Санкт-Петербургский Научный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Задачи шестой недели

по курсу «Алгоритмы и структуры данных»

на Openedu

Выполнил: студент группы P3218

Артамонов Александр Владимирович

Санкт-Петербург, 2019г

Задача 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.

Пример

input.txt	output.txt
5 1 1 2 2 2 3 1 2 3	1 2 3 5 -1 -1

Код программы (C++)

```
//Находит элемент, приближая правую границу к левой
long findEl(long* btree, long left, long right, long element, long size) {
	left--;
	right++;
	long middle;
	while (right > left + 1)
	{
		middle = (left + right) / 2;
		if (btree[middle] < element) {
			left = middle;
		}
		else {
			right = middle;
		}

	}
	if (right < size && btree[right] == element) {
		return right;
	}
	else {
		return -1;
	}
}
//Находит элемент приближая левую границу к правой
long findRight(long* btree, long left, long right, long element, long size) {
	left--;
	right++;
	long middle;
	while (right > left + 1)
	{
		middle = (left + right) / 2;
		if (btree[middle] <= element) {
			left = middle;
		}
		else {
			right = middle;
		}

	}
	if (left < size && btree[left] == element) {
		return left;
	}
	else {
		return -1;
	}
}


int main() {
	long N, M;

	//Вводим двоичное дерево
	io >> N;
	long* btree = new long[N];
	for (long i = 0; i < N; i++)
	{
		io >> btree[i];
	}
	//Вводим массив искомых чисел
	io >> M;
	long* queries = new long[M];
	for (long i = 0; i < M; i++)
```

```
        {
                io >> queries[i];
        }

        long cur_query, first, last, pos;

        //Для каждого числа из массива осуществляем поиск
        for (long i = 0; i < M; i++)
        {
                cur_query = queries[i];
                //Находим первый попавшийся удовлетворяющий элемент
                pos = findEl(btree, 0, N - 1, cur_query, N);
                last = first = pos;
                //Если такого элемента нет, просто выводим -1 -1
                if (pos != -1) {
                        //Если есть ищем первое появление с приближением границы справа
                        first = findEl(btree, 0, pos, cur_query, N);
                        //А последнее с приближение границы слева
                        last = findRight(btree, pos + 1, N - 1, cur_query, N);
                        first++;
                        last++;
                }

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

        return 0;
}
```

Бенчмарк (задача 1)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.078	4591616	1978102	1277538
1	OK	0.000	2224128	22	17

Задача 2. 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 ($h_i = (h_{i-1} + h_{i+1})/2 - 1$ 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 + ε 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}.

Примеры

input.txt	output.txt
8 15	9.75
692 532.81	446113.34434782615

Код программы (C++)

```
int main() {
	long N;
	double A;
	io >> N >> A;

	//Количество лампочек на минимальном уровне
	int zero = 1;

	//Находим количество лампочек слева от минимального уровня решив уравнение i(i+1)
= A, i(i+1) - верхняя граница для i лампочек
	double i = (-1 + sqrt(1 + 4 * A)) / 2;

	//Если высота = верхней границе, значит на минимальном уровне будет две лапочки
	if (i == ceil(i)) {
		zero++;
	}
```

```
	long H = ceil(i);

	//Для определённой высоты можно по количеству лампочек подсчитать первый шаг от минимальной лампочки по формуле
	//Высота = кол-во лампочек * (шаг от минимальной лампочки + (кол-во лампочек - 1))
	//Выразим из неё шаг
	double x = (A - H * (H - 1)) / H;

	//Сумма элементов вокруг 0 по формуле hi = (h1+h2)/2 -1 должна быть равна 2
	double y = 2 - x;

	//Следовательно, если y = 0? значит у нас 2 нуля и у надо сместить
	if (y == 0) {
		y = 2;
	}

	//Количество лампочек справа равно общему количеству - кол-во слева - кол-во нулей
	long j = N - H - zero;
	double B;

	//Если справа лампочек нет - значит B = 0
	if (j <= 0) {
		B = 0;
	}
	else {
		B = j * y + j * (j - 1);
	}

	io << B;

	return 0;
}}
```

Бенчмарк (задача 2)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.031	2277376	14	18
1	OK	0.000	2260992	9	4

Задача 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 Ki,Li,Ri, разделенных пробелами — ключа в i-ой вершине ($|Ki| \le 10^9$), номера левого ребенка i-ой вершины ($i < Li \le N$ или Li=0, если левого ребенка нет) и номера правого ребенка i-ой вершины ($i < Ri \le N$ или Ri=0, если правого ребенка нет). Все ключи различны.

Формат выходного файла

Выведите одно целое число — высоту дерева.

Пример

input.txt	output.txt
6 -2 0 2 8 4 3 9 0 0 3 6 5 6 0 0 0 0 0	4

Код программы (C++)

```
//Узел хранит позиции левого и правого узлов-потомков
struct node {
	int left, right;
};

//Возвращает максимальное из двух чисел
long max(long a, long b) {
	if (a > b) {
		return a;
	}
	else {
		return b;
	}
}

//Находит высоту i-того элемента дерева tree
long depth(node* tree, long i) {
	long d = 0;
	//Если потомок существует, то находим его высоту
	if (tree[i].left != 0) {
		d = max(depth(tree, tree[i].left), d);
	}
	if (tree[i].right != 0) {
		d = max(depth(tree, tree[i].right), d);
	}
	//Возвращаем максимальную высоту потомков + 1
	return ++d;
}

int main() {
	long N, K;

	io >> N;
	if (N != 0) {
		//Создаём массив узлов
		node* tree = new node[N + 1];

		for (long i = 1; i < N + 1; i++)
		{
			io >> K >> tree[i].left >> tree[i].right;
		}
		//Находим высоту 1 узла (корня)
		io << depth(tree, 1);
	}
	else
	{
		io << 0;
	}

	return 0;
}
```

Бенчмарк (задача 3)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.046	16990208	3989144	6
1	OK	0.000	2220032	46	1

Задача 4. 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.

После каждого запроса на удаление выведите число оставшихся вершин в дереве.

В вершинах данного дерева записаны ключи — целые числа, по модулю не превышающие 10^9. Гарантируется, что данное дерево является двоичным деревом поиска, в частности, 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.

Высота дерева не превосходит 25, таким образом, можно считать, что оно сбалансировано.

Формат входного файла

Входной файл содержит описание двоичного дерева и описание запросов на удаление.

В первой строке файла находится число N ($1 \le N \le 2 \cdot 10^5$) — 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 Ki, Li, Ri, разделенных пробелами — ключа в i-ой вершине ($|Ki| \le 10^9$), номера левого ребенка i-ой вершины ($i < Li \le N$ или Li=0, если левого ребенка нет) и номера правого ребенка i-ой вершины ($i < Ri \le N$ или Ri=0, если правого ребенка нет).

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

В следующей строке находится число M ($1 \le M \le 2 \cdot 10^5$) — число запросов на удаление. В следующей строке находятся M чисел, разделенных пробелами — ключи, вершины с которыми (вместе с их поддеревьями) необходимо удалить. Все эти числа не превосходят 10^9 по абсолютному значению. Вершина с таким ключом не обязана существовать в дереве — в этом случае дерево изменять не требуется. Гарантируется, что корень дерева никогда не будет удален.

Формат выходного файла

Выведите M строк. На i-ой строке требуется вывести число вершин, оставшихся в дереве после выполнения i-го запроса на удаление.

Пример

input.txt	output.txt
6 -2 0 2 8 4 3 9 0 0 3 6 5 6 0 0 0 0 0 4 6 9 7 8	5 4 4 1

Код программы (C++)

```
struct node {
	long key, left, right;
};
//Определяет количество узлов в поддереве узла
long depth(node* tree, long i) {
	long d = 0;
	if (tree[i].left != 0) {
		d += depth(tree, tree[i].left);
	}
	if (tree[i].right != 0) {
		d += depth(tree, tree[i].right);
	}
	return ++d;
}
//Находит узел по значению
long find_node(node* tree, long del_value) {
	long i = 1;
	while (tree[i].key != del_value) {
		if (del_value < tree[i].key) {
			if (tree[i].left) {
				i = tree[i].left;
			}
			else { return 0;}
		}
		else {
			if (tree[i].right) {
				i = tree[i].right;
			}
			else { return 0;}
		}
	}

	return i;
}
```

```
//Удаляет узел и его поддерево по значению и возвращает количество элементов в дереве
long delete_node(node* tree, long del_value, long size, long* parents) {
	//Находим узел, который необходимо удалить
	long pos = find_node(tree, del_value);
	if (pos == 0) {
		return size;
	}
	else {
		long parent_pos = parents[pos];
		//Если узел является левым потомком, обнуляем левого потомка родителя
		if (tree[parent_pos].left == pos) {
			tree[parent_pos].left = 0;
		}
		//Если нет, то правого вотомка
		else {
			tree[parent_pos].right = 0;
		}

		return size - depth(tree, pos);
	}
}

int main() {
	long N, M;
	io >> N;
	node* tree = new node[N + 1];
	long* parents = new long[N + 1];

	for (long i = 1; i < N + 1; i++)
	{
		io >> tree[i].key >> tree[i].left >> tree[i].right;
		//При наличии левого и правого потомков, заносим данные об их родителе в
массив parents
		if (tree[i].left != 0) {
			parents[tree[i].left] = i;
		}
		if (tree[i].right != 0) {
			parents[tree[i].right] = i;
		}
	}
	io >> M;

	long del_value;

	for (long i = 0; i < M; i++)
	{
		io >> del_value;
		//Для каждого вводимого значения вызываем функцию удаления элемента
		N = delete_node(tree, del_value, N, parents);
		io << N << "\n";

	}
	return 0;
}
```

Бенчмарк (задача 4)

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.093	11042816	6029382	1077960
1	OK	0.015	2220032	58	12