Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа № 7

по дисциплине «Алгоритмы и структуры данных»

Openedu – неделя 7

Подготовил:

студент группы P3217

Бураков Илья Алексеевич

Преподаватели:

Романов Алексей Андреевич

Волчек Дмитрий Геннадьевич

Санкт-Петербург, 2019

Проверка сбалансированности

Условие

АВЛ-дерево является сбалансированным в следующем смысле: для любой вершины высота ее левого поддерева отличается от высоты ее правого поддерева не больше, чем на единицу.

Введем понятие *баланса вершины*: для вершины дерева V ее баланс $B(V)$ равен разности высоты правого поддерева и высоты левого поддерева. Таким образом, свойство АВЛ-дерева, приведенное выше, можно сформулировать следующим образом: для любой ее вершины V выполняется следующее неравенство:

$$-1 \le B(V) \le 1$$

Обратите внимание, что, по историческим причинам, определение баланса в этой и последующих задачах этой недели "зеркально отражено" по сравнению с определением баланса в лекциях! Надеемся, что этот факт не доставит Вам неудобств. В литературе по алгоритмам — как российской, так и мировой — ситуация, как правило, примерно та же.

Дано двоичное дерево поиска. Для каждой его вершины требуется определить ее баланс.

Формат входного файла

Входной файл содержит описание двоичного дерева. 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 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

Для i-ой вершины в i-ой строке выведите одно число — баланс данной вершины.

Пример

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

Решение

openedu/week7/lab7_1.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <list>
```

```
#include <assert.h>
#include <tuple>

using namespace std;

typedef int32_t balance_t;

struct Node {
	int key;
	balance_t balance = 0;
	int height = 1;

	Node* parent = nullptr;
	Node* left = nullptr;
	Node* right = nullptr;

	balance_t get_left_balance() { return (left) ? left->balance : -1; };
	balance_t get_right_balance() { return (right) ? right->balance : -1; };

	int get_left_height() { return (left) ? left->height : -1; };
	int get_right_height() { return (right) ? right->height : -1; };
};

struct NodeDescr {
	int k;
	int li;
	int ri;

	// remember balance just for this task
	balance_t balance_result;
};

Node* build_tree(vector<NodeDescr> &input, int root_index = 0) {
	// root node description
	NodeDescr& rd = input[root_index];

	// create new node
	Node* tree = new Node();
	tree->key = rd.k;

	// recursively build left and right subtrees if needed
	if (rd.li) {
		tree->left = build_tree(input, rd.li - 1);
		tree->left->parent = tree;
	}

	if (rd.ri) {
		tree->right = build_tree(input, rd.ri - 1);
		tree->right->parent = tree;
	}

	// calculate height and balance
	tree->height = max(tree->get_right_height(), tree->get_left_height()) + 1;
	tree->balance = tree->get_right_height() - tree->get_left_height();

	// remember balance just for this task
	rd.balance_result = tree->balance;

	return tree;
}

Node* find_node(Node* tree, int key) {
	if (tree == nullptr) {
		return nullptr;
	} else if (key == tree->key) {
		return tree;
	} else if (key < tree->key) {
```

```
            return find_node(tree->left, key);
        } else {
            return find_node(tree->right, key);
        }
}

int main() {
    int n;
    io >> n;

    vector<NodeDescr> input(n);
    for (auto& descr : input) {
        io >> descr.k >> descr.li >> descr.ri;
    }

    auto tree = build_tree(input);

    for (auto& descr : input) {
        io << descr.balance_result << "\n";
    }

    return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.125	28200960	3986010	1688889
1	OK	0.015	2220032	46	19
2	OK	0.000	2224128	10	3
3	OK	0.000	2220032	17	6
4	OK	0.000	2224128	17	7
5	OK	0.000	2224128	24	9
6	OK	0.000	2232320	24	10
7	OK	0.000	2220032	24	9
8	OK	0.000	2220032	24	10
9	OK	0.015	2224128	24	11
10	OK	0.015	2228224	31	12
11	OK	0.015	2236416	31	13
12	OK	0.000	2240512	31	12
13	OK	0.000	2224128	31	13
14	OK	0.015	2224128	31	14
15	OK	0.000	2224128	31	12
16	OK	0.015	2220032	31	13
17	OK	0.000	2224128	31	13
18	OK	0.000	2232320	31	14
19	OK	0.000	2236416	31	13
20	OK	0.015	2236416	31	14
21	OK	0.031	2224128	31	13
22	OK	0.015	2224128	31	14

23	OK	0.015	2240512	31	15
24	OK	0.000	2236416	38	15
25	OK	0.015	2240512	38	16
26	OK	0.000	2236416	38	15
27	OK	0.015	2240512	38	16
28	OK	0.000	2236416	38	17
29	OK	0.000	2224128	38	15
30	OK	0.000	2240512	38	16
31	OK	0.000	2236416	38	16
32	OK	0.046	2236416	38	17
33	OK	0.015	2236416	38	16
34	OK	0.000	2224128	38	17
35	OK	0.000	2236416	38	16
36	OK	0.015	2232320	38	17
37	OK	0.000	2224128	38	18
38	OK	0.015	2224128	38	15
39	OK	0.000	2236416	38	16
40	OK	0.000	2240512	38	15
41	OK	0.015	2240512	38	16
42	OK	0.000	2224128	38	17
43	OK	0.000	2236416	38	15
44	OK	0.000	2220032	38	16
45	OK	0.000	2236416	38	16
46	OK	0.015	2236416	38	17
47	OK	0.015	2224128	38	16
48	OK	0.000	2232320	38	17
49	OK	0.015	2236416	38	16
50	OK	0.000	2228224	38	17
51	OK	0.000	2220032	38	18
52	OK	0.000	2224128	38	16
53	OK	0.000	2240512	38	17
54	OK	0.000	2236416	38	16
55	OK	0.000	2236416	38	17
56	OK	0.000	2236416	38	18
57	OK	0.000	2236416	38	16
58	OK	0.000	2220032	38	17
59	OK	0.000	2224128	38	17
60	OK	0.015	2240512	38	18
61	OK	0.000	2224128	38	17
62	OK	0.000	2232320	38	18
63	OK	0.000	2236416	38	17
64	OK	0.000	2220032	38	18

65	OK	0.000	2236416	38	19
66	OK	0.000	2236416	45	18
67	OK	0.000	2240512	45	19
68	OK	0.000	2224128	45	18
69	OK	0.015	2224128	45	19
70	OK	0.000	2236416	45	20
71	OK	0.000	2220032	45	18
72	OK	0.000	2224128	45	19
73	OK	0.015	2220032	45	19
74	OK	0.000	2220032	45	20
75	OK	0.015	2220032	45	19
76	OK	0.000	2240512	45	20
77	OK	0.000	2224128	45	19
78	OK	0.015	2236416	45	20
79	OK	0.015	2220032	45	21
80	OK	0.000	2240512	45	18
81	OK	0.000	2224128	45	19
82	OK	0.000	2224128	45	18
83	OK	0.015	2224128	45	19
84	OK	0.000	2220032	45	20
85	OK	0.000	2220032	45	18
86	OK	0.000	2236416	45	19
87	OK	0.015	2224128	45	19
88	OK	0.000	2224128	45	20
89	OK	0.000	2224128	45	19
90	OK	0.000	2220032	45	20
91	OK	0.000	2224128	45	19
92	OK	0.000	2236416	45	20
93	OK	0.015	2232320	45	21
94	OK	0.000	2228224	45	19
95	OK	0.000	2236416	45	20
96	OK	0.015	2236416	45	19
97	OK	0.000	2220032	45	20
98	OK	0.000	2224128	45	21
99	OK	0.015	2244608	45	19
100	OK	0.000	2236416	45	20
101	OK	0.015	2220032	45	20
102	OK	0.000	2232320	45	21
103	OK	0.000	2224128	45	20
104	OK	0.000	2224128	45	21
105	OK	0.000	2236416	45	20
106	OK	0.000	2224128	45	21

107	OK	0.000	2220032	45	22
108	OK	0.000	2224128	45	18
109	OK	0.000	2240512	45	19
110	OK	0.000	2236416	45	18
111	OK	0.000	2236416	45	19
112	OK	0.000	2224128	45	20
113	OK	0.000	2224128	45	18
114	OK	0.015	2224128	45	19
115	OK	0.000	2236416	45	19
116	OK	0.000	2224128	45	20
117	OK	0.015	2224128	45	19
118	OK	0.015	2220032	45	20
119	OK	0.000	2224128	45	19
120	OK	0.000	2236416	45	20
121	OK	0.000	2220032	45	21
122	OK	0.000	2224128	45	18
123	OK	0.000	2224128	45	19
124	OK	0.000	2220032	45	18
125	OK	0.000	2236416	45	19
126	OK	0.000	2236416	45	20
127	OK	0.015	2224128	45	19
128	OK	0.015	2224128	45	20
129	OK	0.000	2236416	45	19
130	OK	0.000	2228224	45	20
131	OK	0.000	2224128	45	21
132	OK	0.000	2220032	45	19
133	OK	0.015	2224128	45	20
134	OK	0.000	2224128	45	20
135	OK	0.000	2236416	45	21
136	OK	0.000	2220032	45	18
137	OK	0.000	2220032	45	19
138	OK	0.000	2224128	45	20
139	OK	0.000	2224128	45	21
140	OK	0.015	2228224	45	21
141	OK	0.000	2236416	45	22
142	OK	0.015	2236416	45	19
143	OK	0.015	2224128	45	20
144	OK	0.015	2224128	45	19
145	OK	0.000	2224128	45	20
146	OK	0.015	2236416	45	21
147	OK	0.000	2220032	45	19
148	OK	0.000	2236416	45	20

149	OK	0.000	2232320	45	20
150	OK	0.000	2228224	45	21
151	OK	0.000	2236416	45	20
152	OK	0.015	2236416	45	21
153	OK	0.000	2224128	45	20
154	OK	0.000	2220032	45	21
155	OK	0.000	2236416	45	22
156	OK	0.000	2224128	45	19
157	OK	0.000	2224128	45	20
158	OK	0.015	2236416	45	19
159	OK	0.000	2240512	45	20
160	OK	0.015	2240512	45	21
161	OK	0.000	2236416	45	19
162	OK	0.000	2236416	45	20
163	OK	0.000	2224128	45	20
164	OK	0.000	2240512	45	21
165	OK	0.015	2224128	45	20
166	OK	0.000	2236416	45	21
167	OK	0.000	2224128	45	20
168	OK	0.015	2224128	45	21
169	OK	0.015	2236416	45	22
170	OK	0.000	2240512	45	19
171	OK	0.000	2236416	45	20
172	OK	0.000	2236416	45	19
173	OK	0.015	2224128	45	20
174	OK	0.015	2236416	45	21
175	OK	0.000	2224128	45	19
176	OK	0.000	2224128	45	20
177	OK	0.000	2224128	45	20
178	OK	0.000	2224128	45	21
179	OK	0.000	2236416	45	20
180	OK	0.000	2240512	45	21
181	OK	0.000	2240512	45	20
182	OK	0.015	2224128	45	21
183	OK	0.000	2224128	45	22
184	OK	0.015	2236416	45	20
185	OK	0.000	2240512	45	21
186	OK	0.015	2224128	45	20
187	OK	0.000	2236416	45	21
188	OK	0.000	2220032	45	22
189	OK	0.000	2236416	45	20
190	OK	0.000	2236416	45	21

191	OK	0.000	2224128	45	21
192	OK	0.000	2228224	45	22
193	OK	0.000	2224128	45	21
194	OK	0.000	2224128	45	22
195	OK	0.015	2236416	45	21
196	OK	0.000	2224128	45	22
197	OK	0.015	2224128	45	23
198	OK	0.000	2224128	221	55
199	OK	0.015	2224128	220	59
200	OK	0.000	2236416	220	46
201	OK	0.000	2236416	223	48
202	OK	0.000	2240512	226	45
203	OK	0.000	2232320	1786	502
204	OK	0.000	2240512	1785	555
205	OK	0.000	2232320	1785	445
206	OK	0.000	2244608	1845	365
207	OK	0.015	2232320	1847	363
208	OK	0.000	2277376	9555	3006
209	OK	0.000	2297856	9554	3297
210	OK	0.000	2277376	9554	2730
211	OK	0.000	2273280	9303	1888
212	OK	0.000	2265088	9984	1877
213	OK	0.000	2461696	37691	12907
214	OK	0.000	2465792	37690	13974
215	OK	0.000	2482176	37690	11820
216	OK	0.015	2387968	39602	7150
217	OK	0.000	2371584	38744	7125
218	OK	0.000	3268608	178903	63876
219	OK	0.000	3264512	178902	68889
220	OK	0.015	3264512	178902	58890
221	OK	0.015	2789376	185712	33049
222	OK	0.015	2797568	180580	33013
223	OK	0.046	14761984	1853240	724890
224	OK	0.046	14761984	1853239	773873
225	OK	0.046	14761984	1853239	675751
226	OK	0.031	10063872	1855624	324156
227	OK	0.046	10063872	1856715	324455
228	OK	0.093	25477120	3473125	1412256
229	OK	0.093	25477120	3473124	1501788
230	OK	0.109	25477120	3473124	1322578
231	OK	0.078	17014784	3603994	592172
232	OK	0.078	17059840	3646224	592525

233	OK	0.125	28196864	3888905	1589032
234	OK	0.109	28200960	3888904	1688889
235	OK	0.109	28200960	3888904	1488890
236	OK	0.078	18608128	3890628	661024
237	OK	0.093	18706432	3986010	661067

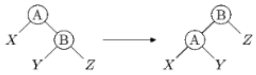
Делаю я левый поворот…

Условие

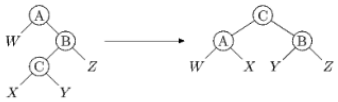
Для балансировки АВЛ-дерева при операциях вставки и удаления производятся *левые* и *правые* повороты. Левый поворот в вершине производится, когда баланс этой вершины больше 1, аналогично, правый поворот производится при балансе, меньшем −1.

Существует два разных левых (как, разумеется, и правых) поворота: *большой* и *малый* левый поворот.

Малый левый поворот осуществляется следующим образом:



Заметим, что если до выполнения малого левого поворота был нарушен баланс только корня дерева, то после его выполнения все вершины становятся сбалансированными, за исключением случая, когда у правого ребенка корня баланс до поворота равен −1. В этом случае вместо малого левого поворота выполняется большой левый поворот, который осуществляется так:



Дано дерево, в котором баланс корня равен 2. Сделайте левый поворот.

Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число N $(3 \le N \le 2 \cdot 10^5)$ — 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. Баланс корня дерева (вершины с номером 1) равен 2, баланс всех остальных вершин находится в пределах от -1 до 1.

Формат выходного файла

Выведите в том же формате дерево после осуществления левого поворота. Нумерация вершин может быть произвольной при условии соблюдения формата. Так, номер вершины должен быть меньше номера ее детей.

Пример

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

Решение

openedu/week7/lab7_2.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <list>
#include <assert.h>
#include <tuple>

using namespace std;

typedef int32_t balance_t;

struct Node {
	int key;
	balance_t balance = 0;
	int height = 1;

	Node* parent = nullptr;
	Node* left = nullptr;
	Node* right = nullptr;

	// dirty hack
	int latest_population_descr_index;

	balance_t get_left_balance() { return (left) ? left->balance : -1; };
	balance_t get_right_balance() { return (right) ? right->balance : -1; };

	int get_left_height() { return (left) ? left->height : -1; };
	int get_right_height() { return (right) ? right->height : -1; };

	void replace_child(Node* from, Node* to) {
		if (left == from) {
			left = to;
		} else {
			assert(right == from);
			right = to;
		}
		to->parent = this;
		from->parent = nullptr;
	}
};

struct NodeDescription {
```

```
	int k;
	int li = 0;
	int ri = 0;
};

struct TreeDescription {
	vector<NodeDescription> descriptions;

	TreeDescription() {}
	TreeDescription(int size) : descriptions(size) {}

	NodeDescription& description_for_index(int index) {
		return descriptions[index];
	}

	void populate_from_tree(Node* tree) {
		// children set later
		descriptions.push_back(NodeDescription{ tree->key });

		// dirty hack for setting children!
		int current_index = descriptions.size() - 1;
		tree->latest_population_descr_index = current_index;

		// setting children of previously added things
		if (tree->parent) {
			if (tree->parent->left == tree) {
				descriptions[tree->parent-
>latest_population_descr_index].li = current_index + 1;
			} else {
				assert(tree->parent->right == tree);
				descriptions[tree->parent-
>latest_population_descr_index].ri = current_index + 1;
			}
		}

		// recursively populate from subtrees
		if (tree->left) {
			populate_from_tree(tree->left);
		}

		if (tree->right) {
			populate_from_tree(tree->right);
		}
	}
};

Node* build_tree(TreeDescription &input, int root_index = 0) {
	// root node description
	auto& rd = input.description_for_index(root_index);

	// create new node
	Node* tree = new Node();
	tree->key = rd.k;

	// recursively build left and right subtrees if needed
	if (rd.li) {
		tree->left = build_tree(input, rd.li - 1);
		tree->left->parent = tree;
	}

	if (rd.ri) {
		tree->right = build_tree(input, rd.ri - 1);
		tree->right->parent = tree;
	}

	// calculate height and balance
	tree->height = max(tree->get_right_height(), tree->get_left_height()) + 1;
```

```
	tree->balance = tree->get_right_height() - tree->get_left_height();

	return tree;
}

Node*& find_node(Node*& tree, int key) {
	if (tree == nullptr) {
		return tree;
	} else if (key == tree->key) {
		return tree;
	} else if (key < tree->key) {
		return find_node(tree->left, key);
	} else {
		return find_node(tree->right, key);
	}
}

void rotate_l(Node*& tree) {
	assert(tree->right != nullptr);
	Node* a = tree;
	Node* b = a->right;
	Node* x = a->left;
	Node* y = b->left;
	Node* z = b->right;

	if (a->parent) {
		a->parent->replace_child(a, b);
	} else {
		b->parent = nullptr;
	}

	b->left = a;
	a->parent = b;

	a->right = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
}

void rotate_r(Node*& tree) {
	assert(tree->left != nullptr);
	Node* a = tree;
	Node* b = a->left;
	Node* x = a->right;
	Node* y = b->right;
	Node* z = b->left;

	if (a->parent) {
		a->parent->replace_child(a, b);
	} else {
		b->parent = nullptr;
	}

	b->right = a;
	a->parent = b;

	a->left = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
}
```

```
void rotate_lr(Node*& tree) {
	// большой правый
	rotate_l(tree->left);
	rotate_r(tree);
}

void rotate_rl(Node*& tree) {
	// большой левый
	rotate_r(tree->right);
	rotate_l(tree);
}

int main() {
	int n;
	io >> n;

	TreeDescription input(n);
	for (auto& descr : input.descriptions) {
		io >> descr.k >> descr.li >> descr.ri;
	}

	auto tree = build_tree(input);

	if (tree->right->balance == -1) {
		rotate_rl(tree);
	} else {
		rotate_l(tree);
	}

	TreeDescription output;
	output.populate_from_tree(tree);

	io << n << "\n";
	for (auto& descr : output.descriptions) {
		io << descr.k << " " << descr.li << " " << descr.ri << "\n";
	}

	return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.203	24616960	3986416	3986416
1	OK	0.000	2252800	54	54
2	OK	0.015	2248704	24	24
3	OK	0.000	2265088	24	24
4	OK	0.015	2252800	31	31
5	OK	0.000	2252800	45	45
6	OK	0.000	2248704	45	45
7	OK	0.015	2265088	45	45
8	OK	0.000	2265088	45	45
9	OK	0.000	2252800	52	52
10	OK	0.015	2248704	52	52
11	OK	0.000	2252800	52	52
12	OK	0.000	2256896	52	52

13	OK	0.015	2252800	52	52
14	OK	0.000	2252800	52	52
15	OK	0.000	2252800	59	59
16	OK	0.000	2248704	59	59
17	OK	0.000	2248704	59	59
18	OK	0.000	2252800	59	59
19	OK	0.015	2260992	66	66
20	OK	0.000	2260992	75	75
21	OK	0.015	2277376	75	75
22	OK	0.000	2256896	75	75
23	OK	0.015	2265088	75	75
24	OK	0.000	2260992	75	75
25	OK	0.015	2277376	75	75
26	OK	0.000	2260992	75	75
27	OK	0.000	2260992	75	75
28	OK	0.000	2277376	75	75
29	OK	0.000	2260992	75	75
30	OK	0.015	2256896	75	75
31	OK	0.000	2269184	75	75
32	OK	0.000	2260992	75	75
33	OK	0.000	2273280	75	75
34	OK	0.000	2265088	75	75
35	OK	0.000	2260992	75	75
36	OK	0.015	2260992	75	75
37	OK	0.015	2277376	75	75
38	OK	0.000	2260992	75	75
39	OK	0.000	2256896	75	75
40	OK	0.015	2256896	75	75
41	OK	0.000	2273280	75	75
42	OK	0.000	2269184	75	75
43	OK	0.000	2273280	75	75
44	OK	0.015	2260992	75	75
45	OK	0.015	2277376	75	75
46	OK	0.000	2256896	75	75
47	OK	0.015	2260992	75	75
48	OK	0.000	2260992	75	75
49	OK	0.000	2260992	75	75
50	OK	0.000	2256896	75	75
51	OK	0.000	2260992	75	75
52	OK	0.000	2260992	84	84
53	OK	0.000	2269184	84	84
54	OK	0.015	2260992	84	84

55	OK	0.000	2265088	84	84
56	OK	0.000	2256896	84	84
57	OK	0.015	2260992	84	84
58	OK	0.000	2277376	84	84
59	OK	0.000	2260992	84	84
60	OK	0.000	2273280	84	84
61	OK	0.000	2273280	84	84
62	OK	0.015	2260992	84	84
63	OK	0.000	2273280	84	84
64	OK	0.000	2260992	84	84
65	OK	0.000	2265088	84	84
66	OK	0.000	2260992	84	84
67	OK	0.000	2277376	84	84
68	OK	0.000	2256896	84	84
69	OK	0.000	2260992	84	84
70	OK	0.000	2273280	84	84
71	OK	0.000	2273280	84	84
72	OK	0.000	2260992	84	84
73	OK	0.000	2273280	84	84
74	OK	0.000	2273280	84	84
75	OK	0.000	2265088	84	84
76	OK	0.015	2277376	84	84
77	OK	0.015	2260992	84	84
78	OK	0.000	2260992	84	84
79	OK	0.015	2260992	84	84
80	OK	0.000	2256896	84	84
81	OK	0.015	2260992	84	84
82	OK	0.000	2260992	84	84
83	OK	0.000	2256896	84	84
84	OK	0.000	2260992	84	84
85	OK	0.015	2265088	84	84
86	OK	0.000	2260992	84	84
87	OK	0.031	2256896	84	84
88	OK	0.000	2260992	84	84
89	OK	0.015	2273280	84	84
90	OK	0.000	2256896	84	84
91	OK	0.000	2256896	84	84
92	OK	0.000	2277376	84	84
93	OK	0.000	2260992	84	84
94	OK	0.015	2273280	84	84
95	OK	0.015	2265088	84	84
96	OK	0.015	2273280	84	84

97	OK	0.000	2260992	84	84
98	OK	0.000	2260992	84	84
99	OK	0.000	2260992	84	84
100	OK	0.015	2260992	84	84
101	OK	0.000	2260992	84	84
102	OK	0.000	2256896	84	84
103	OK	0.000	2260992	84	84
104	OK	0.015	2260992	84	84
105	OK	0.015	2260992	84	84
106	OK	0.015	2260992	84	84
107	OK	0.015	2260992	84	84
108	OK	0.000	2260992	84	84
109	OK	0.000	2260992	84	84
110	OK	0.000	2273280	84	84
111	OK	0.000	2273280	84	84
112	OK	0.000	2260992	84	84
113	OK	0.015	2260992	84	84
114	OK	0.015	2273280	84	84
115	OK	0.000	2260992	84	84
116	OK	0.015	2265088	84	84
117	OK	0.015	2260992	84	84
118	OK	0.000	2260992	84	84
119	OK	0.000	2273280	84	84
120	OK	0.015	2273280	84	84
121	OK	0.015	2273280	84	84
122	OK	0.015	2277376	84	84
123	OK	0.015	2260992	84	84
124	OK	0.015	2260992	84	84
125	OK	0.000	2256896	84	84
126	OK	0.000	2260992	84	84
127	OK	0.015	2273280	84	84
128	OK	0.015	2260992	84	84
129	OK	0.000	2273280	84	84
130	OK	0.000	2260992	84	84
131	OK	0.000	2260992	84	84
132	OK	0.000	2273280	93	93
133	OK	0.000	2273280	93	93
134	OK	0.000	2260992	93	93
135	OK	0.015	2265088	93	93
136	OK	0.000	2273280	93	93
137	OK	0.000	2273280	93	93
138	OK	0.000	2260992	93	93

139	OK	0.000	2260992	93	93
140	OK	0.000	2260992	93	93
141	OK	0.015	2260992	93	93
142	OK	0.031	2273280	93	93
143	OK	0.000	2260992	93	93
144	OK	0.015	2260992	93	93
145	OK	0.000	2265088	93	93
146	OK	0.015	2273280	93	93
147	OK	0.015	2260992	93	93
148	OK	0.015	2273280	93	93
149	OK	0.015	2260992	93	93
150	OK	0.000	2260992	93	93
151	OK	0.000	2260992	93	93
152	OK	0.000	2260992	93	93
153	OK	0.000	2260992	93	93
154	OK	0.015	2260992	93	93
155	OK	0.000	2277376	93	93
156	OK	0.000	2260992	93	93
157	OK	0.000	2260992	93	93
158	OK	0.000	2260992	93	93
159	OK	0.015	2273280	93	93
160	OK	0.000	2260992	93	93
161	OK	0.000	2256896	93	93
162	OK	0.000	2273280	93	93
163	OK	0.000	2260992	93	93
164	OK	0.000	2260992	93	93
165	OK	0.015	2252800	93	93
166	OK	0.015	2265088	93	93
167	OK	0.015	2281472	93	93
168	OK	0.000	2260992	93	93
169	OK	0.000	2269184	93	93
170	OK	0.000	2260992	93	93
171	OK	0.000	2260992	93	93
172	OK	0.031	2269184	93	93
173	OK	0.015	2260992	93	93
174	OK	0.000	2260992	93	93
175	OK	0.000	2273280	93	93
176	OK	0.000	2265088	93	93
177	OK	0.015	2260992	93	93
178	OK	0.000	2260992	93	93
179	OK	0.000	2256896	93	93
180	OK	0.015	2256896	93	93

181	OK	0.015	2260992	93	93
182	OK	0.000	2260992	93	93
183	OK	0.015	2260992	93	93
184	OK	0.015	2260992	93	93
185	OK	0.000	2256896	93	93
186	OK	0.000	2265088	93	93
187	OK	0.015	2256896	93	93
188	OK	0.015	2260992	93	93
189	OK	0.015	2260992	93	93
190	OK	0.000	2260992	93	93
191	OK	0.000	2260992	93	93
192	OK	0.015	2260992	93	93
193	OK	0.000	2256896	93	93
194	OK	0.015	2273280	93	93
195	OK	0.015	2269184	93	93
196	OK	0.000	2265088	93	93
197	OK	0.000	2260992	93	93
198	OK	0.015	2260992	93	93
199	OK	0.015	2260992	93	93
200	OK	0.000	2260992	93	93
201	OK	0.000	2256896	93	93
202	OK	0.000	2273280	93	93
203	OK	0.000	2273280	93	93
204	OK	0.015	2260992	93	93
205	OK	0.000	2273280	93	93
206	OK	0.000	2265088	93	93
207	OK	0.000	2260992	93	93
208	OK	0.031	2260992	93	93
209	OK	0.000	2260992	93	93
210	OK	0.015	2273280	93	93
211	OK	0.015	2260992	93	93
212	OK	0.015	2273280	93	93
213	OK	0.000	2273280	93	93
214	OK	0.000	2273280	93	93
215	OK	0.015	2273280	93	93
216	OK	0.000	2265088	93	93
217	OK	0.000	2273280	93	93
218	OK	0.000	2260992	93	93
219	OK	0.000	2260992	93	93
220	OK	0.000	2273280	93	93
221	OK	0.000	2260992	93	93
222	OK	0.015	2256896	93	93

223	OK	0.000	2260992	93	93
224	OK	0.000	2260992	93	93
225	OK	0.015	2260992	93	93
226	OK	0.000	2265088	93	93
227	OK	0.015	2256896	93	93
228	OK	0.015	2256896	93	93
229	OK	0.000	2260992	93	93
230	OK	0.000	2260992	93	93
231	OK	0.000	2260992	93	93
232	OK	0.000	2273280	93	93
233	OK	0.000	2256896	93	93
234	OK	0.000	2260992	93	93
235	OK	0.015	2256896	93	93
236	OK	0.000	2277376	93	93
237	OK	0.000	2260992	93	93
238	OK	0.000	2260992	93	93
239	OK	0.000	2269184	93	93
240	OK	0.015	2260992	93	93
241	OK	0.015	2260992	93	93
242	OK	0.000	2252800	93	93
243	OK	0.015	2260992	93	93
244	OK	0.000	2260992	93	93
245	OK	0.000	2260992	93	93
246	OK	0.000	2273280	93	93
247	OK	0.000	2260992	93	93
248	OK	0.000	2256896	93	93
249	OK	0.000	2260992	93	93
250	OK	0.000	2260992	93	93
251	OK	0.000	2273280	93	93
252	OK	0.000	2273280	220	220
253	OK	0.000	2269184	1798	1798
254	OK	0.015	2265088	1842	1842
255	OK	0.000	2338816	9606	9606
256	OK	0.000	2318336	9569	9569
257	OK	0.015	2318336	9662	9662
258	OK	0.015	2318336	9777	9777
259	OK	0.000	2433024	37717	37717
260	OK	0.000	2433024	37874	37874
261	OK	0.000	2441216	39268	39268
262	OK	0.015	2433024	39470	39470
263	OK	0.015	3096576	181024	181024
264	OK	0.015	3092480	183405	183405

265	OK	0.015	3096576	180784	180784
266	OK	0.015	3096576	183152	183152
267	OK	0.015	3096576	181246	181246
268	OK	0.015	3096576	180886	180886
269	OK	0.093	13688832	1843051	1843051
270	OK	0.093	13737984	1888721	1888721
271	OK	0.093	13475840	1866794	1866794
272	OK	0.093	13746176	1898864	1898864
273	OK	0.093	13754368	1908693	1908693
274	OK	0.093	13729792	1881384	1881384
275	OK	0.171	22421504	3526463	3526463
276	OK	0.171	22458368	3559824	3559824
277	OK	0.171	22495232	3598289	3598289
278	OK	0.171	22482944	3590402	3590402
279	OK	0.156	22466560	3567894	3567894
280	OK	0.171	22462464	3566660	3566660
281	OK	0.171	22384640	3487414	3487414
282	OK	0.171	24502272	3874088	3874088
283	OK	0.203	24608768	3978208	3978208
284	OK	0.187	24588288	3957801	3957801
285	OK	0.187	24608768	3978349	3978349
286	OK	0.187	24616960	3986416	3986416
287	OK	0.187	24559616	3933437	3933437
288	OK	0.187	24555520	3926735	3926735

Вставка в АВЛ-дерево

Условие

Вставка в АВЛ-дерево вершины V с ключом X при условии, что такой вершины в этом дереве нет, осуществляется следующим образом:

- находится вершина W, ребенком которой должна стать вершина V;
- вершина V делается ребенком вершины W;
- производится подъем от вершины W к корню, при этом, если какая-то из вершин несбалансирована, производится, в зависимости от значения баланса, левый или правый поворот.

Первый этап нуждается в пояснении. Спуск до будущего родителя вершины V осуществляется, начиная от корня, следующим образом:

- Пусть ключ текущей вершины равен Y.
- Если $X < Y$ и у текущей вершины есть левый ребенок, переходим к левому ребенку.
- Если $X < Y$ и у текущей вершины нет левого ребенка, то останавливаемся, текущая вершина будет родителем новой вершины.
- Если $X > Y$ и у текущей вершины есть правый ребенок, переходим к правому ребенку.
- Если $X > Y$ и у текущей вершины нет правого ребенка, то останавливаемся, текущая вершина будет родителем новой вершины.

Отдельно рассматривается следующий крайний случай — если до вставки дерево было пустым, то вставка новой вершины осуществляется проще: новая вершина становится корнем дерева.

Формат входного файла

Входной файл содержит описание двоичного дерева, а также ключа вершины, которую требуется вставить в дерево.

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 корректным АВЛ-деревом.

В последней строке содержится число X ($|X| \le 10^9$) — ключ вершины, которую требуется вставить в дерево. Гарантируется, что такой вершины в дереве нет.

Формат выходного файла

Выведите в том же формате дерево после осуществления операции вставки. Нумерация вершин может быть произвольной при условии соблюдения формата.

Пример

input.txt	output.txt
2 3 0 2 4 0 0 5	3 4 2 3 3 0 0 5 0 0

Решение

openedu/week7/lab7_3.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <list>
#include <assert.h>
#include <tuple>

using namespace std;
```

```
typedef int32_t balance_t;

struct Node {
	int key;
	balance_t balance = 0;
	int height = 1;

	Node* parent = nullptr;
	Node* left = nullptr;
	Node* right = nullptr;

	// dirty hack
	size_t latest_population_descr_index;

	Node(int key) : key(key) {}

	balance_t get_left_balance() { return (left) ? left->balance : -1; };
	balance_t get_right_balance() { return (right) ? right->balance : -1; };

	int get_left_height() { return (left) ? left->height : 0; };
	int get_right_height() { return (right) ? right->height : 0; };

	void replace_child(Node* from, Node* to) {
		if (left == from) {
			left = to;
		} else {
			assert(right == from);
			right = to;
		}
		if (to) {
			to->parent = this;
		}
		from->parent = nullptr;
	}

	void subtree_change_height(Node* from, int dh = 1) {
		if (left == from) {
			balance -= dh;
		} else {
			assert(right == from);
			balance += dh;
		}
		height += dh;
	}

	void recalc_structure(int trust_deep = 3) {
		if (trust_deep) {
			if (left) left->recalc_structure(trust_deep - 1);
			if (right) right->recalc_structure(trust_deep - 1);
			height = max(get_right_height(), get_left_height()) + 1;
			balance = get_right_height() - get_left_height();
		}
	}
};

struct NodeDescription {
	int k;
	int li = 0;
	int ri = 0;
};

struct TreeDescription {
	vector<NodeDescription> descriptions;

	TreeDescription() {}
	TreeDescription(int size) : descriptions(size) {}
```

```
	NodeDescription& description_for_index(size_t index) {
		return descriptions[index];
	}

	void populate_from_tree(Node* tree) {
		// children set later
		descriptions.push_back(NodeDescription{ tree->key });

		// dirty hack for setting children!
		size_t current_index = descriptions.size() - 1;
		tree->latest_population_descr_index = current_index;

		// setting children of previously added things
		if (tree->parent) {
			if (tree->parent->left == tree) {
				descriptions[tree->parent-
>latest_population_descr_index].li = current_index + 1;
			} else {
				assert(tree->parent->right == tree);
				descriptions[tree->parent-
>latest_population_descr_index].ri = current_index + 1;
			}
		}

		// recursively populate from subtrees
		if (tree->left) {
			populate_from_tree(tree->left);
		}

		if (tree->right) {
			populate_from_tree(tree->right);
		}
	}
};

Node* build_tree(TreeDescription &input, size_t root_index = 0) {
	// root node description
	auto& rd = input.description_for_index(root_index);

	// create new node
	Node* tree = new Node(rd.k);

	// recursively build left and right subtrees if needed
	if (rd.li) {
		tree->left = build_tree(input, rd.li - 1);
		tree->left->parent = tree;
	}

	if (rd.ri) {
		tree->right = build_tree(input, rd.ri - 1);
		tree->right->parent = tree;
	}

	// calculate height and balance
	tree->height = max(tree->get_right_height(), tree->get_left_height()) + 1;
	tree->balance = tree->get_right_height() - tree->get_left_height();

	return tree;
}

Node*& find_node(Node*& tree, int key) {
	if (tree == nullptr) {
		return tree;
	} else if (key == tree->key) {
		return tree;
	} else if (key < tree->key) {
		return find_node(tree->left, key);
```

```
	} else {
		return find_node(tree->right, key);
	}
}

void rotate_l(Node*& tree) {
	// малый левый
	assert(tree->right != nullptr);
	Node* a = tree;
	Node* b = a->right;
	Node* x = a->left;
	Node* y = b->left;
	Node* z = b->right;

	if (a->parent) {
		a->parent->replace_child(a, b);
	} else {
		b->parent = nullptr;
	}

	b->left = a;
	a->parent = b;

	a->right = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
	tree->recalc_structure();
}

void rotate_r(Node*& tree) {
	// малый правый
	assert(tree->left != nullptr);
	Node* a = tree;
	Node* b = a->left;
	Node* x = a->right;
	Node* y = b->right;
	Node* z = b->left;

	if (a->parent) {
		a->parent->replace_child(a, b);
	} else {
		b->parent = nullptr;
	}

	b->right = a;
	a->parent = b;

	a->left = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
	tree->recalc_structure();
}

void rotate_lr(Node*& tree) {
	// большой правый
	rotate_l(tree->left);
	rotate_r(tree);
}

void rotate_rl(Node*& tree) {
	// большой левый
```

```
	rotate_r(tree->right);
	rotate_l(tree);
}

void balance_tree(Node*& tree) {
	if (tree->balance == 2) {
		if (tree->right->balance == -1) {
			rotate_rl(tree);
		} else {
			rotate_l(tree);
		}
	} else if (tree->balance == -2) {
		if (tree->left->balance == 1) {
			rotate_lr(tree);
		} else {
			rotate_r(tree);
		}
	}
}

bool insert_to_tree(Node*& tree, Node* node) {
	if (tree == nullptr) {
		tree = node;
		return true;
	}

	bool balancing_done = false;
	// digging into the tree to find a right place for a node
	if (node->key == tree->key) {
		// assert(false && "key already in tree");
		return true;
	} else if (node->key < tree->key) {
		if (tree->left) {
			balancing_done = insert_to_tree(tree->left, node);
		} else {
			tree->left = node;
			node->parent = tree;
			tree->subtree_change_height(node, 1);
		}
	} else {
		if (tree->right) {
			balancing_done = insert_to_tree(tree->right, node);
		} else {
			tree->right = node;
			node->parent = tree;
			tree->subtree_change_height(node, 1);
		}
	}

	// balancing on our way back if needed
	balance_tree(tree);

	// recalculate balances due to height change
	// if node has a parent and subtree's height was increased (balance != 0)
	if (tree->parent && tree->balance != 0 && !balancing_done) {
		tree->parent->subtree_change_height(tree, 1);
		return false;
	}
	return true;
}

void delete_node_simple(Node*& tree) {
	if (tree->left == nullptr && tree->right == nullptr) {
		if (tree->parent) {
			tree->parent->subtree_change_height(tree, -1);
			// tree->parent->replace_child(tree, nullptr);
		}
```

```
		delete tree;
		tree = nullptr;
	} else if (tree->left == nullptr || tree->right == nullptr) {
		Node* saved = (tree->right) ? tree->right : tree->left;
		if (tree->parent) {
			tree->parent->subtree_change_height(tree, -1);
			// tree->parent->replace_child(tree, saved);
		}
		saved->parent = tree->parent;
		delete tree;
		tree = saved;
	} else {
		assert(false && "not a simple node to delete!");
	}
}


bool delete_from_tree(Node*& tree, int key, Node* target = nullptr) {
	if (tree == nullptr) {
		return true;
	}

	if (!tree->parent && !tree->left && !tree->right) {
		if (key == tree->key) {
			delete_node_simple(tree);
			return false;
		} else {
			// assert(false && "key not in a tree!");
			return true;
		}
	}

	bool balancing_done = false;
	if (!target) {
		// digging into the tree
		if (key == tree->key) {
			if (!tree->left && !tree->right) {
				delete_node_simple(tree);
				return false;
			} else if (!tree->left) {
				delete_node_simple(tree);
				return false;
			} else {
				balancing_done = delete_from_tree(tree->left, key,
tree);
			}
		} else if (key < tree->key) {
			balancing_done = delete_from_tree(tree->left, key);
		} else {
			assert(key > tree->key);
			balancing_done = delete_from_tree(tree->right, key);
		}
	} else {
		if (tree->right) {
			balancing_done = delete_from_tree(tree->right, key, target);
		} else {
			// found prev
			target->key = tree->key;
			delete_node_simple(tree);
			return false;
		}
	}

	// balancing on our way back if needed
	balance_tree(tree);

	// recalculate balances due to height change
```

```
	// if node has a parent and subtree's height was DECREASED (balance == 0)
	if (tree->parent && tree->balance == 0 && !balancing_done) {
		tree->parent->subtree_change_height(tree, -1);
		return false;
	}
	return true;
}

int main() {
	int n;
	io >> n;

	TreeDescription input(n);
	for (auto& descr : input.descriptions) {
		io >> descr.k >> descr.li >> descr.ri;
	}

	Node* tree = nullptr;
	if (n) {
		tree = build_tree(input);
	}

	int k;
	io >> k;
	insert_to_tree(tree, new Node(k));

	TreeDescription output;
	output.populate_from_tree(tree);

	io << output.descriptions.size() << "\n";
	for (auto& descr : output.descriptions) {
		io << descr.k << " " << descr.li << " " << descr.ri << "\n";
	}

	return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.203	24645632	4011957	4011966
1	OK	0.015	2269184	20	24
2	OK	0.000	2269184	6	10
3	OK	0.000	2256896	14	18
4	OK	0.015	2260992	13	17
5	OK	0.000	2269184	21	25
6	OK	0.000	2269184	20	24
7	OK	0.000	2269184	20	24
8	OK	0.000	2269184	21	25
9	OK	0.015	2269184	20	24
10	OK	0.015	2269184	20	24
11	OK	0.015	2277376	28	32
12	OK	0.000	2269184	27	31
13	OK	0.015	2256896	27	31
14	OK	0.000	2281472	27	31

15	OK	0.000	2269184	35	39
16	OK	0.000	2269184	34	38
17	OK	0.000	2269184	34	38
18	OK	0.000	2269184	34	38
19	OK	0.015	2269184	34	38
20	OK	0.015	2281472	35	39
21	OK	0.000	2260992	34	38
22	OK	0.000	2256896	34	38
23	OK	0.000	2269184	34	38
24	OK	0.015	2269184	34	38
25	OK	0.000	2265088	35	39
26	OK	0.015	2281472	34	38
27	OK	0.000	2269184	34	38
28	OK	0.031	2277376	34	38
29	OK	0.015	2260992	34	38
30	OK	0.015	2269184	35	39
31	OK	0.000	2260992	34	38
32	OK	0.000	2256896	34	38
33	OK	0.000	2265088	34	38
34	OK	0.015	2269184	34	38
35	OK	0.015	2269184	42	46
36	OK	0.000	2269184	41	45
37	OK	0.000	2269184	41	45
38	OK	0.000	2281472	41	45
39	OK	0.015	2269184	41	45
40	OK	0.015	2269184	41	45
41	OK	0.000	2256896	42	46
42	OK	0.015	2256896	41	45
43	OK	0.015	2269184	41	45
44	OK	0.000	2269184	41	45
45	OK	0.000	2269184	41	45
46	OK	0.000	2285568	41	45
47	OK	0.000	2281472	42	46
48	OK	0.015	2269184	41	45
49	OK	0.000	2269184	41	45
50	OK	0.000	2281472	41	45
51	OK	0.015	2277376	41	45
52	OK	0.015	2256896	41	45
53	OK	0.000	2269184	42	46
54	OK	0.000	2265088	41	45
55	OK	0.000	2269184	41	45
56	OK	0.000	2269184	41	45

57	OK	0.000	2281472	41	45
58	OK	0.000	2281472	41	45
59	OK	0.000	2269184	42	46
60	OK	0.000	2281472	41	45
61	OK	0.000	2260992	41	45
62	OK	0.015	2269184	41	45
63	OK	0.015	2269184	41	45
64	OK	0.000	2269184	41	45
65	OK	0.015	2281472	42	46
66	OK	0.000	2269184	41	45
67	OK	0.000	2281472	41	45
68	OK	0.000	2269184	41	45
69	OK	0.000	2269184	41	45
70	OK	0.015	2281472	41	45
71	OK	0.000	2256896	50	54
72	OK	0.000	2269184	49	53
73	OK	0.000	2281472	49	53
74	OK	0.000	2252800	49	53
75	OK	0.000	2285568	49	53
76	OK	0.000	2265088	49	53
77	OK	0.000	2265088	50	54
78	OK	0.015	2277376	50	54
79	OK	0.000	2281472	49	53
80	OK	0.000	2269184	49	53
81	OK	0.000	2273280	49	53
82	OK	0.000	2269184	49	53
83	OK	0.000	2281472	49	53
84	OK	0.000	2252800	50	54
85	OK	0.015	2269184	50	54
86	OK	0.000	2269184	49	53
87	OK	0.000	2269184	49	53
88	OK	0.000	2265088	49	53
89	OK	0.000	2269184	49	53
90	OK	0.000	2269184	49	53
91	OK	0.000	2269184	50	54
92	OK	0.000	2256896	50	54
93	OK	0.015	2269184	49	53
94	OK	0.000	2252800	49	53
95	OK	0.000	2269184	49	53
96	OK	0.000	2281472	49	53
97	OK	0.000	2281472	49	53
98	OK	0.000	2269184	50	54

99	OK	0.015	2281472	58	62
100	OK	0.000	2269184	57	61
101	OK	0.015	2269184	57	61
102	OK	0.000	2260992	57	61
103	OK	0.000	2269184	57	61
104	OK	0.015	2256896	57	61
105	OK	0.015	2265088	58	62
106	OK	0.015	2281472	58	62
107	OK	0.015	2281472	58	62
108	OK	0.000	2265088	57	61
109	OK	0.000	2269184	57	61
110	OK	0.015	2277376	57	61
111	OK	0.000	2281472	57	61
112	OK	0.015	2265088	57	61
113	OK	0.000	2273280	58	62
114	OK	0.000	2269184	58	62
115	OK	0.000	2256896	58	62
116	OK	0.015	2281472	57	61
117	OK	0.000	2269184	57	61
118	OK	0.015	2269184	57	61
119	OK	0.015	2269184	57	61
120	OK	0.000	2277376	57	61
121	OK	0.000	2281472	58	62
122	OK	0.015	2265088	58	62
123	OK	0.000	2269184	58	62
124	OK	0.000	2256896	57	61
125	OK	0.000	2273280	57	61
126	OK	0.031	2269184	57	61
127	OK	0.031	2269184	57	61
128	OK	0.015	2265088	57	61
129	OK	0.000	2281472	58	62
130	OK	0.000	2269184	58	62
131	OK	0.000	2269184	58	62
132	OK	0.031	2269184	57	61
133	OK	0.000	2256896	57	61
134	OK	0.015	2260992	57	61
135	OK	0.015	2269184	57	61
136	OK	0.000	2265088	57	61
137	OK	0.000	2269184	58	62
138	OK	0.015	2281472	58	62
139	OK	0.000	2281472	58	62
140	OK	0.015	2281472	57	61

141	OK	0.015	2281472	57	61
142	OK	0.000	2269184	57	61
143	OK	0.000	2252800	57	61
144	OK	0.000	2273280	57	61
145	OK	0.000	2269184	58	62
146	OK	0.000	2269184	58	62
147	OK	0.000	2281472	58	62
148	OK	0.015	2265088	57	61
149	OK	0.000	2281472	57	61
150	OK	0.000	2269184	57	61
151	OK	0.000	2265088	57	61
152	OK	0.000	2285568	57	61
153	OK	0.015	2260992	58	62
154	OK	0.000	2256896	58	62
155	OK	0.000	2269184	58	62
156	OK	0.000	2277376	57	61
157	OK	0.000	2269184	57	61
158	OK	0.000	2265088	57	61
159	OK	0.000	2269184	57	61
160	OK	0.000	2269184	57	61
161	OK	0.000	2269184	58	62
162	OK	0.015	2265088	58	62
163	OK	0.000	2273280	58	62
164	OK	0.000	2252800	57	61
165	OK	0.015	2269184	57	61
166	OK	0.000	2269184	57	61
167	OK	0.000	2269184	57	61
168	OK	0.000	2269184	57	61
169	OK	0.015	2269184	58	62
170	OK	0.000	2281472	58	62
171	OK	0.000	2281472	58	62
172	OK	0.000	2269184	57	61
173	OK	0.015	2260992	57	61
174	OK	0.000	2256896	57	61
175	OK	0.000	2281472	57	61
176	OK	0.000	2281472	57	61
177	OK	0.015	2269184	58	62
178	OK	0.000	2277376	58	62
179	OK	0.000	2281472	58	62
180	OK	0.000	2269184	57	61
181	OK	0.015	2269184	57	61
182	OK	0.015	2269184	57	61

183	OK	0.015	2269184	57	61
184	OK	0.015	2256896	57	61
185	OK	0.015	2281472	58	62
186	OK	0.000	2269184	58	62
187	OK	0.000	2269184	58	62
188	OK	0.015	2265088	57	61
189	OK	0.000	2281472	57	61
190	OK	0.015	2269184	57	61
191	OK	0.000	2265088	57	61
192	OK	0.015	2281472	57	61
193	OK	0.015	2256896	58	62
194	OK	0.000	2273280	58	62
195	OK	0.000	2265088	58	62
196	OK	0.000	2269184	57	61
197	OK	0.000	2265088	57	61
198	OK	0.015	2281472	57	61
199	OK	0.000	2281472	57	61
200	OK	0.000	2281472	57	61
201	OK	0.015	2269184	58	62
202	OK	0.000	2265088	58	62
203	OK	0.015	2277376	58	62
204	OK	0.000	2256896	57	61
205	OK	0.015	2277376	57	61
206	OK	0.000	2269184	57	61
207	OK	0.015	2281472	57	61
208	OK	0.015	2269184	57	61
209	OK	0.015	2269184	58	62
210	OK	0.000	2269184	58	62
211	OK	0.000	2281472	58	62
212	OK	0.015	2269184	57	61
213	OK	0.015	2273280	57	61
214	OK	0.000	2256896	57	61
215	OK	0.000	2269184	57	61
216	OK	0.015	2281472	57	61
217	OK	0.000	2269184	58	62
218	OK	0.015	2269184	58	62
219	OK	0.000	2269184	58	62
220	OK	0.015	2265088	57	61
221	OK	0.000	2269184	57	61
222	OK	0.000	2269184	57	61
223	OK	0.000	2269184	57	61
224	OK	0.000	2269184	57	61

225	OK	0.000	2269184	58	62
226	OK	0.000	2265088	58	62
227	OK	0.000	2269184	58	62
228	OK	0.000	2281472	57	61
229	OK	0.000	2281472	57	61
230	OK	0.000	2281472	57	61
231	OK	0.000	2269184	57	61
232	OK	0.000	2269184	57	61
233	OK	0.000	2260992	58	62
234	OK	0.000	2252800	58	62
235	OK	0.000	2265088	240	245
236	OK	0.000	2277376	243	248
237	OK	0.015	2285568	1835	1841
238	OK	0.000	2285568	1815	1821
239	OK	0.015	2285568	1834	1840
240	OK	0.015	2334720	9951	9957
241	OK	0.015	2338816	9831	9837
242	OK	0.000	2330624	9854	9860
243	OK	0.015	2441216	38301	38308
244	OK	0.000	2433024	37664	37671
245	OK	0.000	2449408	39108	39115
246	OK	0.000	2449408	39190	39197
247	OK	0.015	3100672	183695	183703
248	OK	0.015	3100672	184258	184266
249	OK	0.015	3104768	185065	185073
250	OK	0.031	3104768	185428	185436
251	OK	0.031	3104768	185741	185749
252	OK	0.093	13754368	1900094	1900102
253	OK	0.093	13709312	1860225	1860233
254	OK	0.093	13746176	1899455	1899463
255	OK	0.078	13717504	1861088	1861096
256	OK	0.093	13795328	1942127	1942135
257	OK	0.093	13783040	1930200	1930208
258	OK	0.125	13717504	1861244	1861252
259	OK	0.171	22417408	3510448	3510457
260	OK	0.171	22556672	3650901	3650910
261	OK	0.171	22458368	3552374	3552383
262	OK	0.156	22335488	3435983	3435992
263	OK	0.171	22466560	3562689	3562698
264	OK	0.156	22417408	3521159	3521168
265	OK	0.171	22433792	3539149	3539158
266	OK	0.187	24621056	3985264	3985273

267	OK	0.187	24506368	3866892	3866901
268	OK	0.187	24580096	3942753	3942762
269	OK	0.187	24461312	3824263	3824272
270	OK	0.187	24645632	4011957	4011966
271	OK	0.187	24592384	3955420	3955429
272	OK	0.187	24584192	3946583	3946592
273	OK	0.203	24539136	3891536	3891545

Удаление из АВЛ-дерева

Условие

Удаление из АВЛ-дерева вершины с ключом X, при условии ее наличия, осуществляется следующим образом:

- путем спуска от корня и проверки ключей находится V — удаляемая вершина;
- если вершина V — лист (то есть, у нее нет детей):
  - удаляем вершину;
  - поднимаемся к корню, начиная с бывшего родителя вершины V, при этом если встречается несбалансированная вершина, то производим поворот.
- если у вершины V не существует левого ребенка:
  - следовательно, баланс вершины равен единице и ее правый ребенок — лист;
  - заменяем вершину V ее правым ребенком;
  - поднимаемся к корню, производя, где необходимо, балансировку.
- иначе:
  - находим R — самую правую вершину в левом поддереве;
  - переносим ключ вершины R в вершину V;
  - удаляем вершину R (у нее нет правого ребенка, поэтому она либо лист, либо имеет левого ребенка, являющегося листом);
  - поднимаемся к корню, начиная с бывшего родителя вершины R, производя балансировку.

Исключением является случай, когда производится удаление из дерева, состоящего из одной вершины — корня. Результатом удаления в этом случае будет пустое дерево.

Указанный алгоритм не является единственно возможным, но мы просим Вас реализовать именно его, так как тестирующая система проверяет точное равенство получающихся деревьев.

Формат входного файла

Входной файл содержит описание двоичного дерева, а также ключа вершины, которую требуется удалить из дерева.

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 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.

В последней строке содержится число X $(|X| \le 10^9)$ — ключ вершины, которую требуется удалить из дерева. Гарантируется, что такая вершина в дереве существует.

Формат выходного файла

Выведите в том же формате дерево после осуществления операции удаления. Нумерация вершин может быть произвольной при условии соблюдения формата.

Пример

input.txt	output.txt
3 4 2 3 3 0 0 5 0 0 4	2 3 0 2 5 0 0

Решение

openedu/week7/lab7_4.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <list>
#include <assert.h>
#include <tuple>

using namespace std;

typedef int32_t balance_t;

struct Node {
	int key;
	balance_t balance = 0;
	int height = 1;

	Node* parent = nullptr;
	Node* left = nullptr;
	Node* right = nullptr;

	// dirty hack
	size_t latest_population_descr_index;

	Node(int key) : key(key) {}

	balance_t get_left_balance() { return (left) ? left->balance : -1; };
	balance_t get_right_balance() { return (right) ? right->balance : -1; };
```

```
	int get_left_height() { return (left) ? left->height : 0; };
	int get_right_height() { return (right) ? right->height : 0; };

	void replace_child(Node* from, Node* to) {
		if (left == from) {
			left = to;
		} else {
			assert(right == from);
			right = to;
		}
		if (to) {
			to->parent = this;
		}
		from->parent = nullptr;
	}

	void subtree_change_height(Node* from, int dh = 1) {
		if (left == from) {
			balance -= dh;
		} else {
			assert(right == from);
			balance += dh;
		}
		height += dh;
	}

	void recalc_structure(int trust_deep = 3) {
		if (trust_deep) {
			if (left) left->recalc_structure(trust_deep - 1);
			if (right) right->recalc_structure(trust_deep - 1);
			height = max(get_right_height(), get_left_height()) + 1;
			balance = get_right_height() - get_left_height();
		}
	}
};

struct NodeDescription {
	int k;
	int li = 0;
	int ri = 0;
};

struct TreeDescription {
	vector<NodeDescription> descriptions;

	TreeDescription() {}
	TreeDescription(int size) : descriptions(size) {}

	NodeDescription& description_for_index(size_t index) {
		return descriptions[index];
	}

	void populate_from_tree(Node* tree) {
		// children set later
		descriptions.push_back(NodeDescription{ tree->key });

		// dirty hack for setting children!
		size_t current_index = descriptions.size() - 1;
		tree->latest_population_descr_index = current_index;

		// setting children of previously added things
		if (tree->parent) {
			if (tree->parent->left == tree) {
				descriptions[tree->parent-
>latest_population_descr_index].li = current_index + 1;
			} else {
				assert(tree->parent->right == tree);
```

```
                        descriptions[tree->parent-
>latest_population_descr_index].ri = current_index + 1;
                }
        }

        // recursively populate from subtrees
        if (tree->left) {
                populate_from_tree(tree->left);
        }

        if (tree->right) {
                populate_from_tree(tree->right);
        }
    }
};

Node* build_tree(TreeDescription &input, size_t root_index = 0) {
    // root node description
    auto& rd = input.description_for_index(root_index);

    // create new node
    Node* tree = new Node(rd.k);

    // recursively build left and right subtrees if needed
    if (rd.li) {
        tree->left = build_tree(input, rd.li - 1);
        tree->left->parent = tree;
    }

    if (rd.ri) {
        tree->right = build_tree(input, rd.ri - 1);
        tree->right->parent = tree;
    }

    // calculate height and balance
    tree->height = max(tree->get_right_height(), tree->get_left_height()) + 1;
    tree->balance = tree->get_right_height() - tree->get_left_height();

    return tree;
}

Node*& find_node(Node*& tree, int key) {
    if (tree == nullptr) {
        return tree;
    } else if (key == tree->key) {
        return tree;
    } else if (key < tree->key) {
        return find_node(tree->left, key);
    } else {
        return find_node(tree->right, key);
    }
}

void rotate_l(Node*& tree) {
    // малый левый
    assert(tree->right != nullptr);
    Node* a = tree;
    Node* b = a->right;
    Node* x = a->left;
    Node* y = b->left;
    Node* z = b->right;

    if (a->parent) {
        a->parent->replace_child(a, b);
    } else {
        b->parent = nullptr;
    }
```

```
	b->left = a;
	a->parent = b;

	a->right = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
	tree->recalc_structure();
}

void rotate_r(Node*& tree) {
	// малый правый
	assert(tree->left != nullptr);
	Node* a = tree;
	Node* b = a->left;
	Node* x = a->right;
	Node* y = b->right;
	Node* z = b->left;

	if (a->parent) {
		a->parent->replace_child(a, b);
	} else {
		b->parent = nullptr;
	}

	b->right = a;
	a->parent = b;

	a->left = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
	tree->recalc_structure();
}

void rotate_lr(Node*& tree) {
	// большой правый
	rotate_l(tree->left);
	rotate_r(tree);
}

void rotate_rl(Node*& tree) {
	// большой левый
	rotate_r(tree->right);
	rotate_l(tree);
}

void balance_tree(Node*& tree) {
	if (tree->balance == 2) {
		if (tree->right->balance == -1) {
			rotate_rl(tree);
		} else {
			rotate_l(tree);
		}
	} else if (tree->balance == -2) {
		if (tree->left->balance == 1) {
			rotate_lr(tree);
		} else {
			rotate_r(tree);
		}
	}
}
```

```
bool insert_to_tree(Node*& tree, Node* node) {
	if (tree == nullptr) {
		tree = node;
		return true;
	}

	bool balancing_done = false;
	// digging into the tree to find a right place for a node
	if (node->key == tree->key) {
		// assert(false && "key already in tree");
		return true;
	} else if (node->key < tree->key) {
		if (tree->left) {
			balancing_done = insert_to_tree(tree->left, node);
		} else {
			tree->left = node;
			node->parent = tree;
			tree->subtree_change_height(node, 1);
		}
	} else {
		if (tree->right) {
			balancing_done = insert_to_tree(tree->right, node);
		} else {
			tree->right = node;
			node->parent = tree;
			tree->subtree_change_height(node, 1);
		}
	}

	// balancing on our way back if needed
	balance_tree(tree);

	// recalculate balances due to height change
	// if node has a parent and subtree's height was increased (balance != 0)
	if (tree->parent && tree->balance != 0 && !balancing_done) {
		tree->parent->subtree_change_height(tree, 1);
		return false;
	}
	return true;
}

void delete_node_simple(Node*& tree) {
	if (tree->left == nullptr && tree->right == nullptr) {
		if (tree->parent) {
			tree->parent->subtree_change_height(tree, -1);
			// tree->parent->replace_child(tree, nullptr);
		}
		delete tree;
		tree = nullptr;
	} else if (tree->left == nullptr || tree->right == nullptr) {
		Node* saved = (tree->right) ? tree->right : tree->left;
		if (tree->parent) {
			tree->parent->subtree_change_height(tree, -1);
			// tree->parent->replace_child(tree, saved);
		}
		saved->parent = tree->parent;
		delete tree;
		tree = saved;
	} else {
		assert(false && "not a simple node to delete!");
	}
}


bool delete_from_tree(Node*& tree, int key, Node* target = nullptr) {
	if (tree == nullptr) {
```

```
			return true;
		}

		if (!tree->parent && !tree->left && !tree->right) {
			if (key == tree->key) {
				delete_node_simple(tree);
				return false;
			} else {
				// assert(false && "key not in a tree!");
				return true;
			}
		}

		bool balancing_done = false;
		if (!target) {
			// digging into the tree
			if (key == tree->key) {
				if (!tree->left && !tree->right) {
					delete_node_simple(tree);
					return false;
				} else if (!tree->left) {
					delete_node_simple(tree);
					return false;
				} else {
					balancing_done = delete_from_tree(tree->left, key,
tree);
				}
			} else if (key < tree->key) {
				balancing_done = delete_from_tree(tree->left, key);
			} else {
				assert(key > tree->key);
				balancing_done = delete_from_tree(tree->right, key);
			}
		} else {
			if (tree->right) {
				balancing_done = delete_from_tree(tree->right, key, target);
			} else {
				// found prev
				target->key = tree->key;
				delete_node_simple(tree);
				return false;
			}
		}

		// balancing on our way back if needed
		balance_tree(tree);

		// recalculate balances due to height change
		// if node has a parent and subtree's height was DECREASED (balance == 0)
		if (tree->parent && tree->balance == 0 && !balancing_done) {
			tree->parent->subtree_change_height(tree, -1);
			return false;
		}
		return true;
}

int main() {
		int n;
		io >> n;

		TreeDescription input(n);
		for (auto& descr : input.descriptions) {
			io >> descr.k >> descr.li >> descr.ri;
		}

		Node* tree = nullptr;
		if (n) {
```

```
		tree = build_tree(input);
	}

	int k;
	io >> k;
	delete_from_tree(tree, k);

	if (tree == nullptr) {
		io << "0";
		return 0;
	}

	TreeDescription output;
	output.populate_from_tree(tree);

	io << output.descriptions.size() << "\n";
	for (auto& descr : output.descriptions) {
		io << descr.k << " " << descr.li << " " << descr.ri << "\n";
	}

	return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.218	24723456	4077288	4077255
1	OK	0.000	2277376	27	17
2	OK	0.015	2269184	13	1
3	OK	0.000	2256896	20	10
4	OK	0.015	2269184	20	10
5	OK	0.015	2281472	20	10
6	OK	0.000	2269184	20	10
7	OK	0.015	2269184	27	17
8	OK	0.000	2269184	27	17
9	OK	0.000	2277376	27	17
10	OK	0.015	2260992	34	24
11	OK	0.015	2265088	34	24
12	OK	0.000	2269184	34	24
13	OK	0.000	2265088	34	24
14	OK	0.015	2269184	34	24
15	OK	0.000	2269184	34	24
16	OK	0.000	2269184	34	24
17	OK	0.000	2269184	34	24
18	OK	0.015	2285568	34	24
19	OK	0.000	2281472	34	24
20	OK	0.000	2260992	34	24
21	OK	0.000	2269184	34	24
22	OK	0.000	2252800	34	24
23	OK	0.000	2265088	34	24

24	OK	0.000	2269184	34	24
25	OK	0.015	2269184	34	24
26	OK	0.000	2269184	41	31
27	OK	0.000	2265088	41	31
28	OK	0.015	2269184	41	31
29	OK	0.000	2273280	41	31
30	OK	0.000	2269184	41	31
31	OK	0.000	2281472	41	31
32	OK	0.000	2256896	41	31
33	OK	0.000	2281472	41	31
34	OK	0.015	2260992	41	31
35	OK	0.000	2265088	41	31
36	OK	0.000	2269184	41	31
37	OK	0.015	2265088	41	31
38	OK	0.015	2260992	41	31
39	OK	0.000	2269184	41	31
40	OK	0.000	2281472	41	31
41	OK	0.000	2256896	41	31
42	OK	0.000	2269184	41	31
43	OK	0.000	2269184	41	31
44	OK	0.015	2269184	41	31
45	OK	0.000	2285568	41	31
46	OK	0.000	2269184	41	31
47	OK	0.000	2269184	41	31
48	OK	0.000	2260992	41	31
49	OK	0.000	2265088	41	31
50	OK	0.015	2269184	41	31
51	OK	0.000	2252800	41	31
52	OK	0.015	2265088	41	31
53	OK	0.000	2269184	41	31
54	OK	0.031	2281472	41	31
55	OK	0.031	2269184	41	31
56	OK	0.000	2265088	48	38
57	OK	0.000	2269184	48	38
58	OK	0.000	2269184	48	38
59	OK	0.000	2269184	48	38
60	OK	0.015	2269184	48	38
61	OK	0.000	2256896	48	38
62	OK	0.000	2281472	48	38
63	OK	0.000	2281472	48	38
64	OK	0.015	2269184	48	38
65	OK	0.000	2269184	48	38

66	OK	0.000	2269184	48	38
67	OK	0.015	2281472	48	38
68	OK	0.000	2260992	48	38
69	OK	0.015	2269184	48	38
70	OK	0.000	2265088	48	38
71	OK	0.015	2252800	48	38
72	OK	0.015	2281472	48	38
73	OK	0.000	2281472	48	38
74	OK	0.000	2269184	48	38
75	OK	0.000	2269184	48	38
76	OK	0.000	2269184	48	38
77	OK	0.015	2269184	48	38
78	OK	0.000	2260992	48	38
79	OK	0.000	2269184	48	38
80	OK	0.000	2269184	55	45
81	OK	0.031	2269184	55	45
82	OK	0.015	2269184	55	45
83	OK	0.000	2265088	55	45
84	OK	0.000	2269184	55	45
85	OK	0.000	2269184	55	45
86	OK	0.015	2265088	55	45
87	OK	0.015	2281472	55	45
88	OK	0.015	2260992	55	45
89	OK	0.000	2269184	55	45
90	OK	0.000	2277376	55	45
91	OK	0.031	2256896	55	45
92	OK	0.015	2277376	55	45
93	OK	0.000	2265088	55	45
94	OK	0.015	2281472	55	45
95	OK	0.000	2281472	55	45
96	OK	0.000	2269184	55	45
97	OK	0.000	2269184	55	45
98	OK	0.015	2273280	55	45
99	OK	0.015	2269184	55	45
100	OK	0.015	2265088	55	45
101	OK	0.000	2256896	55	45
102	OK	0.000	2269184	55	45
103	OK	0.000	2281472	55	45
104	OK	0.000	2269184	55	45
105	OK	0.000	2281472	55	45
106	OK	0.000	2281472	55	45
107	OK	0.000	2281472	55	45

108	OK	0.000	2273280	55	45
109	OK	0.000	2265088	55	45
110	OK	0.000	2269184	55	45
111	OK	0.015	2269184	55	45
112	OK	0.000	2269184	55	45
113	OK	0.000	2265088	55	45
114	OK	0.000	2265088	55	45
115	OK	0.015	2281472	55	45
116	OK	0.000	2265088	55	45
117	OK	0.015	2269184	55	45
118	OK	0.015	2273280	55	45
119	OK	0.015	2265088	55	45
120	OK	0.015	2269184	55	45
121	OK	0.000	2269184	55	45
122	OK	0.000	2269184	55	45
123	OK	0.015	2269184	55	45
124	OK	0.000	2269184	55	45
125	OK	0.000	2269184	55	45
126	OK	0.015	2269184	55	45
127	OK	0.000	2269184	55	45
128	OK	0.000	2273280	55	45
129	OK	0.015	2265088	55	45
130	OK	0.000	2252800	55	45
131	OK	0.000	2269184	55	45
132	OK	0.000	2269184	55	45
133	OK	0.015	2281472	55	45
134	OK	0.015	2269184	55	45
135	OK	0.000	2269184	55	45
136	OK	0.000	2285568	55	45
137	OK	0.000	2265088	55	45
138	OK	0.015	2260992	55	45
139	OK	0.000	2269184	55	45
140	OK	0.000	2256896	55	45
141	OK	0.000	2281472	55	45
142	OK	0.015	2281472	55	45
143	OK	0.000	2281472	55	45
144	OK	0.015	2265088	55	45
145	OK	0.000	2265088	55	45
146	OK	0.000	2269184	55	45
147	OK	0.000	2269184	55	45
148	OK	0.000	2260992	55	45
149	OK	0.000	2269184	55	45

150	OK	0.015	2273280	55	45
151	OK	0.000	2269184	55	45
152	OK	0.000	2269184	55	45
153	OK	0.000	2269184	55	45
154	OK	0.015	2269184	55	45
155	OK	0.000	2269184	55	45
156	OK	0.000	2265088	55	45
157	OK	0.000	2285568	55	45
158	OK	0.000	2256896	55	45
159	OK	0.000	2269184	55	45
160	OK	0.000	2269184	55	45
161	OK	0.000	2281472	55	45
162	OK	0.000	2269184	55	45
163	OK	0.015	2265088	55	45
164	OK	0.000	2269184	55	45
165	OK	0.000	2265088	55	45
166	OK	0.015	2269184	55	45
167	OK	0.000	2269184	55	45
168	OK	0.000	2260992	55	45
169	OK	0.015	2269184	55	45
170	OK	0.000	2252800	55	45
171	OK	0.015	2269184	55	45
172	OK	0.000	2269184	55	45
173	OK	0.000	2269184	55	45
174	OK	0.000	2269184	55	45
175	OK	0.000	2269184	55	45
176	OK	0.000	2269184	55	45
177	OK	0.000	2269184	55	45
178	OK	0.000	2277376	55	45
179	OK	0.000	2269184	55	45
180	OK	0.000	2256896	55	45
181	OK	0.000	2269184	55	45
182	OK	0.000	2281472	55	45
183	OK	0.000	2269184	55	45
184	OK	0.000	2260992	55	45
185	OK	0.000	2269184	55	45
186	OK	0.000	2281472	55	45
187	OK	0.000	2269184	55	45
188	OK	0.000	2273280	55	45
189	OK	0.000	2277376	55	45
190	OK	0.000	2269184	55	45
191	OK	0.000	2265088	55	45

192	OK	0.000	2281472	55	45
193	OK	0.000	2269184	55	45
194	OK	0.015	2269184	55	45
195	OK	0.000	2277376	55	45
196	OK	0.000	2269184	55	45
197	OK	0.000	2269184	55	45
198	OK	0.000	2260992	55	45
199	OK	0.015	2273280	239	210
200	OK	0.000	2277376	235	208
201	OK	0.015	2301952	1797	1769
202	OK	0.000	2285568	1809	1781
203	OK	0.015	2285568	1831	1803
204	OK	0.015	2338816	9625	9597
205	OK	0.000	2342912	10026	9996
206	OK	0.000	2355200	9672	9642
207	OK	0.000	2453504	39459	39428
208	OK	0.015	2441216	39672	39641
209	OK	0.000	2465792	38780	38749
210	OK	0.000	2441216	38392	38361
211	OK	0.000	3104768	179425	179394
212	OK	0.015	3108864	182878	182845
213	OK	0.015	3125248	185986	185953
214	OK	0.000	3112960	186275	186244
215	OK	0.015	3104768	179082	179051
216	OK	0.078	13774848	1912349	1912319
217	OK	0.093	13729792	1864033	1864003
218	OK	0.093	13770752	1913940	1913908
219	OK	0.078	13742080	1879988	1879958
220	OK	0.093	13742080	1887512	1887482
221	OK	0.093	13795328	1932558	1932526
222	OK	0.078	13737984	1875036	1875006
223	OK	0.203	22511616	3597394	3597361
224	OK	0.171	22482944	3569973	3569940
225	OK	0.171	22425600	3512224	3512193
226	OK	0.171	22536192	3624329	3624296
227	OK	0.156	22437888	3523352	3523321
228	OK	0.171	22544384	3638077	3638044
229	OK	0.156	22425600	3512844	3512813
230	OK	0.187	24637440	4001320	4001287
231	OK	0.171	24600576	3954282	3954249
232	OK	0.187	24555520	3907814	3907783
233	OK	0.187	24637440	3983014	3982983

234	OK	0.187	24674304	4029895	4029862
235	OK	0.218	24690688	4046188	4046157
236	OK	0.187	24723456	4077288	4077255
237	OK	0.187	24547328	3902092	3902061

Упорядоченное множество на АВЛ-дереве

Условие

Если Вы сдали все предыдущие задачи, Вы уже можете написать эффективную реализацию упорядоченного множества на АВЛ-дереве. Сделайте это.

Для проверки того, что множество реализовано именно на АВЛ-дереве, мы просим Вас выводить баланс корня после каждой операции вставки и удаления.

Операции вставки и удаления требуется реализовать точно так же, как это было сделано в предыдущих двух задачах, потому что в ином случае баланс корня может отличаться от требуемого.

Формат входного файла

В первой строке файла находится число N ($1 \le N \le 2 \cdot 10^5$) — число операций над множеством. Изначально множество пусто. В каждой из последующих N строк файла находится описание операции.

Операции бывают следующих видов:

- `A x` — вставить число x в множество. Если число x там уже содержится, множество изменять не следует.
- `D x` — удалить число x из множества. Если числа x нет в множестве, множество изменять не следует.
- `C x` — проверить, есть ли число x в множестве.

Формат выходного файла

Для каждой операции вида `C x` выведите `Y`, если число x содержится в множестве, и `N`, если не содержится.

Для каждой операции вида `A x` или `D x` выведите баланс корня дерева после выполнения операции. Если дерево пустое (в нем нет вершин), выведите 0.

Вывод для каждой операции должен содержаться на отдельной строке.

Пример

input.txt	output.txt
6 A 3 A 4 A 5 C 4 C 6 D 5	0 1 0 Y N -1

Решение

openedu/week7/lab7_5.cpp

```
#include "edx-io.hpp"
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <string>
#include <list>
#include <assert.h>
#include <tuple>

using namespace std;

typedef int32_t balance_t;

struct Node {
	int key;
	balance_t balance = 0;
	int height = 1;

	Node* parent = nullptr;
	Node* left = nullptr;
	Node* right = nullptr;

	// dirty hack
	size_t latest_population_descr_index;

	Node(int key) : key(key) {}

	balance_t get_left_balance() { return (left) ? left->balance : -1; };
	balance_t get_right_balance() { return (right) ? right->balance : -1; };

	int get_left_height() { return (left) ? left->height : 0; };
	int get_right_height() { return (right) ? right->height : 0; };

	void replace_child(Node* from, Node* to) {
		if (left == from) {
			left = to;
		} else {
			assert(right == from);
			right = to;
		}
		if (to) {
			to->parent = this;
		}
		from->parent = nullptr;
	}

	void subtree_change_height(Node* from, int dh = 1) {
		if (left == from) {
			balance -= dh;
		} else {
			assert(right == from);
			balance += dh;
		}
		height += dh;
	}

	void recalc_structure(int trust_deep = 30) {
		if (trust_deep) {
			if (left) left->recalc_structure(trust_deep - 1);
			if (right) right->recalc_structure(trust_deep - 1);
			height = max(get_right_height(), get_left_height()) + 1;
```

```
                balance = get_right_height() - get_left_height();
            }
        }
};

struct NodeDescription {
        int k;
        int li = 0;
        int ri = 0;
};

struct TreeDescription {
        vector<NodeDescription> descriptions;

        TreeDescription() {}
        TreeDescription(int size) : descriptions(size) {}

        NodeDescription& description_for_index(size_t index) {
                return descriptions[index];
        }

        void populate_from_tree(Node* tree) {
                // children set later
                descriptions.push_back(NodeDescription{ tree->key });

                // dirty hack for setting children!
                size_t current_index = descriptions.size() - 1;
                tree->latest_population_descr_index = current_index;

                // setting children of previously added things
                if (tree->parent) {
                        if (tree->parent->left == tree) {
                                descriptions[tree->parent-
>latest_population_descr_index].li = current_index + 1;
                        } else {
                                assert(tree->parent->right == tree);
                                descriptions[tree->parent-
>latest_population_descr_index].ri = current_index + 1;
                        }
                }

                // recursively populate from subtrees
                if (tree->left) {
                        populate_from_tree(tree->left);
                }

                if (tree->right) {
                        populate_from_tree(tree->right);
                }
        }
};

Node* build_tree(TreeDescription &input, size_t root_index = 0) {
        // root node description
        auto& rd = input.description_for_index(root_index);

        // create new node
        Node* tree = new Node(rd.k);

        // recursively build left and right subtrees if needed
        if (rd.li) {
                tree->left = build_tree(input, rd.li - 1);
                tree->left->parent = tree;
        }

        if (rd.ri) {
                tree->right = build_tree(input, rd.ri - 1);
```

```
		tree->right->parent = tree;
	}

	// calculate height and balance
	tree->height = max(tree->get_right_height(), tree->get_left_height()) + 1;
	tree->balance = tree->get_right_height() - tree->get_left_height();

	return tree;
}

Node*& find_node(Node*& tree, int key) {
	if (tree == nullptr) {
		return tree;
	} else if (key == tree->key) {
		return tree;
	} else if (key < tree->key) {
		return find_node(tree->left, key);
	} else {
		return find_node(tree->right, key);
	}
}

void rotate_l(Node*& tree) {
	// малый левый
	assert(tree->right != nullptr);
	Node* a = tree;
	Node* b = a->right;
	Node* x = a->left;
	Node* y = b->left;
	Node* z = b->right;

	if (a->parent) {
		a->parent->replace_child(a, b);
	} else {
		b->parent = nullptr;
	}

	b->left = a;
	a->parent = b;

	a->right = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
	tree->recalc_structure();
}

void rotate_r(Node*& tree) {
	// малый правый
	assert(tree->left != nullptr);
	Node* a = tree;
	Node* b = a->left;
	Node* x = a->right;
	Node* y = b->right;
	Node* z = b->left;

	if (a->parent) {
		a->parent->replace_child(a, b);
	} else {
		b->parent = nullptr;
	}

	b->right = a;
	a->parent = b;
```

```
	a->left = y;
	if (y) {
		y->parent = a;
	}

	tree = b;
	tree->recalc_structure();
}

void rotate_lr(Node*& tree) {
	// большой правый
	rotate_l(tree->left);
	rotate_r(tree);
}

void rotate_rl(Node*& tree) {
	// большой левый
	rotate_r(tree->right);
	rotate_l(tree);
}

void balance_tree(Node*& tree) {
	if (tree->balance == 2) {
		if (tree->right->balance == -1) {
			rotate_rl(tree);
		} else {
			rotate_l(tree);
		}
	} else if (tree->balance == -2) {
		if (tree->left->balance == 1) {
			rotate_lr(tree);
		} else {
			rotate_r(tree);
		}
	}
}

bool insert_to_tree(Node*& tree, Node* node) {
	if (tree == nullptr) {
		tree = node;
		return true;
	}

	bool balancing_done = false;
	// digging into the tree to find a right place for a node
	if (node->key == tree->key) {
		// assert(false && "key already in tree");
		return true;
	} else if (node->key < tree->key) {
		if (tree->left) {
			balancing_done = insert_to_tree(tree->left, node);
		} else {
			tree->left = node;
			node->parent = tree;
			tree->subtree_change_height(node, 1);
		}
	} else {
		if (tree->right) {
			balancing_done = insert_to_tree(tree->right, node);
		} else {
			tree->right = node;
			node->parent = tree;
			tree->subtree_change_height(node, 1);
		}
	}

	// balancing on our way back if needed
```

```
	balance_tree(tree);

	// recalculate balances due to height change
	// if node has a parent and subtree's height was increased (balance != 0)
	if (tree->parent && tree->balance != 0 && !balancing_done) {
		tree->parent->subtree_change_height(tree, 1);
		return false;
	}
	return true;
}

void delete_node_simple(Node*& tree) {
	if (tree->left == nullptr && tree->right == nullptr) {
		if (tree->parent) {
			tree->parent->subtree_change_height(tree, -1);
			// tree->parent->replace_child(tree, nullptr);
		}
		delete tree;
		tree = nullptr;
	} else if (tree->left == nullptr || tree->right == nullptr) {
		Node* saved = (tree->right) ? tree->right : tree->left;
		if (tree->parent) {
			tree->parent->subtree_change_height(tree, -1);
			// tree->parent->replace_child(tree, saved);
		}
		saved->parent = tree->parent;
		delete tree;
		tree = saved;
	} else {
		assert(false && "not a simple node to delete!");
	}
}


bool delete_from_tree(Node*& tree, int key, Node* target = nullptr) {
	if (tree == nullptr) {
		return true;
	}

	if (!tree->parent && !tree->left && !tree->right) {
		if (key == tree->key) {
			delete_node_simple(tree);
			return false;
		} else {
			// assert(false && "key not in a tree!");
			return true;
		}
	}

	bool balancing_done = false;
	if (!target) {
		// digging into the tree
		if (key == tree->key) {
			if (!tree->left && !tree->right) {
				delete_node_simple(tree);
				return false;
			} else if (!tree->left) {
				delete_node_simple(tree);
				return false;
			} else {
				balancing_done = delete_from_tree(tree->left, key,
tree);
			}
		} else if (key < tree->key) {
			balancing_done = delete_from_tree(tree->left, key);
		} else {
			assert(key > tree->key);
```

```
                        balancing_done = delete_from_tree(tree->right, key);
                }
        } else {
                if (tree->right) {
                        balancing_done = delete_from_tree(tree->right, key, target);
                } else {
                        // found prev
                        target->key = tree->key;
                        delete_node_simple(tree);
                        return false;
                }
        }

        // balancing on our way back if needed
        balance_tree(tree);

        // recalculate balances due to height change
        // if node has a parent and subtree's height was DECREASED (balance == 0)
        if (tree->parent && tree->balance == 0 && !balancing_done) {
                tree->parent->subtree_change_height(tree, -1);
                return false;
        }
        return true;
}

int main() {
        int n;
        io >> n;

        Node* tree = nullptr;

        char c;
        int arg;
        while (n--) {
                io >> c >> arg;

                switch (c) {
                        case 'A': {
                                insert_to_tree(tree, new Node(arg));
                                io << ((tree) ? tree->balance : 0);
                        } break;
                        case 'D': {
                                delete_from_tree(tree, arg);
                                io << ((tree) ? tree->balance : 0);
                        } break;
                        case 'C': {
                                io << ((find_node(tree, arg)) ? 'Y' : 'N');
                        } break;
                }
                io << '\n';
        }

        return 0;
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.187	17522688	2678110	731071
1	OK	0.000	2256896	33	19
2	OK	0.000	2256896	114	66
3	OK	0.000	2256896	154	90

4	OK	0.000	2244608	154	91
5	OK	0.000	2256896	154	90
6	OK	0.000	2252800	154	95
7	OK	0.015	2252800	154	91
8	OK	0.015	2277376	154	94
9	OK	0.000	2244608	154	95
10	OK	0.000	2252800	154	90
11	OK	0.000	2248704	154	90
12	OK	0.015	2256896	154	90
13	OK	0.015	2256896	154	95
14	OK	0.000	2256896	154	97
15	OK	0.015	2277376	154	94
16	OK	0.015	2269184	154	93
17	OK	0.000	2256896	154	90
18	OK	0.000	2256896	154	90
19	OK	0.000	2269184	154	98
20	OK	0.015	2256896	154	93
21	OK	0.000	2260992	154	92
22	OK	0.000	2269184	154	98
23	OK	0.031	5079040	1000008	616458
24	OK	0.046	5079040	1000008	622272
25	OK	0.031	5152768	1000008	625335
26	OK	0.031	5079040	1000008	628546
27	OK	0.046	5079040	1000008	631472
28	OK	0.046	5079040	1000008	632217
29	OK	0.046	5074944	1000008	631772
30	OK	0.031	5079040	1000008	631071
31	OK	0.046	5074944	1000008	630132
32	OK	0.046	5169152	1017957	630451
33	OK	0.062	5079040	1000008	616595
34	OK	0.046	5079040	1000008	622199
35	OK	0.046	5079040	1000008	625057
36	OK	0.046	5079040	1000008	628040
37	OK	0.046	5079040	1000008	631495
38	OK	0.046	5079040	1000008	632086
39	OK	0.046	5079040	1000008	631753
40	OK	0.046	5074944	1000008	630849
41	OK	0.046	5074944	1000008	630110
42	OK	0.046	5095424	1018151	630800
43	OK	0.000	2281472	756	369
44	OK	0.000	2285568	758	432
45	OK	0.000	2265088	1659	408

46	OK	0.000	2269184	723	383
47	OK	0.000	2281472	723	385
48	OK	0.000	2265088	723	415
49	OK	0.000	2256896	723	415
50	OK	0.000	2281472	1668	377
51	OK	0.000	2277376	1660	396
52	OK	0.000	2326528	5348	2337
53	OK	0.015	2326528	5350	2848
54	OK	0.015	2338816	10439	2648
55	OK	0.015	2285568	5238	2343
56	OK	0.015	2285568	5238	2465
57	OK	0.000	2285568	5238	2719
58	OK	0.000	2297856	5238	2719
59	OK	0.000	2273280	10450	2421
60	OK	0.000	2293760	10439	2405
61	OK	0.000	2600960	32784	12708
62	OK	0.015	2600960	32787	14896
63	OK	0.000	2592768	56716	12715
64	OK	0.015	2359296	31674	12778
65	OK	0.000	2322432	31674	13220
66	OK	0.000	2318336	31674	14383
67	OK	0.000	2359296	31674	14825
68	OK	0.000	2318336	56748	13671
69	OK	0.000	2367488	56716	13193
70	OK	0.015	3616768	162462	57855
71	OK	0.046	3637248	162466	68948
72	OK	0.015	3629056	258205	71756
73	OK	0.015	2609152	152067	59306
74	OK	0.015	2646016	152067	59903
75	OK	0.015	2646016	152067	66900
76	OK	0.015	2613248	152067	67497
77	OK	0.015	2777088	258312	70001
78	OK	0.015	2732032	258332	58111
79	OK	0.062	8609792	811002	274035
80	OK	0.062	8613888	811006	332612
81	OK	0.078	9027584	1222794	299942
82	OK	0.046	5287936	799892	286940
83	OK	0.046	5316608	799892	282227
84	OK	0.062	5320704	799892	324420
85	OK	0.062	5287936	799892	319707
86	OK	0.062	6053888	1222871	284516
87	OK	0.046	5103616	1223246	288111

88	OK	0.156	16736256	1888898	600000
89	OK	0.156	16732160	1888903	731071
90	OK	0.187	17522688	2677526	600067
91	OK	0.125	9334784	1777788	601696
92	OK	0.125	9367552	1777788	632768
93	OK	0.125	9359360	1777788	698302
94	OK	0.125	9326592	1777788	698303
95	OK	0.156	11026432	2678110	611713
96	OK	0.109	8863744	2677266	600286