Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Алгоритмы и структуры данных»

ОТЧЁТ

по лабораторной работе №7 (Week 7 Openedu)

Студент Дунаев Алексей Игоревич

Группа P3217

Преподаватель Муромцев Дмитрий Ильич

Санкт-Петербург

2019 г.

Содержание

Задача 1 Проверка сбалансированности

1.0 из 1.0 балла (оценивается)

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

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 ($1 \le N \le 2 * 10^5$ — 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 $-10^9 \le K \le 10^9$, номера левого ребенка i-ой вершины (I < Li ≤ N или Li = 0, если левого ребенка нет) и номера правого ребенка i-ой вершины (I < R ≤ N или Ri = 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	3 -1

8 4 3 9 0 0 3 6 5 6 0 0 0 0 0	0 0 0 0

Исходный код к задаче 1

```
#include <iostream>
#include <string>
using namespace std;
/**
#define cin std::cin
#define cout std::cout
/*/
#include "edx-io.hpp"
#define cin io
#define cout io
/**/
struct t_node {
     int left, right, key;
} *tree;
int *keys, *h, *b;
int sz;
int cnt(int i) {
     int d = b[i] = 0;
     if (tree[i].left) {
          d = max(cnt(tree[i].left - 1), d);
          b[i] -= h[tree[i].left - 1];
     }
     if (tree[i].right) {
          d = max(cnt(tree[i].right - 1), d);
          b[i] += h[tree[i].right - 1];
     }
     return h[i] = (d + 1);
}
int find(int x) {
     int i = 0;
     while (tree[i].key != x) {
          if (x < tree[i].key) {
               if (tree[i].left) {
                    i = tree[i].left - 1;
               }
               else {
                    return -1;
               }
          }
          else {
               if (tree[i].right) {
                    i = tree[i].right - 1;
               }
               else {
                    return -1;
               }
          }
     }
     return i;
}
int main() {
     int n;
     cin >> n;
```

```
    tree = new t_node[sz = n];
    h = new int[n];
    b = new int[n];
    for (int i = 0; i < n; ++i) {
        cin >> tree[i].key >> tree[i].left >> tree[i].right;
        h[i] = 0;
    }
    cnt(0);
    for (int i = 0; i < n; ++i) {
        cout << b[i] << '\n';
    }
    return 0;
}
```

Бенчмарк к задаче 1

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.140	27299840	3986010	1688889
1	OK	0.031	3420160	46	19
2	OK	0.000	3416064	10	3
3	OK	0.000	3424256	17	6
4	OK	0.000	3424256	17	7
5	OK	0.015	3452928	24	9
6	OK	0.015	3420160	24	10
7	OK	0.000	3436544	24	9
8	OK	0.046	3432448	24	10
9	OK	0.015	3428352	24	11
10	OK	0.000	3436544	31	12
11	OK	0.015	3416064	31	13
12	OK	0.015	3444736	31	12
13	OK	0.000	3428352	31	13
14	OK	0.015	3420160	31	14
15	OK	0.000	3420160	31	12
16	OK	0.000	3428352	31	13
17	OK	0.000	3436544	31	13
18	OK	0.015	3428352	31	14

19	OK	0.000	3432448	31	13
20	OK	0.000	3420160	31	14
21	OK	0.000	3436544	31	13
22	OK	0.000	3424256	31	14
23	OK	0.015	3432448	31	15
24	OK	0.000	3436544	38	15
25	OK	0.000	3416064	38	16
26	OK	0.015	3420160	38	15
27	OK	0.000	3452928	38	16
28	OK	0.000	3424256	38	17
29	OK	0.000	3457024	38	15
30	OK	0.000	3432448	38	16
31	OK	0.000	3432448	38	16
32	OK	0.015	3411968	38	17
33	OK	0.015	3416064	38	16
34	OK	0.015	3407872	38	17
35	OK	0.015	3432448	38	16
36	OK	0.000	3416064	38	17
37	OK	0.046	3436544	38	18
38	OK	0.015	3432448	38	15
39	OK	0.015	3432448	38	16
40	OK	0.031	3420160	38	15
41	OK	0.015	3444736	38	16
42	OK	0.046	3440640	38	17
43	OK	0.015	3436544	38	15
44	OK	0.015	3428352	38	16
45	OK	0.000	3432448	38	16
46	OK	0.000	3440640	38	17
47	OK	0.000	3424256	38	16

48	OK	0.015	3424256	38	17
49	OK	0.000	3411968	38	16
50	OK	0.015	3420160	38	17
51	OK	0.000	3432448	38	18
52	OK	0.000	3420160	38	16
53	OK	0.000	3420160	38	17
54	OK	0.000	3432448	38	16
55	OK	0.015	3440640	38	17
56	OK	0.000	3457024	38	18
57	OK	0.015	3420160	38	16
58	OK	0.015	3440640	38	17
59	OK	0.000	3448832	38	17
60	OK	0.000	3432448	38	18
61	OK	0.000	3432448	38	17
62	OK	0.015	3428352	38	18
63	OK	0.015	3420160	38	17
64	OK	0.000	3432448	38	18
65	OK	0.000	3444736	38	19
66	OK	0.000	3448832	45	18
67	OK	0.000	3432448	45	19
68	OK	0.000	3448832	45	18
69	OK	0.000	3420160	45	19
70	OK	0.000	3440640	45	20
71	OK	0.015	3436544	45	18
72	OK	0.000	3424256	45	19
73	OK	0.000	3416064	45	19
74	OK	0.015	3444736	45	20
75	OK	0.015	3420160	45	19
76	OK	0.000	3416064	45	20

77	OK	0.000	3420160	45	19
78	OK	0.000	3416064	45	20
79	OK	0.000	3465216	45	21
80	OK	0.000	3428352	45	18
81	OK	0.015	3440640	45	19
82	OK	0.000	3424256	45	18
83	OK	0.015	3411968	45	19
84	OK	0.015	3411968	45	20
85	OK	0.015	3428352	45	18
86	OK	0.000	3416064	45	19
87	OK	0.015	3440640	45	19
88	OK	0.000	3428352	45	20
89	OK	0.000	3444736	45	19
90	OK	0.015	3432448	45	20
91	OK	0.000	3444736	45	19
92	OK	0.015	3436544	45	20
93	OK	0.000	3416064	45	21
94	OK	0.015	3416064	45	19
95	OK	0.015	3428352	45	20
96	OK	0.000	3420160	45	19
97	OK	0.015	3420160	45	20
98	OK	0.015	3448832	45	21
99	OK	0.015	3432448	45	19
100	OK	0.000	3420160	45	20
101	OK	0.015	3432448	45	20
102	OK	0.000	3416064	45	21
103	OK	0.000	3452928	45	20
104	OK	0.000	3428352	45	21
105	OK	0.015	3436544	45	20

106	OK	0.000	3411968	45	21
107	OK	0.000	3428352	45	22
108	OK	0.000	3424256	45	18
109	OK	0.000	3424256	45	19
110	OK	0.015	3432448	45	18
111	OK	0.000	3428352	45	19
112	OK	0.015	3432448	45	20
113	OK	0.015	3424256	45	18
114	OK	0.015	3424256	45	19
115	OK	0.000	3440640	45	19
116	OK	0.000	3424256	45	20
117	OK	0.000	3428352	45	19
118	OK	0.000	3407872	45	20
119	OK	0.015	3428352	45	19
120	OK	0.046	3424256	45	20
121	OK	0.000	3420160	45	21
122	OK	0.015	3436544	45	18
123	OK	0.000	3452928	45	19
124	OK	0.015	3407872	45	18
125	OK	0.031	3411968	45	19
126	OK	0.015	3452928	45	20
127	OK	0.015	3457024	45	19
128	OK	0.000	3411968	45	20
129	OK	0.015	3424256	45	19
130	OK	0.015	3411968	45	20
131	OK	0.015	3424256	45	21
132	OK	0.015	3444736	45	19
133	OK	0.000	3420160	45	20
134	OK	0.015	3420160	45	20

135	OK	0.000	3436544	45	21
136	OK	0.015	3436544	45	18
137	OK	0.015	3428352	45	19
138	OK	0.000	3448832	45	20
139	OK	0.000	3432448	45	21
140	OK	0.015	3424256	45	21
141	OK	0.000	3436544	45	22
142	OK	0.000	3432448	45	19
143	OK	0.000	3411968	45	20
144	OK	0.000	3420160	45	19
145	OK	0.000	3424256	45	20
146	OK	0.000	3424256	45	21
147	OK	0.015	3440640	45	19
148	OK	0.015	3432448	45	20
149	OK	0.015	3444736	45	20
150	OK	0.046	3436544	45	21
151	OK	0.015	3420160	45	20
152	OK	0.015	3432448	45	21
153	OK	0.000	3424256	45	20
154	OK	0.015	3436544	45	21
155	OK	0.031	3420160	45	22
156	OK	0.000	3420160	45	19
157	OK	0.000	3416064	45	20
158	OK	0.000	3420160	45	19
159	OK	0.015	3436544	45	20
160	OK	0.031	3420160	45	21
161	OK	0.015	3432448	45	19
162	OK	0.015	3432448	45	20
163	OK	0.015	3448832	45	20

164	OK	0.000	3420160	45	21
165	OK	0.046	3420160	45	20
166	OK	0.000	3416064	45	21
167	OK	0.000	3424256	45	20
168	OK	0.000	3448832	45	21
169	OK	0.000	3420160	45	22
170	OK	0.000	3432448	45	19
171	OK	0.000	3420160	45	20
172	OK	0.000	3416064	45	19
173	OK	0.000	3432448	45	20
174	OK	0.000	3432448	45	21
175	OK	0.000	3416064	45	19
176	OK	0.015	3432448	45	20
177	OK	0.015	3428352	45	20
178	OK	0.015	3416064	45	21
179	OK	0.000	3416064	45	20
180	OK	0.015	3444736	45	21
181	OK	0.015	3448832	45	20
182	OK	0.000	3432448	45	21
183	OK	0.000	3428352	45	22
184	OK	0.000	3411968	45	20
185	OK	0.000	3407872	45	21
186	OK	0.015	3461120	45	20
187	OK	0.000	3416064	45	21
188	OK	0.000	3436544	45	22
189	OK	0.031	3416064	45	20
190	OK	0.000	3440640	45	21
191	OK	0.015	3448832	45	21
192	OK	0.015	3436544	45	22

193	OK	0.000	3420160	45	21
194	OK	0.000	3416064	45	22
195	OK	0.015	3440640	45	21
196	OK	0.015	3420160	45	22
197	OK	0.015	3432448	45	23
198	OK	0.015	3457024	221	55
199	OK	0.000	3416064	220	59
200	OK	0.015	3416064	220	46
201	OK	0.000	3436544	223	48
202	OK	0.000	3432448	226	45
203	OK	0.031	3444736	1786	502
204	OK	0.000	3420160	1785	555
205	OK	0.000	3440640	1785	445
206	OK	0.000	3448832	1845	365
207	OK	0.000	3420160	1847	363
208	OK	0.000	3465216	9555	3006
209	OK	0.015	3481600	9554	3297
210	OK	0.000	3473408	9554	2730
211	OK	0.000	3428352	9303	1888
212	OK	0.000	3444736	9984	1877
213	OK	0.015	3616768	37691	12907
214	OK	0.000	3641344	37690	13974
215	OK	0.015	3633152	37690	11820
216	OK	0.000	3461120	39602	7150
217	OK	0.000	3465216	38744	7125
218	OK	0.015	4550656	178903	63876
219	OK	0.015	4546560	178902	68889
220	OK	0.000	4538368	178902	58890
221	OK	0.000	3756032	185712	33049

222	OK	0.015	3768320	180580	33013
223	OK	0.046	15028224	1853240	724890
224	OK	0.046	15044608	1853239	773873
225	OK	0.046	15020032	1853239	675751
226	OK	0.031	7204864	1855624	324156
227	OK	0.046	7213056	1856715	324455
228	OK	0.093	24752128	3473125	1412256
229	OK	0.078	24772608	3473124	1501788
230	OK	0.062	24776704	3473124	1322578
231	OK	0.062	10551296	3603994	592172
232	OK	0.078	10612736	3646224	592525
233	OK	0.125	27299840	3888905	1589032
234	OK	0.078	27254784	3888904	1688889
235	OK	0.140	27254784	3888904	1488890
236	OK	0.062	11276288	3890628	661024
237	OK	0.046	11354112	3986010	661067

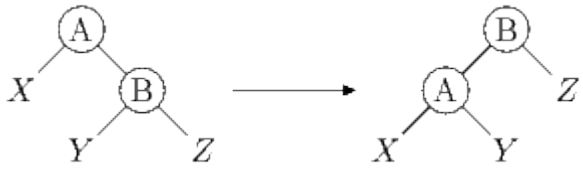
Задача 2. Делаю я левый поворот...

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

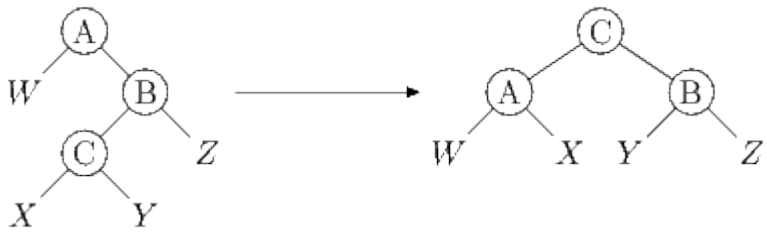
Для балансировки АВЛ-дерева при операциях вставки и удаления производятся *левые* и *правые* повороты. Левый поворот в вершине производится, когда баланс этой вершины больше 1, аналогично, правый поворот производится при балансе, меньшем −1.

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 ($3 \le N \le 2 * 10^5$ — 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 $-10^9 \le K \le 10^9$, номера левого ребенка i-ой вершины (I < Li ≤ N или Li = 0, если левого ребенка нет) и номера правого ребенка i-ой вершины (I < R ≤ N или Ri = 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

Исходный код к задаче 2

```
#include <iostream>
#include <string>
using namespace std;
/**
#define cin std::cin
#define cout std::cout
/*/
#include "edx-io.hpp"
#define cin io
#define cout io
/**/
struct t_node {
     int left, right, key;
} *tree;
int *keys, *h, *b;
int sz;
int cnt(int i) {
     int d = b[i] = 0;
     if (tree[i].left) {
          d = max(cnt(tree[i].left - 1), d);
          b[i] -= h[tree[i].left - 1];
     }
     if (tree[i].right) {
          d = max(cnt(tree[i].right - 1), d);
          b[i] += h[tree[i].right - 1];
     }
     return h[i] = (d + 1);
}
int find(int x) {
     int i = 0;
     while (tree[i].key != x) {
          if (x < tree[i].key) {
               if (tree[i].left) {
                    i = tree[i].left - 1;
               }
               else {
                    return -1;
               }
          }
          else {
               if (tree[i].right) {
                    i = tree[i].right - 1;
               }
               else {
                    return -1;
               }
          }
     }
     return i;
```

```
}
void big_left_rotation(int root_index) {
    t_node
            a = tree[root_index],
            b = tree[a.right - 1],
            c = tree[b.left - 1];
    int x_ind = c.left - 1,
            y_ind = c.right - 1,
            old_c_ind = b.left - 1,
            b_ind = a.right - 1;
    c.left = old_c_ind + 1;
    c.right = b_ind + 1;
    b.left = y_ind + 1;
    a.right = x_ind + 1;
    tree[root_index] = c;
    tree[old_c_ind] = a;
    tree[b_ind] = b;
}
void left_rotation(int root_index) {
    if (b[tree[root_index].right - 1] < 0) {
        big_left_rotation(root_index);
        return;
    }
    t_node
            a = tree[root_index],
            b = tree[a.right - 1];
    int y_ind = b.left - 1,
            old_b_index = a.right - 1;
    b.left = old_b_index + 1;
    a.right = y_ind + 1;
    tree[root_index] = b;
    tree[old_b_index] = a;
}
int *indexes;
int curr_index = 1;
void calc_index(int i) {
    if (!i) { return; }
    t_node n = tree[i - 1];
    indexes[i] = curr_index++;
    calc_index(n.left);
    calc_index(n.right);
}
void print_node(int i) {
    if (!i) { return; }
    t_node n = tree[i - 1];
    cout << n.key << " " << indexes[n.left] << " " << indexes[n.right] << "\n";
    print_node(n.left);
    print_node(n.right);
}
int main() {
    int n;
    cin >> n;
    tree = new t_node[sz = n];
    h = new int[n];
    b = new int[n];
    indexes = new int[n + 1];
    indexes[0] = 0;
    for (int i = 0; i < n; ++i) {
        cin >> tree[i].key >> tree[i].left >> tree[i].right;
        h[i] = 0;
    }
    cnt(0);
    left_rotation(0);
    calc_index(1);
```

```
    cout << n << "\n";
    print_node(1);
    return 0;
}
```

Бенчмарк к задаче 2

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.187	12181504	3986416	3986416
1	OK	0.000	3407872	54	54
2	OK	0.015	3424256	24	24
3	OK	0.015	3420160	24	24
4	OK	0.000	3420160	31	31
5	OK	0.000	3411968	45	45
6	OK	0.015	3411968	45	45
7	OK	0.000	3432448	45	45
8	OK	0.000	3428352	45	45
9	OK	0.000	3448832	52	52
10	OK	0.000	3428352	52	52
11	OK	0.000	3428352	52	52
12	OK	0.000	3416064	52	52
13	OK	0.000	3452928	52	52
14	OK	0.015	3436544	52	52
15	OK	0.000	3436544	59	59
16	OK	0.000	3420160	59	59
17	OK	0.015	3411968	59	59
18	OK	0.000	3428352	59	59
19	OK	0.000	3432448	66	66
20	OK	0.015	3411968	75	75
21	OK	0.000	3428352	75	75
22	OK	0.000	3444736	75	75

23	OK	0.000	3416064	75	75
24	OK	0.046	3416064	75	75
25	OK	0.015	3428352	75	75
26	OK	0.015	3444736	75	75
27	OK	0.000	3436544	75	75
28	OK	0.015	3403776	75	75
29	OK	0.015	3411968	75	75
30	OK	0.000	3411968	75	75
31	OK	0.015	3436544	75	75
32	OK	0.000	3428352	75	75
33	OK	0.000	3411968	75	75
34	OK	0.015	3440640	75	75
35	OK	0.015	3440640	75	75
36	OK	0.000	3428352	75	75
37	OK	0.000	3416064	75	75
38	OK	0.000	3440640	75	75
39	OK	0.000	3448832	75	75
40	OK	0.015	3420160	75	75
41	OK	0.000	3416064	75	75
42	OK	0.015	3444736	75	75
43	OK	0.000	3416064	75	75
44	OK	0.000	3444736	75	75
45	OK	0.000	3424256	75	75
46	OK	0.000	3411968	75	75
47	OK	0.015	3461120	75	75
48	OK	0.015	3411968	75	75
49	OK	0.015	3420160	75	75
50	OK	0.000	3424256	75	75
51	OK	0.015	3444736	75	75

52	OK	0.015	3440640	84	84
53	OK	0.000	3416064	84	84
54	OK	0.015	3424256	84	84
55	OK	0.015	3424256	84	84
56	OK	0.015	3432448	84	84
57	OK	0.000	3416064	84	84
58	OK	0.000	3416064	84	84
59	OK	0.000	3457024	84	84
60	OK	0.000	3411968	84	84
61	OK	0.000	3432448	84	84
62	OK	0.015	3416064	84	84
63	OK	0.015	3448832	84	84
64	OK	0.000	3420160	84	84
65	OK	0.000	3432448	84	84
66	OK	0.000	3428352	84	84
67	OK	0.000	3424256	84	84
68	OK	0.000	3436544	84	84
69	OK	0.015	3420160	84	84
70	OK	0.015	3444736	84	84
71	OK	0.015	3440640	84	84
72	OK	0.000	3420160	84	84
73	OK	0.000	3411968	84	84
74	OK	0.062	3440640	84	84
75	OK	0.000	3432448	84	84
76	OK	0.000	3436544	84	84
77	OK	0.000	3416064	84	84
78	OK	0.000	3428352	84	84
79	OK	0.000	3420160	84	84
80	OK	0.000	3432448	84	84

81	OK	0.015	3440640	84	84
82	OK	0.015	3432448	84	84
83	OK	0.015	3416064	84	84
84	OK	0.000	3428352	84	84
85	OK	0.000	3416064	84	84
86	OK	0.015	3448832	84	84
87	OK	0.000	3440640	84	84
88	OK	0.015	3457024	84	84
89	OK	0.015	3444736	84	84
90	OK	0.015	3440640	84	84
91	OK	0.015	3436544	84	84
92	OK	0.000	3420160	84	84
93	OK	0.015	3432448	84	84
94	OK	0.000	3448832	84	84
95	OK	0.000	3420160	84	84
96	OK	0.015	3420160	84	84
97	OK	0.000	3407872	84	84
98	OK	0.000	3411968	84	84
99	OK	0.015	3407872	84	84
100	OK	0.000	3424256	84	84
101	OK	0.000	3416064	84	84
102	OK	0.015	3440640	84	84
103	OK	0.000	3407872	84	84
104	OK	0.000	3432448	84	84
105	OK	0.000	3428352	84	84
106	OK	0.000	3440640	84	84
107	OK	0.015	3432448	84	84
108	OK	0.000	3420160	84	84
109	OK	0.000	3440640	84	84

110	OK	0.031	3448832	84	84
111	OK	0.000	3428352	84	84
112	OK	0.000	3440640	84	84
113	OK	0.015	3420160	84	84
114	OK	0.015	3444736	84	84
115	OK	0.015	3444736	84	84
116	OK	0.015	3424256	84	84
117	OK	0.015	3420160	84	84
118	OK	0.000	3448832	84	84
119	OK	0.015	3420160	84	84
120	OK	0.000	3424256	84	84
121	OK	0.000	3424256	84	84
122	OK	0.000	3444736	84	84
123	OK	0.000	3416064	84	84
124	OK	0.000	3424256	84	84
125	OK	0.000	3420160	84	84
126	OK	0.015	3436544	84	84
127	OK	0.000	3428352	84	84
128	OK	0.000	3436544	84	84
129	OK	0.015	3440640	84	84
130	OK	0.000	3436544	84	84
131	OK	0.015	3432448	84	84
132	OK	0.000	3416064	93	93
133	OK	0.015	3416064	93	93
134	OK	0.015	3457024	93	93
135	OK	0.000	3424256	93	93
136	OK	0.015	3444736	93	93
137	OK	0.015	3411968	93	93
138	OK	0.000	3411968	93	93

139	OK	0.015	3420160	93	93
140	OK	0.000	3420160	93	93
141	OK	0.015	3448832	93	93
142	OK	0.000	3424256	93	93
143	OK	0.015	3420160	93	93
144	OK	0.015	3407872	93	93
145	OK	0.000	3411968	93	93
146	OK	0.015	3432448	93	93
147	OK	0.000	3424256	93	93
148	OK	0.015	3457024	93	93
149	OK	0.000	3424256	93	93
150	OK	0.015	3428352	93	93
151	OK	0.000	3428352	93	93
152	OK	0.015	3428352	93	93
153	OK	0.000	3420160	93	93
154	OK	0.015	3428352	93	93
155	OK	0.015	3416064	93	93
156	OK	0.000	3424256	93	93
157	OK	0.015	3420160	93	93
158	OK	0.046	3436544	93	93
159	OK	0.000	3411968	93	93
160	OK	0.000	3448832	93	93
161	OK	0.000	3416064	93	93
162	OK	0.015	3444736	93	93
163	OK	0.015	3416064	93	93
164	OK	0.000	3436544	93	93
165	OK	0.015	3444736	93	93
166	OK	0.015	3436544	93	93
167	OK	0.000	3420160	93	93

168	OK	0.000	3411968	93	93
169	OK	0.000	3428352	93	93
170	OK	0.000	3452928	93	93
171	OK	0.015	3432448	93	93
172	OK	0.015	3428352	93	93
173	OK	0.000	3411968	93	93
174	OK	0.000	3416064	93	93
175	OK	0.015	3428352	93	93
176	OK	0.000	3436544	93	93
177	OK	0.015	3444736	93	93
178	OK	0.000	3420160	93	93
179	OK	0.000	3436544	93	93
180	OK	0.000	3432448	93	93
181	OK	0.000	3420160	93	93
182	OK	0.031	3432448	93	93
183	OK	0.015	3436544	93	93
184	OK	0.031	3428352	93	93
185	OK	0.000	3432448	93	93
186	OK	0.015	3416064	93	93
187	OK	0.015	3424256	93	93
188	OK	0.015	3424256	93	93
189	OK	0.000	3440640	93	93
190	OK	0.000	3440640	93	93
191	OK	0.000	3432448	93	93
192	OK	0.000	3436544	93	93
193	OK	0.000	3424256	93	93
194	OK	0.015	3444736	93	93
195	OK	0.015	3444736	93	93
196	OK	0.000	3428352	93	93

197	OK	0.000	3416064	93	93
198	OK	0.000	3411968	93	93
199	OK	0.000	3448832	93	93
200	OK	0.015	3440640	93	93
201	OK	0.031	3440640	93	93
202	OK	0.000	3411968	93	93
203	OK	0.000	3436544	93	93
204	OK	0.015	3424256	93	93
205	OK	0.015	3452928	93	93
206	OK	0.062	3444736	93	93
207	OK	0.015	3428352	93	93
208	OK	0.000	3424256	93	93
209	OK	0.000	3416064	93	93
210	OK	0.000	3428352	93	93
211	OK	0.015	3420160	93	93
212	OK	0.000	3420160	93	93
213	OK	0.000	3452928	93	93
214	OK	0.015	3424256	93	93
215	OK	0.000	3424256	93	93
216	OK	0.000	3424256	93	93
217	OK	0.000	3436544	93	93
218	OK	0.000	3420160	93	93
219	OK	0.015	3457024	93	93
220	OK	0.015	3436544	93	93
221	OK	0.000	3416064	93	93
222	OK	0.015	3432448	93	93
223	OK	0.000	3444736	93	93
224	OK	0.015	3428352	93	93
225	OK	0.000	3440640	93	93

226	OK	0.000	3436544	93	93
227	OK	0.000	3420160	93	93
228	OK	0.015	3424256	93	93
229	OK	0.015	3416064	93	93
230	OK	0.000	3436544	93	93
231	OK	0.015	3448832	93	93
232	OK	0.000	3440640	93	93
233	OK	0.015	3420160	93	93
234	OK	0.000	3416064	93	93
235	OK	0.015	3420160	93	93
236	OK	0.015	3424256	93	93
237	OK	0.015	3444736	93	93
238	OK	0.000	3428352	93	93
239	OK	0.000	3428352	93	93
240	OK	0.000	3416064	93	93
241	OK	0.015	3411968	93	93
242	OK	0.000	3444736	93	93
243	OK	0.000	3432448	93	93
244	OK	0.000	3420160	93	93
245	OK	0.000	3440640	93	93
246	OK	0.000	3420160	93	93
247	OK	0.000	3436544	93	93
248	OK	0.015	3432448	93	93
249	OK	0.000	3432448	93	93
250	OK	0.015	3411968	93	93
251	OK	0.000	3432448	93	93
252	OK	0.000	3424256	220	220
253	OK	0.000	3444736	1798	1798
254	OK	0.000	3436544	1842	1842

255	OK	0.000	3440640	9606	9606
256	OK	0.000	3457024	9569	9569
257	OK	0.000	3457024	9662	9662
258	OK	0.015	3444736	9777	9777
259	OK	0.015	3473408	37717	37717
260	OK	0.000	3502080	37874	37874
261	OK	0.062	3493888	39268	39268
262	OK	0.031	3485696	39470	39470
263	OK	0.015	3796992	181024	181024
264	OK	0.015	3809280	183405	183405
265	OK	0.031	3784704	180784	180784
266	OK	0.015	3801088	183152	183152
267	OK	0.015	3801088	181246	181246
268	OK	0.015	3792896	180886	180886
269	OK	0.093	7569408	1843051	1843051
270	OK	0.093	7643136	1888721	1888721
271	OK	0.093	7614464	1866794	1866794
272	OK	0.078	7643136	1898864	1898864
273	OK	0.093	7634944	1908693	1908693
274	OK	0.093	7618560	1881384	1881384
275	OK	0.156	11218944	3526463	3526463
276	OK	0.156	11247616	3559824	3559824
277	OK	0.156	11321344	3598289	3598289
278	OK	0.156	11264000	3590402	3590402
279	OK	0.156	11255808	3567894	3567894
280	OK	0.187	11255808	3566660	3566660
281	OK	0.156	11202560	3487414	3487414
282	OK	0.187	12062720	3874088	3874088
283	OK	0.171	12181504	3978208	3978208

284	OK	0.171	12144640	3957801	3957801
285	OK	0.171	12177408	3978349	3978349
286	OK	0.171	12165120	3986416	3986416
287	OK	0.171	12103680	3933437	3933437
288	OK	0.171	12103680	3926735	3926735

Задача 3 Вставка в АВЛ-дерево

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

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. В первой строке файла находится число $N\left(0 \le N \le 2 * 10^5\right)$ — 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 $-10^9 \le Ki \le 10^9$, номера левого ребенка i-ой вершины (i < Li < N или

Li = 0 , если левого ребенка нет) и номера правого ребенка i-ой вершины (I < Ri < N или Ri = 0, если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является корректным АВЛ-деревом.

В последней строке содержится число $X\left(X \le 10^9\right)$ — 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

Исходный код к задаче 3

```
#include <fstream>
#include <vector>
using namespace std;
class AVL_tree {
public:
    struct Node {
        int value;
        Node *left_child;
        Node *right_child;
        int height;
        int number;
    };
    AVL_tree(ifstream &input, int n) {
        int K, L, R;
        vector<Node *> parents(n + 1);
        for (int i = 1; i <= n; i++) {
            input >> K >> L >> R;
            Node *new_node = new Node{K, nullptr, nullptr, 0, 0};
            parents[L] = new_node;
            parents[R] = new_node;
            if (i == 1) {
                tree = new_node;
            } else {
                if (K < parents[i]->value) {
                    parents[i]->left_child = new_node;
                } else {
                    parents[i]->right_child = new_node;
                }
            }
        }
        height(tree);
    }
    AVL_tree() {
        tree = nullptr;
    }
```

```
void rotate_right(Node *node) {
    if (count_balance(node->left_child) == 1) {
        Node *A = node;
        Node *B = node->left_child;
        Node *C = B->right_child;
        Node *X = C->left_child;
        Node *Y = C->right_child;
        B->right_child = X;
        A->left_child = Y;
        swap(*A, *C);
        A->left_child = B;
        A->right_child = C;
        fix_height(C);
        fix_height(B);
        fix_height(A);
    } else {
        Node *B = node->left_child;
        Node *Y = B->right_child;
        Node *A = node;
        A->left_child = Y;
        swap(*A, *B);
        A->right_child = B;
        fix_height(B);
        fix_height(A);
    }
}
void rotate_left(Node *node) {
    if (count_balance(node->right_child) == -1) {
        Node *A = node;
        Node *B = node->right_child;
        Node *C = B->left_child;
        Node *X = C->left_child;
        Node *Y = C->right_child;
        A->right_child = X;
        B->left_child = Y;
        Node temp = *C;
        *C = *A;
        *A = temp;
        A->left_child = C;
        A->right_child = B;
        fix_height(C);
        fix_height(B);
        fix_height(A);
    } else {
        Node *B = node->right_child;
        Node *Y = B->left_child;
        Node *A = node;
        A->right_child = Y;
        Node temp = *B;
        *B = *A;
        *A = temp;
        A->left_child = B;
        fix_height(B);
        fix_height(A);
    }
}
void print_tree(ofstream &output) {
    int counter = 1;
    numerate(tree, &counter);
    output << counter - 1 << '\n';
    print(output, tree);
}
void insert(int value) {
```

```
            tree = append(tree, value);
        }
        void remove(int value) {
            tree = remove(tree, value);
        }
        int root_balance() {
            if (tree != nullptr) {
                return count_balance(tree);
            } else {
                return 0;
            }
        }
        bool contains(int x) {
            Node* temp = find(tree, x);
            if (temp != nullptr) {
                return temp->value == x;
            } else {
                return false;
            }
        }
    private:
        Node *find(Node *node, int x) {
            if (node == nullptr)
                return node;
            if (node->value == x) {
                return node;
            }
            if (node->value < x) {
                if (node->right_child == nullptr) {
                    return node;
                } else {
                    return find(node->right_child, x);
                }
            } else {
                if (node->left_child == nullptr) {
                    return node;
                } else {
                    return find(node->left_child, x);
                }
            }
        }
        Node* max_right(Node* node) {
            while (node->right_child != nullptr) {
                node = node->right_child;
            }
            return node;
        }
        Node* remove(Node* node, int value) {
            if (node == nullptr || node->value == value) {
                if (node == nullptr)
                    return node;
                if (node->right_child == nullptr && node->left_child == nullptr) {
                    return nullptr;
                }
                if (node->left_child == nullptr) {
                    return node->right_child;
                }
                Node* m_right = max_right(node->left_child);
                node->value = m_right->value;
                node->left_child = remove(node->left_child, m_right->value);
            }
            if (node->value < value) {
                node->right_child = remove(node->right_child, value);
```

```
     } else {
          node->left_child = remove(node->left_child, value);
     }
     return balance(node);
}
Node* append(Node* node, int value) {
     if (node == nullptr || node->value == value) {
          if (node != nullptr) {
               return node;
          } else {
               return new Node{value, nullptr, nullptr, 1, 0};
          }
     }
     if (node->value < value) {
          node->right_child = append(node->right_child, value);
     } else {
          node->left_child = append(node->left_child, value);
     }
     return balance(node);
}
Node *balance(Node *node) {
     fix_height(node);
     if (count_balance(node) == 2) {
          rotate_left(node);
          return node;
     }
     if (count_balance(node) == -2) {
          rotate_right(node);
          return node;
     }
     return node;
}
int count_balance(Node *node) {
     if (node == nullptr)
          return 0;
     int left_h = 0, right_h = 0;
     if (node->right_child != nullptr) {
          right_h = node->right_child->height;
     }
     if (node->left_child != nullptr) {
          left_h = node->left_child->height;
     }
     return right_h - left_h;
}
void fix_height(Node *node) {
     int h = 0;
     if (node->left_child != nullptr) {
          h = node->left_child->height;
     }
     if (node->right_child != nullptr) {
          h = max(h, node->right_child->height);
     }
     node->height = ++h;
}
int height(Node *node) {
     if (node == nullptr)
          return 0;
     int h = 0;
     if (node->left_child != nullptr) {
          h = max(h, height(node->left_child));
     }
     if (node->right_child != nullptr) {
          h = max(h, height(node->right_child));
```

```
        }
        node->height = ++h;
        return h;
    }
    void numerate(Node *node, int *current_number) {
        if (node == nullptr)
            return;
        node->number = (*current_number)++;
        if (node->left_child != nullptr) {
            numerate(node->left_child, current_number);
        }
        if (node->right_child != nullptr) {
            numerate(node->right_child, current_number);
        }
    }
    void print(ofstream &output, Node *node) {
        if (node == nullptr)
            return;
        output << node->value << ' ';
        if (node->left_child != nullptr) {
            output << node->left_child->number << ' ';
        } else {
            output << 0 << ' ';
        }
        if (node->right_child != nullptr) {
            output << node->right_child->number << '\n';
        } else {
            output << 0 << '\n';
        }
        if (node->left_child != nullptr) {
            print(output, node->left_child);
        }
        if (node->right_child != nullptr) {
            print(output, node->right_child);
        }
    }
    Node *tree = nullptr;
};
int main() {
    ifstream fi("input.txt");
    ofstream fo("output.txt");
    int n, x;
    char command;
    fi >> n;
    AVL_tree tree(fi, n);
    fi >> x;
    tree.insert(x);
    tree.print_tree(fo);
}
```

Бенчмарк к задаче 3

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.609	74362880	4011957	3811967
1	OK	0.046	12697600	20	23
2	OK	0.046	11325440	6	11

3	OK	0.046	12603392	14	18
4	OK	0.031	12595200	13	17
5	OK	0.031	12681216	21	24
6	OK	0.046	12611584	20	23
7	OK	0.046	12660736	20	23
8	OK	0.046	12689408	21	24
9	OK	0.031	12623872	20	23
10	OK	0.046	12644352	20	23
11	OK	0.046	12603392	28	30
12	OK	0.046	12443648	27	29
13	OK	0.031	12439552	27	29
14	OK	0.046	12398592	27	29
15	OK	0.046	12423168	35	36
16	OK	0.062	12636160	34	35
17	OK	0.046	12484608	34	35
18	OK	0.046	12394496	34	35
19	OK	0.046	12484608	34	35
20	OK	0.031	12427264	35	36
21	OK	0.031	12365824	34	35
22	OK	0.046	12480512	34	35
23	OK	0.046	12414976	34	35
24	OK	0.046	12402688	34	35
25	OK	0.031	12423168	35	36
26	OK	0.046	12427264	34	35
27	OK	0.031	12390400	34	35
28	OK	0.046	12562432	34	35
29	OK	0.031	12390400	34	35
30	OK	0.046	12529664	35	36
31	OK	0.031	12451840	34	35

32	OK	0.046	12472320	34	35
33	OK	0.046	12402688	34	35
34	OK	0.031	12390400	34	35
35	OK	0.078	12398592	42	42
36	OK	0.062	12419072	41	41
37	OK	0.046	12513280	41	41
38	OK	0.046	12414976	41	41
39	OK	0.031	12394496	41	41
40	OK	0.046	12439552	41	41
41	OK	0.031	12525568	42	42
42	OK	0.031	12476416	41	41
43	OK	0.031	12398592	41	41
44	OK	0.046	12394496	41	41
45	OK	0.031	12386304	41	41
46	OK	0.046	12402688	41	41
47	OK	0.046	12410880	42	42
48	OK	0.046	12505088	41	41
49	OK	0.046	12427264	41	41
50	OK	0.078	12431360	41	41
51	OK	0.031	12414976	41	41
52	OK	0.031	12525568	41	41
53	OK	0.046	12492800	42	42
54	OK	0.046	12378112	41	41
55	OK	0.046	12423168	41	41
56	OK	0.062	12410880	41	41
57	OK	0.031	12414976	41	41
58	OK	0.046	12464128	41	41
59	OK	0.046	12455936	42	42
60	OK	0.078	12423168	41	41

61	OK	0.062	12480512	41	41
62	OK	0.046	12386304	41	41
63	OK	0.031	12402688	41	41
64	OK	0.046	12488704	41	41
65	OK	0.046	12517376	42	42
66	OK	0.062	12451840	41	41
67	OK	0.031	12419072	41	41
68	OK	0.031	12427264	41	41
69	OK	0.031	12410880	41	41
70	OK	0.031	12419072	41	41
71	OK	0.046	12414976	50	49
72	OK	0.046	12476416	49	48
73	OK	0.046	12427264	49	48
74	OK	0.046	12410880	49	48
75	OK	0.031	12480512	49	48
76	OK	0.046	12566528	49	48
77	OK	0.046	12447744	50	49
78	OK	0.031	12382208	50	49
79	OK	0.031	12406784	49	48
80	OK	0.046	12406784	49	48
81	OK	0.031	12525568	49	48
82	OK	0.031	12480512	49	48
83	OK	0.046	12406784	49	48
84	OK	0.031	12439552	50	49
85	OK	0.031	12464128	50	49
86	OK	0.046	12406784	49	48
87	OK	0.046	12414976	49	48
88	OK	0.046	12451840	49	48
89	OK	0.031	12455936	49	48

90	OK	0.046	12386304	49	48
91	OK	0.046	12382208	50	49
92	OK	0.031	12484608	50	49
93	OK	0.046	12460032	49	48
94	OK	0.046	12541952	49	48
95	OK	0.031	12427264	49	48
96	OK	0.062	12447744	49	48
97	OK	0.046	12427264	49	48
98	OK	0.031	12460032	50	49
99	OK	0.046	12390400	58	56
100	OK	0.046	12427264	57	55
101	OK	0.031	12496896	57	55
102	OK	0.031	12427264	57	55
103	OK	0.031	12414976	57	55
104	OK	0.046	12468224	57	55
105	OK	0.046	12423168	58	56
106	OK	0.031	12455936	58	56
107	OK	0.046	12484608	58	56
108	OK	0.046	12546048	57	55
109	OK	0.046	12427264	57	55
110	OK	0.031	12410880	57	55
111	OK	0.031	12451840	57	55
112	OK	0.031	12390400	57	55
113	OK	0.046	12484608	58	56
114	OK	0.046	12431360	58	56
115	OK	0.046	12398592	58	56
116	OK	0.046	12398592	57	55
117	OK	0.031	12394496	57	55
118	OK	0.046	12398592	57	55

119	OK	0.046	12480512	57	55
120	OK	0.031	12484608	57	55
121	OK	0.046	12431360	58	56
122	OK	0.031	12382208	58	56
123	OK	0.046	12476416	58	56
124	OK	0.046	12472320	57	55
125	OK	0.031	12414976	57	55
126	OK	0.031	12410880	57	55
127	OK	0.046	12398592	57	55
128	OK	0.062	12414976	57	55
129	OK	0.046	12439552	58	56
130	OK	0.031	12476416	58	56
131	OK	0.046	12414976	58	56
132	OK	0.046	12509184	57	55
133	OK	0.046	12460032	57	55
134	OK	0.046	12464128	57	55
135	OK	0.031	12472320	57	55
136	OK	0.031	12509184	57	55
137	OK	0.046	12423168	58	56
138	OK	0.046	12394496	58	56
139	OK	0.031	12394496	58	56
140	OK	0.046	12414976	57	55
141	OK	0.046	12386304	57	55
142	OK	0.031	12472320	57	55
143	OK	0.046	12410880	57	55
144	OK	0.046	12492800	57	55
145	OK	0.046	12423168	58	56
146	OK	0.046	12423168	58	56
147	OK	0.031	12529664	58	56

148	OK	0.046	12468224	57	55
149	OK	0.046	12414976	57	55
150	OK	0.046	12431360	57	55
151	OK	0.031	12394496	57	55
152	OK	0.031	12398592	57	55
153	OK	0.046	12394496	58	56
154	OK	0.046	12513280	58	56
155	OK	0.062	12427264	58	56
156	OK	0.031	12398592	57	55
157	OK	0.031	12451840	57	55
158	OK	0.046	12398592	57	55
159	OK	0.046	12529664	57	55
160	OK	0.031	12492800	57	55
161	OK	0.046	12419072	58	56
162	OK	0.046	12435456	58	56
163	OK	0.031	12394496	58	56
164	OK	0.031	12435456	57	55
165	OK	0.031	12443648	57	55
166	OK	0.031	12558336	57	55
167	OK	0.031	12402688	57	55
168	OK	0.031	12431360	57	55
169	OK	0.031	12505088	58	56
170	OK	0.031	12439552	58	56
171	OK	0.031	12480512	58	56
172	OK	0.031	12480512	57	55
173	OK	0.046	12402688	57	55
174	OK	0.031	12402688	57	55
175	OK	0.046	12451840	57	55
176	OK	0.031	12402688	57	55

177	OK	0.046	12406784	58	56
178	OK	0.046	12484608	58	56
179	OK	0.031	12398592	58	56
180	OK	0.046	12427264	57	55
181	OK	0.046	12394496	57	55
182	OK	0.031	12509184	57	55
183	OK	0.046	12554240	57	55
184	OK	0.031	12464128	57	55
185	OK	0.031	12410880	58	56
186	OK	0.031	12439552	58	56
187	OK	0.046	12406784	58	56
188	OK	0.031	12406784	57	55
189	OK	0.046	12476416	57	55
190	OK	0.046	12419072	57	55
191	OK	0.046	12414976	57	55
192	OK	0.031	12431360	57	55
193	OK	0.046	12390400	58	56
194	OK	0.031	12513280	58	56
195	OK	0.031	12496896	58	56
196	OK	0.046	12472320	57	55
197	OK	0.031	12402688	57	55
198	OK	0.046	12431360	57	55
199	OK	0.031	12419072	57	55
200	OK	0.046	12398592	57	55
201	OK	0.031	12550144	58	56
202	OK	0.031	12435456	58	56
203	OK	0.046	12414976	58	56
204	OK	0.062	12406784	57	55
205	OK	0.031	12386304	57	55

206	OK	0.031	12468224	57	55
207	OK	0.031	12517376	57	55
208	OK	0.046	12447744	57	55
209	OK	0.046	12394496	58	56
210	OK	0.046	12427264	58	56
211	OK	0.046	12423168	58	56
212	OK	0.046	12447744	57	55
213	OK	0.031	12521472	57	55
214	OK	0.046	12398592	57	55
215	OK	0.031	12398592	57	55
216	OK	0.046	12398592	57	55
217	OK	0.031	12394496	58	56
218	OK	0.046	12472320	58	56
219	OK	0.093	12476416	58	56
220	OK	0.031	12480512	57	55
221	OK	0.046	12431360	57	55
222	OK	0.046	12414976	57	55
223	OK	0.046	12394496	57	55
224	OK	0.031	12435456	57	55
225	OK	0.031	12480512	58	56
226	OK	0.031	12423168	58	56
227	OK	0.031	12419072	58	56
228	OK	0.046	12419072	57	55
229	OK	0.046	12468224	57	55
230	OK	0.046	12492800	57	55
231	OK	0.031	12509184	57	55
232	OK	0.031	12488704	57	55
233	OK	0.031	12460032	58	56
234	OK	0.031	12410880	58	56

235	OK	0.046	12402688	240	232
236	OK	0.046	12554240	243	235
237	OK	0.031	12492800	1835	1731
238	OK	0.046	12480512	1815	1711
239	OK	0.031	12517376	1834	1730
240	OK	0.031	12845056	9951	9390
241	OK	0.046	13008896	9831	9270
242	OK	0.046	12832768	9854	9293
243	OK	0.046	13651968	38301	36154
244	OK	0.046	13611008	37664	35517
245	OK	0.046	13619200	39108	36961
246	OK	0.078	13582336	39190	37043
247	OK	0.062	19845120	183695	173704
248	OK	0.046	19910656	184258	174267
249	OK	0.046	19853312	185065	175074
250	OK	0.062	19881984	185428	175437
251	OK	0.062	19861504	185741	175750
252	OK	0.328	44990464	1900094	1801980
253	OK	0.328	44924928	1860225	1762111
254	OK	0.312	44691456	1899455	1801341
255	OK	0.328	45142016	1861088	1762974
256	OK	0.328	44969984	1942127	1844013
257	OK	0.312	45355008	1930200	1832086
258	OK	0.328	44560384	1861244	1763130
259	OK	0.515	67756032	3510448	3331247
260	OK	0.515	68018176	3650901	3471700
261	OK	0.531	67731456	3552374	3373173
262	OK	0.500	66859008	3435983	3256782
263	OK	0.500	67948544	3562689	3383488

264	OK	0.546	67629056	3521159	3341958
265	OK	0.531	67739648	3539149	3359948
266	OK	0.578	74362880	3985264	3785274
267	OK	0.578	73617408	3866892	3666902
268	OK	0.593	74133504	3942753	3742763
269	OK	0.562	72343552	3824263	3624273
270	OK	0.578	73494528	4011957	3811967
271	OK	0.578	73318400	3955420	3755430
272	OK	0.609	73367552	3946583	3746593
273	OK	0.578	73195520	3891536	3691546

Задача 4 Удаление из АВЛ-дерева

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

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 K 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. В первой строке файла находится число $N\left(0 \le N \le 2 * 10^5\right)$ — 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 $-10^9 \le Ki \le 10^9$, номера левого ребенка i-ой вершины (i < Li < N или Li = 0 , если левого ребенка нет) и номера правого ребенка i-ой вершины (I < Ri < N или Ri = 0, если правого ребенка нет).

Все ключи различны. Гарантируется, что данное дерево является корректным АВЛ-деревом.

В последней строке содержится число $X\left(X \le 10^9\right)$ — 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 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

Исходный код к задаче 4

```
int main() {
    ifstream fi("input.txt");
    ofstream fo("output.txt");
    int n, x;
    char command;
    fi >> n;
    AVL_tree tree(fi, n);
    fi >> x;
    tree.remove(x);
    tree.print_tree(fo);
}
```

Бенчмарк к задаче 4

Задача 5 Упорядоченное множество на АВЛ-дереве

Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

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 $N\left(1\le N\le 2*10^5\right)$ — 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 в множество. Если число x там уже содержится, множество изменять не следует.
- `D x` — удалить число из множества. Если числа x нет в множестве, множество изменять не следует.
- `C x` — проверить, есть ли число 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	0 1 0 Y

C 4 C 6 D 5	N -1

Исходный код к задаче 5

```
int main() {
    ifstream fi("input.txt");
    ofstream fo("output.txt");
    int n, x;
    char command;
    fi >> n;
    AVL_tree tree;
    for (int i = 0; i < n; i++) {
        fi >> command >> x;
        if (command == 'A') {
            tree.insert(x);
            fo << tree.root_balance() << '\n';
        } else if (command == 'D') {
            tree.remove(x);
            fo << tree.root_balance() << '\n';
        } else {
            if (tree.contains(x)) {
                fo << 'Y' << '\n';
            } else {
                fo << 'N' << '\n';
            }
        }
    }
}
```

Бенчмарк к задаче 5

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.593	73105408	4077288	3877258
1	OK	0.031	11280384	27	17
2	OK	0.031	10674176	13	3
3	OK	0.031	11259904	20	11
4	OK	0.031	11169792	20	11
5	OK	0.031	11202560	20	11
6	OK	0.031	11214848	20	11
7	OK	0.031	11218944	27	17
8	OK	0.031	11153408	27	17
9	OK	0.031	11161600	27	17
10	OK	0.031	11169792	34	23

11	OK	0.031	11235328	34	23
12	OK	0.046	11366400	34	23
13	OK	0.031	11182080	34	23
14	OK	0.031	11137024	34	23
15	OK	0.031	11177984	34	23
16	OK	0.031	11214848	34	23
17	OK	0.031	11145216	34	23
18	OK	0.046	11190272	34	23
19	OK	0.031	11141120	34	23
20	OK	0.078	11153408	34	23
21	OK	0.031	11141120	34	23
22	OK	0.031	11145216	34	23
23	OK	0.031	11223040	34	23
24	OK	0.031	11235328	34	23
25	OK	0.031	11161600	34	23
26	OK	0.046	11177984	41	29
27	OK	0.031	11124736	41	29
28	OK	0.031	11251712	41	29
29	OK	0.078	11202560	41	29
30	OK	0.031	11157504	41	29
31	OK	0.031	11137024	41	29
32	OK	0.031	11141120	41	29
33	OK	0.031	11169792	41	29
34	OK	0.031	11141120	41	29
35	OK	0.031	11227136	41	29
36	OK	0.031	11149312	41	29
37	OK	0.031	11235328	41	29
38	OK	0.046	11165696	41	29
39	OK	0.046	11198464	41	29

40	OK	0.046	11177984	41	29
41	OK	0.046	11177984	41	29
42	OK	0.031	11137024	41	29
43	OK	0.031	11153408	41	29
44	OK	0.031	11177984	41	29
45	OK	0.031	11132928	41	29
46	OK	0.031	11141120	41	29
47	OK	0.031	11272192	41	29
48	OK	0.031	11132928	41	29
49	OK	0.031	11206656	41	29
50	OK	0.031	11210752	41	29
51	OK	0.031	11177984	41	29
52	OK	0.062	11157504	41	29
53	OK	0.046	11206656	41	29
54	OK	0.046	11137024	41	29
55	OK	0.031	11116544	41	29
56	OK	0.031	11165696	48	35
57	OK	0.031	11169792	48	35
58	OK	0.031	11202560	48	35
59	OK	0.046	11194368	48	35
60	OK	0.031	11165696	48	35
61	OK	0.031	11210752	48	35
62	OK	0.046	11247616	48	35
63	OK	0.046	11223040	48	35
64	OK	0.031	11141120	48	35
65	OK	0.046	11137024	48	35
66	OK	0.062	11145216	48	35
67	OK	0.031	11149312	48	35
68	OK	0.031	11153408	48	35

69	OK	0.031	11218944	48	35
70	OK	0.031	11227136	48	35
71	OK	0.031	11227136	48	35
72	OK	0.046	11161600	48	35
73	OK	0.031	11210752	48	35
74	OK	0.046	11206656	48	35
75	OK	0.031	11186176	48	35
76	OK	0.031	11141120	48	35
77	OK	0.031	11165696	48	35
78	OK	0.031	11165696	48	35
79	OK	0.046	11124736	48	35
80	OK	0.031	11149312	55	41
81	OK	0.046	11157504	55	41
82	OK	0.031	11194368	55	41
83	OK	0.031	11198464	55	41
84	OK	0.046	11153408	55	41
85	OK	0.031	11194368	55	41
86	OK	0.031	11186176	55	41
87	OK	0.031	11214848	55	41
88	OK	0.031	11145216	55	41
89	OK	0.062	11149312	55	41
90	OK	0.031	11157504	55	41
91	OK	0.031	11173888	55	41
92	OK	0.031	11186176	55	41
93	OK	0.046	11210752	55	41
94	OK	0.031	11153408	55	41
95	OK	0.031	11210752	55	41
96	OK	0.046	11202560	55	41
97	OK	0.031	11223040	55	41

98	OK	0.031	11173888	55	41
99	OK	0.031	11161600	55	41
100	OK	0.031	11165696	55	41
101	OK	0.031	11157504	55	41
102	OK	0.046	11194368	55	41
103	OK	0.031	11173888	55	41
104	OK	0.031	11153408	55	41
105	OK	0.031	11169792	55	41
106	OK	0.031	11157504	55	41
107	OK	0.031	11210752	55	41
108	OK	0.031	11190272	55	41
109	OK	0.031	11182080	55	41
110	OK	0.046	11157504	55	41
111	OK	0.031	11145216	55	41
112	OK	0.031	11153408	55	41
113	OK	0.031	11149312	55	41
114	OK	0.031	11145216	55	41
115	OK	0.031	11255808	55	41
116	OK	0.031	11198464	55	41
117	OK	0.031	11161600	55	41
118	OK	0.031	11157504	55	41
119	OK	0.031	11255808	55	41
120	OK	0.031	11206656	55	41
121	OK	0.031	11132928	55	41
122	OK	0.031	11227136	55	41
123	OK	0.031	11235328	55	41
124	OK	0.046	11190272	55	41
125	OK	0.031	11161600	55	41
126	OK	0.031	11186176	55	41

127	OK	0.046	11227136	55	41
128	OK	0.015	11157504	55	41
129	OK	0.046	11145216	55	41
130	OK	0.031	11149312	55	41
131	OK	0.031	11223040	55	41
132	OK	0.031	11190272	55	41
133	OK	0.031	11137024	55	41
134	OK	0.046	11186176	55	41
135	OK	0.046	11194368	55	41
136	OK	0.031	11149312	55	41
137	OK	0.031	11165696	55	41
138	OK	0.031	11165696	55	41
139	OK	0.046	11198464	55	41
140	OK	0.031	11202560	55	41
141	OK	0.031	11186176	55	41
142	OK	0.031	11137024	55	41
143	OK	0.046	11206656	55	41
144	OK	0.031	11128832	55	41
145	OK	0.031	11218944	55	41
146	OK	0.031	11186176	55	41
147	OK	0.062	11239424	55	41
148	OK	0.031	11173888	55	41
149	OK	0.031	11161600	55	41
150	OK	0.031	11169792	55	41
151	OK	0.031	11247616	55	41
152	OK	0.031	11173888	55	41
153	OK	0.031	11177984	55	41
154	OK	0.031	11132928	55	41
155	OK	0.031	11157504	55	41

156	OK	0.031	11210752	55	41
157	OK	0.031	11173888	55	41
158	OK	0.031	11165696	55	41
159	OK	0.031	11247616	55	41
160	OK	0.046	11153408	55	41
161	OK	0.031	11165696	55	41
162	OK	0.046	11145216	55	41
163	OK	0.046	11227136	55	41
164	OK	0.031	11149312	55	41
165	OK	0.031	11202560	55	41
166	OK	0.031	11145216	55	41
167	OK	0.031	11149312	55	41
168	OK	0.031	11186176	55	41
169	OK	0.031	11141120	55	41
170	OK	0.031	11132928	55	41
171	OK	0.031	11235328	55	41
172	OK	0.031	11165696	55	41
173	OK	0.031	11141120	55	41
174	OK	0.031	11161600	55	41
175	OK	0.046	11169792	55	41
176	OK	0.031	11190272	55	41
177	OK	0.046	11169792	55	41
178	OK	0.031	11157504	55	41
179	OK	0.046	11141120	55	41
180	OK	0.031	11218944	55	41
181	OK	0.031	11153408	55	41
182	OK	0.031	11173888	55	41
183	OK	0.031	11231232	55	41
184	OK	0.062	11137024	55	41

185	OK	0.031	11190272	55	41
186	OK	0.031	11206656	55	41
187	OK	0.031	11194368	55	41
188	OK	0.046	11227136	55	41
189	OK	0.031	11235328	55	41
190	OK	0.031	11145216	55	41
191	OK	0.031	11153408	55	41
192	OK	0.031	11231232	55	41
193	OK	0.046	11194368	55	41
194	OK	0.031	11145216	55	41
195	OK	0.031	11157504	55	41
196	OK	0.031	11210752	55	41
197	OK	0.031	11145216	55	41
198	OK	0.031	11190272	55	41
199	OK	0.031	11165696	239	199
200	OK	0.031	11210752	235	197
201	OK	0.031	11259904	1797	1661
202	OK	0.046	11210752	1809	1673
203	OK	0.062	11239424	1831	1695
204	OK	0.031	11649024	9625	9032
205	OK	0.046	11603968	10026	9431
206	OK	0.031	11554816	9672	9077
207	OK	0.046	12337152	39459	37276
208	OK	0.031	12378112	39672	37489
209	OK	0.046	12312576	38780	36597
210	OK	0.031	12312576	38392	36209
211	OK	0.046	18575360	179425	169397
212	OK	0.062	18714624	182878	172848
213	OK	0.062	18661376	185986	175956

214	OK	0.062	18616320	186275	176247
215	OK	0.062	18558976	179082	169054
216	OK	0.312	43905024	1912349	1814199
217	OK	0.296	43671552	1864033	1765883
218	OK	0.328	43782144	1913940	1815788
219	OK	0.312	43384832	1879988	1781838
220	OK	0.312	43466752	1887512	1789362
221	OK	0.343	43692032	1932558	1834406
222	OK	0.296	43356160	1875036	1776886
223	OK	0.515	67313664	3597394	3418153
224	OK	0.546	66379776	3569973	3390732
225	OK	0.500	66260992	3512224	3332985
226	OK	0.546	66990080	3624329	3445088
227	OK	0.515	66396160	3523352	3344113
228	OK	0.531	67207168	3638077	3458836
229	OK	0.500	66342912	3512844	3333605
230	OK	0.593	73105408	4001320	3801290
231	OK	0.578	71942144	3954282	3754252
232	OK	0.578	71716864	3907814	3707786
233	OK	0.578	72245248	3983014	3782986
234	OK	0.578	72617984	4029895	3829865
235	OK	0.593	72736768	4046188	3846160
236	OK	0.578	72945664	4077288	3877258
237	OK	0.562	71864320	3902092	3702064