Министерство образования и науки Российской Федерации
САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Дисциплина «**Алгоритмы и структуры данных**»

ОТЧЁТ

по лабораторным работам за 6 неделю

Студент **Пастухов К.А. группы P3218**

Санкт-Петербург
2018 г.

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.

Выполнение

```
def main():
    results = {}
    with open('input.txt', "r") as file:
        _ = file.readline()
        array = [int(x) for x in file.readline().strip().split(" ")]
        _ = file.readline()
        requests = [int(x) for x in file.readline().strip().split(" ")]
        res = open('output.txt', 'w')
        for i in range(0, len(requests)):
            if requests[i] in results:
                res.write(results[requests[i]] + '\n')
            else:
                binary_search(array, requests[i], results)
                res.write(results[requests[i]] + '\n')


def binary_search(array, value, results):
    l = -1
    r = len(array)
    while (r > l+1 or (r >= len(array) and l < 0 and array[l] != value and
array[r] != value)):
        mid = (l+r) / 2
        if array[int(mid)] < value:
            l = mid
        else:
            r = mid

    if r < len(array) and array[int(r)] == value:
        while r < len(array) and array[int(r)] == value:
            r += 1
        while l >= 0 and array[int(l)] == value:
            l -= 1
        l += 2
        results[value] = (f"{int(l)} {int(r)}")

    else:
        results[value] = "-1 -1"


if __name__ == "__main__":
    main()
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.312	26415104	1978102	1277538
1	OK	0.031	9089024	22	17
2	OK	0.031	9179136	20	38
3	OK	0.031	9068544	41	15
4	OK	0.078	13762560	204081	21587
5	OK	0.093	14086144	412716	21559
6	OK	0.078	14172160	412714	12243
7	OK	0.328	17178624	498728	612555
8	OK	0.328	18309120	1008458	612906
9	OK	0.640	18497536	1008832	341682
10	OK	0.375	18833408	471365	861755
11	OK	0.375	20406272	953290	859761
12	OK	0.906	20393984	953404	548738
13	OK	0.109	13434880	197660	51796
14	OK	0.109	13828096	399789	51761
15	OK	0.109	13729792	399826	29610
16	OK	0.375	19628032	511344	947660
17	OK	0.390	21147648	1034328	951787
18	OK	1.031	21323776	1034511	608920
19	OK	0.218	15306752	384717	274370
20	OK	0.265	16097280	777782	274601
21	OK	0.312	16121856	778270	152655
22	OK	0.187	12300288	219786	228823
23	OK	0.203	12759040	444845	228627
24	OK	0.281	12898304	444580	136297
25	OK	0.171	19009536	452007	84006
26	OK	0.203	19783680	914248	84077
27	OK	0.156	19865600	914384	46178
28	OK	0.250	20017152	534373	224808
29	OK	0.265	20451328	1080911	225002
30	OK	0.312	20451328	1080929	123417

Гирлянда

Гирлянда состоит из n лампочек на общем проводе. Один её конец закреплён на заданной высоте A мм ($h_1 = A$). Благодаря силе тяжести гирлянда прогибается: высота каждой неконцевой лампы на 1 мм меньше, чем средняя высота ближайших соседей ($h_i = \frac{h_{i-1}+h_{i+1}}{2} - 1$ 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.

Выполнение

```
def main():
    with open("input.txt", "r") as file:
        text = [float(x) for x in file.readline().strip().split(" ")]
        print(text)
        n = text[0]
        a = text[1]
```

```
        print(find_min_height(n, 0, a))
        with open("output.txt", "w") as res_file:
            res_file.write(str(find_min_height(n, 0, a)))


def find_min_height(n, left, right):
    last = -1
    left_height = right

    while ((right-left) > 0.00000000001):
        mid = (left+right) / 2
        prev = left_height
        current = mid
        above_ground = current > 0
        for i in range(3, int(n)+1):
            next = 2 * current - prev + 2
            above_ground &= next > 0
            prev = current
            current = next

        if above_ground:
            right = mid
            last = current
        else:
            left = mid
    return last


if __name__ == "__main__":
    main()
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.093	9162752	14	22
1	OK	0.031	9007104	9	15
2	OK	0.046	9048064	12	18
3	OK	0.031	8982528	9	22
4	OK	0.046	9035776	11	21
5	OK	0.031	9105408	9	22
6	OK	0.031	9048064	9	17
7	OK	0.062	9076736	14	17
8	OK	0.062	8998912	12	16
9	OK	0.046	9035776	11	18
10	OK	0.046	9015296	13	17
11	OK	0.046	9158656	10	22
12	OK	0.062	9072640	13	17
13	OK	0.046	9089024	10	21
14	OK	0.031	9019392	10	21
15	OK	0.062	9048064	12	17
16	OK	0.031	9007104	9	18
17	OK	0.078	9064448	12	17
18	OK	0.046	9052160	12	17
19	OK	0.046	9060352	12	18
20	OK	0.046	9027584	11	16
21	OK	0.046	9003008	11	18
22	OK	0.046	9003008	11	17
23	OK	0.031	9105408	11	22
24	OK	0.031	9052160	11	18
25	OK	0.062	9064448	12	17
26	OK	0.046	9003008	12	17
27	OK	0.062	9056256	12	17
28	OK	0.062	9015296	12	16
29	OK	0.078	9031680	12	17
30	OK	0.046	8994816	11	21
31	OK	0.046	9023488	12	17

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.

Выполнение

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

namespace Openedu.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 week6_lab3
{
    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());
            }
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.375	45043712	3989144	8
1	OK	0.031	11284480	46	3
2	OK	0.015	10170368	3	1
3	OK	0.015	11251712	11	3
4	OK	0.031	11329536	18	3
5	OK	0.031	11329536	103	3
6	OK	0.031	11259904	76	4
7	OK	0.031	11280384	155	4
8	OK	0.031	11341824	163	4
9	OK	0.062	11345920	57	3
10	OK	0.031	11272192	161	3
11	OK	0.046	11386880	2099	3
12	OK	0.062	11427840	1197	5
13	OK	0.031	11194368	2073	5
14	OK	0.031	11251712	2139	5
15	OK	0.031	11071488	686	3
16	OK	0.031	11120640	2128	4
17	OK	0.015	11329536	8777	3
18	OK	0.031	11554816	10426	5
19	OK	0.031	11546624	16336	5
20	OK	0.031	11591680	16835	5
21	OK	0.031	11194368	3520	3
22	OK	0.031	11591680	16969	4
23	OK	0.046	12292096	36534	4
24	OK	0.031	12304384	38820	6
25	OK	0.031	12283904	55707	6
26	OK	0.031	12251136	57235	6
27	OK	0.015	11309056	7784	4
28	OK	0.046	12222464	56607	4
29	OK	0.046	15839232	149518	4
30	OK	0.046	16027648	117171	6

Удаление поддеревьев

Дано некоторое двоичное дерево поиска. Также даны запросы на удаление из него вершин, имеющих заданные ключи, причем вершины удаляются целиком вместе со своими поддеревьями.

Выполнение

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

namespace Openedu.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 week6_lab4
    {
        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);
                }
            }
        }
    }
}
```

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.578	65306624	6029382	1077960
1	OK	0.031	11517952	58	12
2	OK	0.046	11415552	27	12
3	OK	0.062	11481088	34	15
4	OK	0.046	11505664	211	30
5	OK	0.062	11440128	246	30
6	OK	0.031	11538432	3437	457
7	OK	0.031	11526144	3363	483
8	OK	0.046	12025856	18842	4247
9	OK	0.031	12234752	25683	3739
10	OK	0.046	12685312	69351	14791
11	OK	0.046	13250560	88936	11629
12	OK	0.062	17944576	244892	40297
13	OK	0.062	18178048	255614	37596
14	OK	0.109	22224896	978616	141281
15	OK	0.109	22310912	992647	137802
16	OK	0.234	37351424	2488583	634135
17	OK	0.312	47480832	3489729	483105
18	OK	0.406	56098816	4639039	1077960
19	OK	0.578	65101824	6007604	931260
20	OK	0.546	65306624	6029382	916969