
12	OK	0.031	11874304	244892	40297
13	OK	0.078	11960320	255614	37596
14	OK	0.078	15974400	978616	141281
15	OK	0.078	15638528	992647	137802