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)

Студенка Жетесова Дана группы P3217

Преподаватель Муромцев Дмитрий Ильич

Санкт-Петербург

2019 г.

Содержание

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 $N (1 \le N \le 2 \cdot 10^5)$ — 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

Исходный код к задаче 1

```
#include <fstream>
#include <stack>

using namespace std;

int maximum(int a, int b) {
  if (a > b)
    return a;
  else
    return b;
```

```
}

class binary_tree {
public:
   struct Node {
      int value;
      int left_child;
      int right_child;
      int balance = -1;
      int height = 0;
      int number;
   };

   binary_tree(ifstream &input, int new_size) : size(new_size) {
      tree = new Node[size + 1];
      int K, L, R;
      for (int i = 1; i <= size; i++) {
         input >> K >> L >> R;
         tree[i] = {K, L, R, 0};
      }
      tree[0] = {0, 0, 0, 0, 0, 0};
      height(&tree[1]);
      balance(&tree[1]);
   }

   void print_balance(ofstream &output) {
      for (int i = 1; i <= size; i++) {
         output << tree[i].balance << '\n';
      }
   }

   void print_tree(ofstream &output) {
      int number = 1;
      numerate(&tree[1], &number);
      output << size << '\n';
      print(output, &tree[1]);
   }

   void rotateRight(Node *node) {
      if (tree[node->left_child].balance == 1) {
         int B = node->left_child;
         int C = tree[B].right_child;
         int X = tree[C].right_child;
         int Y = tree[C].left_child;
         node->left_child = X;
         tree[B].left_child = Y;
         auto temp = *node;
         *node = tree[C];
         tree[C] = temp;
         node->right_child = C;
         node->left_child = B;
      } else {
         int B = node->left_child;
         int Y = tree[node->left_child].right_child;
```

```
            tree[B].right_child = B;
            Node temp = tree[B];
            node->left_child = Y;
            tree[B] = *node;
            *node = temp;
        }
    }

    void rotateLeft(Node *node) {
        if (tree[node->right_child].balance == -1) {
            int B = node->right_child;
            int C = tree[B].left_child;
            int X = tree[C].left_child;
            int Y = tree[C].right_child;
            node->right_child = X;
            tree[B].left_child = Y;
            auto temp = *node;
            *node = tree[C];
            tree[C] = temp;
            node->left_child = C;
            node->right_child = B;
        } else {
            int B = node->right_child;
            int Y = tree[node->right_child].left_child;
            tree[B].left_child = B;
            Node temp = tree[B];
            node->right_child = Y;
            tree[B] = *node;
            *node = temp;
        }
    }

private:
    Node *tree;
    int size;

    int height(Node *current_node) {
        if (current_node->right_child == 0 && current_node->left_child == 0) {
            current_node->height = 1;
            return 1;
        }
        int left_h = 0, right_h = 0;
        if (current_node->left_child != 0)
            left_h = height(&tree[current_node->left_child]);
        if (current_node->right_child != 0)
            right_h = height(&tree[current_node->right_child]);
        current_node->height = maximum(left_h, right_h) + 1;
        return current_node->height;
    }

    Node *find(Node *node, int x) {
        if (node->value == x || (node->right_child == 0 && node->left_child == 0))
            return node;
        if (node->value < x) {
```

```
        if (node->right_child == 0)
            return node;
        else
            return find(&tree[node->right_child], x);
    } else {
        if (node->left_child == 0)
            return node;
        else
            return find(&tree[node->left_child], x);
    }
}

void balance(Node *node) {
    int left_h, right_h;
    left_h = tree[node->left_child].height;
    right_h = tree[node->right_child].height;
    node->balance = right_h - left_h;
    if (left_h == 0 && right_h == 0)
        return;
    if (left_h != 0)
        balance(&tree[node->left_child]);
    if (right_h != 0)
        balance(&tree[node->right_child]);
}

void numerate(Node *current_node, int *current_number) {
    current_node->number = (*current_number)++;
    if (current_node->left_child != 0)
        numerate(&tree[current_node->left_child], current_number);
    if (current_node->right_child != 0)
        numerate(&tree[current_node->right_child], current_number);
}

void print(ofstream &output, Node *node) {
    output << node->value << ' ';
    if (node->left_child != 0)
        output << tree[node->left_child].number << ' ';
    else
        output << 0 << ' ';
    if (node->right_child != 0)
        output << tree[node->right_child].number << '\n';
    else
        output << 0 << '\n';
    if (node->left_child != 0)
        print(output, &tree[node->left_child]);
    if (node->right_child != 0)
        print(output, &tree[node->right_child]);
}
};

int main() {
    ifstream input ("input.txt");
    ofstream output("output.txt");
```

```
    int n;
    input >> n;
    binary_tree tree(input, n);
    tree.print_balance(output);
}
```

Бенчмарк к задаче 1

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.671	16777216	3986010	1688889
1	OK	0.015	2363392	46	19
2	OK	0.000	2363392	10	3
3	OK	0.015	2363392	17	6
4	OK	0.000	2367488	17	7
5	OK	0.000	2371584	24	9
6	OK	0.000	2383872	24	10
7	OK	0.000	2379776	24	9
8	OK	0.000	2367488	24	10
9	OK	0.000	2363392	24	11
10	OK	0.000	2367488	31	12
11	OK	0.000	2367488	31	13
12	OK	0.000	2383872	31	12
13	OK	0.015	2383872	31	13
14	OK	0.015	2363392	31	14
15	OK	0.000	2367488	31	12
16	OK	0.000	2367488	31	13
17	OK	0.015	2363392	31	13
18	OK	0.000	2367488	31	14
19	OK	0.000	2367488	31	13
20	OK	0.000	2371584	31	14
21	OK	0.015	2379776	31	13
22	OK	0.015	2367488	31	14
23	OK	0.015	2367488	31	15

24	OK	0.000	2367488	38	15
25	OK	0.000	2367488	38	16
26	OK	0.000	2367488	38	15
27	OK	0.000	2367488	38	16
28	OK	0.000	2367488	38	17
29	OK	0.000	2371584	38	15
30	OK	0.000	2383872	38	16
31	OK	0.000	2367488	38	16
32	OK	0.000	2367488	38	17
33	OK	0.000	2367488	38	16
34	OK	0.000	2367488	38	17
35	OK	0.015	2367488	38	16
36	OK	0.000	2379776	38	17
37	OK	0.015	2363392	38	18
38	OK	0.015	2367488	38	15
39	OK	0.000	2371584	38	16
40	OK	0.000	2363392	38	15
41	OK	0.000	2383872	38	16
42	OK	0.000	2367488	38	17
43	OK	0.015	2363392	38	15
44	OK	0.000	2363392	38	16
45	OK	0.000	2383872	38	16
46	OK	0.000	2367488	38	17
47	OK	0.000	2367488	38	16
48	OK	0.000	2367488	38	17
49	OK	0.015	2367488	38	16
50	OK	0.000	2379776	38	17
51	OK	0.000	2367488	38	18
52	OK	0.015	2367488	38	16

53	OK	0.015	2379776	38	17
54	OK	0.000	2367488	38	16
55	OK	0.000	2367488	38	17
56	OK	0.015	2371584	38	18
57	OK	0.015	2363392	38	16
58	OK	0.015	2367488	38	17
59	OK	0.000	2363392	38	17
60	OK	0.000	2375680	38	18
61	OK	0.015	2367488	38	17
62	OK	0.000	2363392	38	18
63	OK	0.015	2371584	38	17
64	OK	0.000	2367488	38	18
65	OK	0.015	2367488	38	19
66	OK	0.000	2367488	45	18
67	OK	0.000	2367488	45	19
68	OK	0.000	2383872	45	18
69	OK	0.000	2379776	45	19
70	OK	0.031	2367488	45	20
71	OK	0.000	2367488	45	18
72	OK	0.000	2367488	45	19
73	OK	0.000	2367488	45	19
74	OK	0.000	2367488	45	20
75	OK	0.015	2367488	45	19
76	OK	0.000	2383872	45	20
77	OK	0.015	2363392	45	19
78	OK	0.000	2375680	45	20
79	OK	0.015	2367488	45	21
80	OK	0.015	2367488	45	18
81	OK	0.015	2371584	45	19

82	OK	0.000	2367488	45	18
83	OK	0.015	2367488	45	19
84	OK	0.000	2367488	45	20
85	OK	0.000	2367488	45	18
86	OK	0.000	2367488	45	19
87	OK	0.031	2371584	45	19
88	OK	0.000	2367488	45	20
89	OK	0.000	2371584	45	19
90	OK	0.031	2363392	45	20
91	OK	0.000	2379776	45	19
92	OK	0.015	2367488	45	20
93	OK	0.015	2371584	45	21
94	OK	0.015	2371584	45	19
95	OK	0.000	2367488	45	20
96	OK	0.000	2367488	45	19
97	OK	0.000	2367488	45	20
98	OK	0.000	2383872	45	21
99	OK	0.000	2367488	45	19
100	OK	0.015	2367488	45	20
101	OK	0.015	2367488	45	20
102	OK	0.000	2367488	45	21
103	OK	0.000	2371584	45	20
104	OK	0.000	2371584	45	21
105	OK	0.000	2367488	45	20
106	OK	0.000	2371584	45	21
107	OK	0.000	2367488	45	22
108	OK	0.000	2379776	45	18
109	OK	0.000	2367488	45	19
110	OK	0.015	2367488	45	18

111	OK	0.000	2367488	45	19
112	OK	0.031	2367488	45	20
113	OK	0.015	2367488	45	18
114	OK	0.000	2371584	45	19
115	OK	0.015	2371584	45	19
116	OK	0.015	2367488	45	20
117	OK	0.015	2387968	45	19
118	OK	0.000	2367488	45	20
119	OK	0.000	2363392	45	19
120	OK	0.015	2363392	45	20
121	OK	0.000	2367488	45	21
122	OK	0.015	2367488	45	18
123	OK	0.000	2363392	45	19
124	OK	0.000	2367488	45	18
125	OK	0.000	2367488	45	19
126	OK	0.031	2371584	45	20
127	OK	0.000	2383872	45	19
128	OK	0.000	2379776	45	20
129	OK	0.000	2367488	45	19
130	OK	0.000	2367488	45	20
131	OK	0.000	2363392	45	21
132	OK	0.015	2367488	45	19
133	OK	0.000	2379776	45	20
134	OK	0.015	2367488	45	20
135	OK	0.000	2363392	45	21
136	OK	0.000	2367488	45	18
137	OK	0.000	2367488	45	19
138	OK	0.000	2371584	45	20
139	OK	0.000	2371584	45	21

140	OK	0.000	2379776	45	21
141	OK	0.000	2371584	45	22
142	OK	0.015	2367488	45	19
143	OK	0.015	2363392	45	20
144	OK	0.000	2367488	45	19
145	OK	0.015	2363392	45	20
146	OK	0.015	2367488	45	21
147	OK	0.015	2367488	45	19
148	OK	0.000	2367488	45	20
149	OK	0.000	2367488	45	20
150	OK	0.000	2371584	45	21
151	OK	0.000	2383872	45	20
152	OK	0.000	2367488	45	21
153	OK	0.000	2371584	45	20
154	OK	0.000	2367488	45	21
155	OK	0.015	2367488	45	22
156	OK	0.000	2367488	45	19
157	OK	0.000	2363392	45	20
158	OK	0.031	2363392	45	19
159	OK	0.000	2367488	45	20
160	OK	0.000	2367488	45	21
161	OK	0.000	2371584	45	19
162	OK	0.000	2367488	45	20
163	OK	0.015	2371584	45	20
164	OK	0.000	2367488	45	21
165	OK	0.000	2363392	45	20
166	OK	0.000	2367488	45	21
167	OK	0.000	2367488	45	20
168	OK	0.000	2367488	45	21

169	OK	0.015	2367488	45	22
170	OK	0.000	2367488	45	19
171	OK	0.000	2367488	45	20
172	OK	0.000	2367488	45	19
173	OK	0.000	2371584	45	20
174	OK	0.015	2367488	45	21
175	OK	0.000	2371584	45	19
176	OK	0.000	2383872	45	20
177	OK	0.000	2379776	45	20
178	OK	0.015	2363392	45	21
179	OK	0.015	2363392	45	20
180	OK	0.000	2367488	45	21
181	OK	0.000	2367488	45	20
182	OK	0.015	2379776	45	21
183	OK	0.000	2379776	45	22
184	OK	0.015	2371584	45	20
185	OK	0.015	2367488	45	21
186	OK	0.000	2367488	45	20
187	OK	0.000	2383872	45	21
188	OK	0.000	2371584	45	22
189	OK	0.000	2367488	45	20
190	OK	0.000	2367488	45	21
191	OK	0.015	2367488	45	21
192	OK	0.000	2383872	45	22
193	OK	0.000	2367488	45	21
194	OK	0.000	2363392	45	22
195	OK	0.015	2367488	45	21
196	OK	0.000	2379776	45	22
197	OK	0.015	2367488	45	23

198	OK	0.015	2363392	221	55
199	OK	0.015	2371584	220	59
200	OK	0.015	2371584	220	46
201	OK	0.000	2363392	223	48
202	OK	0.000	2367488	226	45
203	OK	0.000	2363392	1786	502
204	OK	0.015	2363392	1785	555
205	OK	0.000	2375680	1785	445
206	OK	0.000	2359296	1845	365
207	OK	0.000	2367488	1847	363
208	OK	0.000	2396160	9555	3006
209	OK	0.015	2396160	9554	3297
210	OK	0.015	2387968	9554	2730
211	OK	0.000	2379776	9303	1888
212	OK	0.000	2375680	9984	1877
213	OK	0.000	2519040	37691	12907
214	OK	0.000	2519040	37690	13974
215	OK	0.000	2506752	37690	11820
216	OK	0.015	2424832	39602	7150
217	OK	0.015	2416640	38744	7125
218	OK	0.031	3088384	178903	63876
219	OK	0.031	3072000	178902	68889
220	OK	0.031	3072000	178902	58890
221	OK	0.046	2600960	185712	33049
222	OK	0.046	2617344	180580	33013
223	OK	0.343	9437184	1853240	724890
224	OK	0.343	9437184	1853239	773873
225	OK	0.328	9441280	1853239	675751
226	OK	0.328	4722688	1855624	324156

227	OK	0.312	4734976	1856715	324455
228	OK	0.593	15278080	3473125	1412256
229	OK	0.593	15278080	3473124	1501788
230	OK	0.593	15282176	3473124	1322578
231	OK	0.578	6688768	3603994	592172
232	OK	0.593	6684672	3646224	592525
233	OK	0.656	16769024	3888905	1589032
234	OK	0.656	16777216	3888904	1688889
235	OK	0.671	16777216	3888904	1488890
236	OK	0.640	7180288	3890628	661024
237	OK	0.640	7180288	3986010	661067

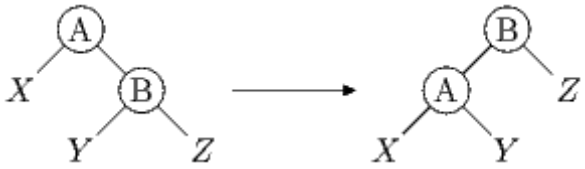
Делаю я левый поворот...

1.0 из 1.0 балла (оценивается)

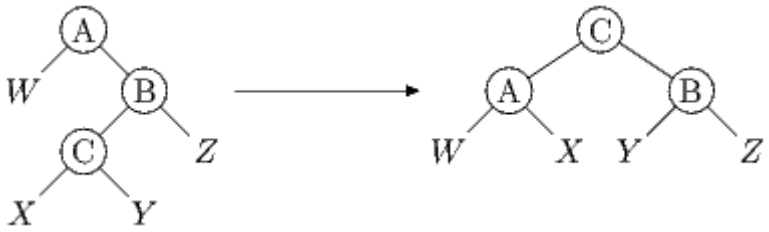
Имя входного файла:	input.txt
Имя выходного файла:	output.txt
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

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

Исходный код к задаче 2

```
#include <fstream>
#include <stack>

using namespace std;

int maximum(int a, int b) {
   if (a > b)
      return a;
   else
      return b;
}

class binary_tree {
public:
   struct Node {
      int value;
      int left_child;
      int right_child;
      int balance = -1;
      int height = 0;
      int number;
   };

   binary_tree(ifstream &input, int new_size) : size(new_size) {
      tree = new Node[size + 1];
      int K, L, R;
      for (int i = 1; i <= size; i++) {
         input >> K >> L >> R;
```

```
        tree[i] = {K, L, R, 0};
    }
    tree[0] = {0, 0, 0, 0, 0, 0};
    height(&tree[1]);
    balance(&tree[1]);
    rotateLeft(&tree[1]);
    int num = 1;
    numerate(&tree[1], &num);
}

void print_balance(ofstream &output) {
    for (int i = 1; i <= size; i++) {
        output << tree[i].balance << '\n';
    }
}

void print_tree(ofstream &output) {
    int number = 1;
    numerate(&tree[1], &number);
    output << size << '\n';
    print(output, &tree[1]);
}

void rotateRight(Node *node) {
    if (tree[node->left_child].balance == 1) {
        int B = node->left_child;
        int C = tree[B].right_child;
        int X = tree[C].right_child;
        int Y = tree[C].left_child;
        node->left_child = X;
        tree[B].left_child = Y;
        auto temp = *node;
        *node = tree[C];
        tree[C] = temp;
        node->right_child = C;
        node->left_child = B;
    } else {
        int B = node->left_child;
        int Y = tree[node->left_child].right_child;
        tree[B].right_child = B;
        Node temp = tree[B];
        node->left_child = Y;
        tree[B] = *node;
        *node = temp;
    }
}

void rotateLeft(Node *node) {
    if (tree[node->right_child].balance == -1) {
        int B = node->right_child;
        int C = tree[B].left_child;
        int X = tree[C].left_child;
        int Y = tree[C].right_child;
        node->right_child = X;
```

```
            tree[B].left_child = Y;
            auto temp = *node;
            *node = tree[C];
            tree[C] = temp;
            node->left_child = C;
            node->right_child = B;
        } else {
            int B = node->right_child;
            int Y = tree[node->right_child].left_child;
            tree[B].left_child = B;
            Node temp = tree[B];
            node->right_child = Y;
            tree[B] = *node;
            *node = temp;
        }
    }

private:
    Node *tree;
    int size;

    int height(Node *current_node) {
        if (current_node->right_child == 0 && current_node->left_child == 0) {
            current_node->height = 1;
            return 1;
        }
        int left_h = 0, right_h = 0;
        if (current_node->left_child != 0)
            left_h = height(&tree[current_node->left_child]);
        if (current_node->right_child != 0)
            right_h = height(&tree[current_node->right_child]);
        current_node->height = maximum(left_h, right_h) + 1;
        return current_node->height;
    }

    Node *find(Node *node, int x) {
        if (node->value == x || (node->right_child == 0 && node->left_child == 0))
            return node;
        if (node->value < x) {
            if (node->right_child == 0)
                return node;
            else
                return find(&tree[node->right_child], x);
        } else {
            if (node->left_child == 0)
                return node;
            else
                return find(&tree[node->left_child], x);
        }
    }

    void balance(Node *node) {
        int left_h, right_h;
        left_h = tree[node->left_child].height;
```

```
        right_h = tree[node->right_child].height;
        node->balance = right_h - left_h;
        if (left_h == 0 && right_h == 0)
            return;
        if (left_h != 0)
            balance(&tree[node->left_child]);
        if (right_h != 0)
            balance(&tree[node->right_child]);
    }

    void numerate(Node *current_node, int *current_number) {
        current_node->number = (*current_number)++;
        if (current_node->left_child != 0)
            numerate(&tree[current_node->left_child], current_number);
        if (current_node->right_child != 0)
            numerate(&tree[current_node->right_child], current_number);
    }

    void print(ofstream &output, Node *node) {
        output << node->value << ' ';
        if (node->left_child != 0)
            output << tree[node->left_child].number << ' ';
        else
            output << 0 << ' ';
        if (node->right_child != 0)
            output << tree[node->right_child].number << '\n';
        else
            output << 0 << '\n';
        if (node->left_child != 0)
            print(output, &tree[node->left_child]);
        if (node->right_child != 0)
            print(output, &tree[node->right_child]);
    }
};

int main() {
    ifstream input("input.txt");
    ofstream output("output.txt");

    int n;
    input >> n;
    binary_tree tree(input, n);
    tree.print_tree(output);
}
```

Бенчмарк к задаче 2

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.578	11980800	12083657	5694235
1	OK	0.015	2375680	37	12
2	OK	0.015	2363392	6	3

3	OK	0.015	2371584	11	3
4	OK	0.000	2383872	22	4
5	OK	0.000	2371584	19	6
6	OK	0.015	2371584	19	6
7	OK	0.015	2387968	19	6
8	OK	0.000	2371584	48	19
9	OK	0.000	2387968	58	29
10	OK	0.000	2387968	57	28
11	OK	0.015	2371584	48	19
12	OK	0.000	2371584	58	29
13	OK	0.000	2387968	57	28
14	OK	0.015	2371584	828	573
15	OK	0.000	2371584	1037	369
16	OK	0.015	2387968	828	573
17	OK	0.015	2371584	988	404
18	OK	0.000	2371584	1082	300
19	OK	0.000	2371584	1139	240
20	OK	0.000	2387968	930	377
21	OK	0.000	2371584	1190	280
22	OK	0.015	2375680	8184	5678
23	OK	0.015	2371584	10768	3637
24	OK	0.000	2367488	8206	5700
25	OK	0.000	2375680	9903	3928
26	OK	0.000	2387968	10814	3000
27	OK	0.015	2392064	11338	2400
28	OK	0.000	2375680	11138	3582
29	OK	0.000	2371584	10904	3851
30	OK	0.015	2424832	81951	56944
31	OK	0.031	2420736	110901	36274

32	OK	0.031	2428928	81971	56964
33	OK	0.015	2445312	99351	39719
34	OK	0.015	2457600	107882	30000
35	OK	0.015	2465792	113181	24000
36	OK	0.031	2408448	112799	37474
37	OK	0.015	2408448	114106	37576
38	OK	0.125	2977792	819273	569265
39	OK	0.156	2916352	1143615	361526
40	OK	0.125	2981888	819455	569447
41	OK	0.125	3182592	992441	396009
42	OK	0.125	3276800	1079125	300000
43	OK	0.140	3342336	1131016	240000
44	OK	0.156	2785280	1175194	377350
45	OK	0.156	2789376	1174192	378071
46	OK	1.312	8380416	8194244	5694235
47	OK	1.578	7716864	11753433	3632457
48	OK	1.250	8380416	8193883	5693874
49	OK	1.281	10379264	9926125	3963652
50	OK	1.296	11378688	10792079	3000000
51	OK	1.312	11980800	11312176	2400000
52	OK	1.500	6385664	12078250	3794039
53	OK	1.531	6389760	12083657	3795822

Вставка в АВЛ-дерево

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

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

Исходный код к задаче 3

```
#include <fstream>
#include <vector>

using namespace std;

int maximum(int a, int b) {
   if (a > b)
      return a;
   else
      return b;
}

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 = maximum(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 = maximum(h, height(node->left_child));
    }
    if (node->right_child != nullptr) {
        h = maximum(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 input("input.txt");
    ofstream output("output.txt");

    int n, x;
    char command;
    input >> n;

    AVL_tree tree(input, n);
    input >> x;
    tree.insert(x);
    tree.print_tree(output);
}
```

Бенчмарк к задаче 3

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.015	13242368	4011957	4011966
1	OK	0.015	2367488	20	24
2	OK	0.000	2371584	6	10
3	OK	0.000	2367488	14	18
4	OK	0.000	2367488	13	17
5	OK	0.000	2383872	21	25
6	OK	0.015	2367488	20	24
7	OK	0.000	2367488	20	24
8	OK	0.000	2371584	21	25
9	OK	0.000	2375680	20	24
10	OK	0.000	2367488	20	24

11	OK	0.000	2367488	28	32
12	OK	0.000	2367488	27	31
13	OK	0.000	2367488	27	31
14	OK	0.015	2371584	27	31
15	OK	0.000	2367488	35	39
16	OK	0.015	2367488	34	38
17	OK	0.015	2375680	34	38
18	OK	0.000	2367488	34	38
19	OK	0.015	2383872	34	38
20	OK	0.015	2367488	35	39
21	OK	0.000	2375680	34	38
22	OK	0.000	2367488	34	38
23	OK	0.000	2367488	34	38
24	OK	0.031	2363392	34	38
25	OK	0.015	2367488	35	39
26	OK	0.015	2371584	34	38
27	OK	0.000	2367488	34	38
28	OK	0.000	2371584	34	38
29	OK	0.000	2363392	34	38
30	OK	0.000	2371584	35	39
31	OK	0.015	2367488	34	38
32	OK	0.015	2363392	34	38
33	OK	0.000	2375680	34	38
34	OK	0.000	2379776	34	38
35	OK	0.000	2367488	42	46
36	OK	0.015	2367488	41	45
37	OK	0.000	2367488	41	45
38	OK	0.000	2371584	41	45
39	OK	0.015	2363392	41	45

40	OK	0.015	2363392	41	45
41	OK	0.000	2363392	42	46
42	OK	0.000	2371584	41	45
43	OK	0.015	2367488	41	45
44	OK	0.000	2367488	41	45
45	OK	0.015	2375680	41	45
46	OK	0.000	2379776	41	45
47	OK	0.015	2367488	42	46
48	OK	0.015	2363392	41	45
49	OK	0.015	2363392	41	45
50	OK	0.015	2367488	41	45
51	OK	0.000	2383872	41	45
52	OK	0.000	2367488	41	45
53	OK	0.000	2371584	42	46
54	OK	0.000	2371584	41	45
55	OK	0.000	2379776	41	45
56	OK	0.000	2363392	41	45
57	OK	0.031	2371584	41	45
58	OK	0.000	2363392	41	45
59	OK	0.015	2367488	42	46
60	OK	0.000	2367488	41	45
61	OK	0.000	2367488	41	45
62	OK	0.000	2367488	41	45
63	OK	0.000	2371584	41	45
64	OK	0.000	2367488	41	45
65	OK	0.000	2371584	42	46
66	OK	0.000	2363392	41	45
67	OK	0.015	2383872	41	45
68	OK	0.000	2363392	41	45

69	OK	0.015	2375680	41	45
70	OK	0.015	2367488	41	45
71	OK	0.015	2367488	50	54
72	OK	0.015	2379776	49	53
73	OK	0.000	2367488	49	53
74	OK	0.015	2367488	49	53
75	OK	0.015	2371584	49	53
76	OK	0.000	2379776	49	53
77	OK	0.000	2371584	50	54
78	OK	0.000	2367488	50	54
79	OK	0.000	2383872	49	53
80	OK	0.000	2367488	49	53
81	OK	0.000	2367488	49	53
82	OK	0.000	2375680	49	53
83	OK	0.015	2367488	49	53
84	OK	0.000	2379776	50	54
85	OK	0.000	2367488	50	54
86	OK	0.000	2367488	49	53
87	OK	0.000	2371584	49	53
88	OK	0.000	2367488	49	53
89	OK	0.000	2371584	49	53
90	OK	0.000	2367488	49	53
91	OK	0.000	2371584	50	54
92	OK	0.015	2367488	50	54
93	OK	0.000	2367488	49	53
94	OK	0.000	2375680	49	53
95	OK	0.015	2367488	49	53
96	OK	0.000	2367488	49	53
97	OK	0.000	2367488	49	53

98	OK	0.000	2367488	50	54
99	OK	0.000	2371584	58	62
100	OK	0.015	2379776	57	61
101	OK	0.000	2367488	57	61
102	OK	0.000	2367488	57	61
103	OK	0.000	2371584	57	61
104	OK	0.000	2359296	57	61
105	OK	0.000	2367488	58	62
106	OK	0.000	2375680	58	62
107	OK	0.000	2367488	58	62
108	OK	0.000	2367488	57	61
109	OK	0.000	2367488	57	61
110	OK	0.000	2367488	57	61
111	OK	0.000	2371584	57	61
112	OK	0.000	2363392	57	61
113	OK	0.015	2379776	58	62
114	OK	0.000	2371584	58	62
115	OK	0.000	2371584	58	62
116	OK	0.015	2363392	57	61
117	OK	0.000	2367488	57	61
118	OK	0.015	2375680	57	61
119	OK	0.000	2363392	57	61
120	OK	0.015	2367488	57	61
121	OK	0.000	2367488	58	62
122	OK	0.015	2367488	58	62
123	OK	0.015	2383872	58	62
124	OK	0.015	2379776	57	61
125	OK	0.000	2367488	57	61
126	OK	0.000	2383872	57	61

127	OK	0.015	2371584	57	61
128	OK	0.015	2383872	57	61
129	OK	0.015	2367488	58	62
130	OK	0.000	2375680	58	62
131	OK	0.015	2367488	58	62
132	OK	0.000	2367488	57	61
133	OK	0.000	2367488	57	61
134	OK	0.000	2379776	57	61
135	OK	0.015	2367488	57	61
136	OK	0.000	2379776	57	61
137	OK	0.000	2367488	58	62
138	OK	0.000	2383872	58	62
139	OK	0.000	2371584	58	62
140	OK	0.000	2383872	57	61
141	OK	0.015	2363392	57	61
142	OK	0.000	2371584	57	61
143	OK	0.015	2383872	57	61
144	OK	0.000	2367488	57	61
145	OK	0.031	2379776	58	62
146	OK	0.015	2387968	58	62
147	OK	0.015	2371584	58	62
148	OK	0.015	2367488	57	61
149	OK	0.000	2367488	57	61
150	OK	0.000	2383872	57	61
151	OK	0.015	2371584	57	61
152	OK	0.000	2367488	57	61
153	OK	0.015	2367488	58	62
154	OK	0.015	2375680	58	62
155	OK	0.000	2379776	58	62

156	OK	0.000	2363392	57	61
157	OK	0.000	2367488	57	61
158	OK	0.000	2363392	57	61
159	OK	0.000	2367488	57	61
160	OK	0.000	2363392	57	61
161	OK	0.000	2379776	58	62
162	OK	0.000	2371584	58	62
163	OK	0.000	2371584	58	62
164	OK	0.015	2367488	57	61
165	OK	0.000	2367488	57	61
166	OK	0.000	2375680	57	61
167	OK	0.015	2367488	57	61
168	OK	0.000	2367488	57	61
169	OK	0.015	2367488	58	62
170	OK	0.000	2379776	58	62
171	OK	0.015	2371584	58	62
172	OK	0.015	2367488	57	61
173	OK	0.000	2367488	57	61
174	OK	0.015	2371584	57	61
175	OK	0.000	2371584	57	61
176	OK	0.000	2367488	57	61
177	OK	0.000	2363392	58	62
178	OK	0.000	2371584	58	62
179	OK	0.000	2363392	58	62
180	OK	0.000	2367488	57	61
181	OK	0.000	2367488	57	61
182	OK	0.000	2363392	57	61
183	OK	0.000	2383872	57	61
184	OK	0.015	2379776	57	61

185	OK	0.000	2367488	58	62
186	OK	0.015	2371584	58	62
187	OK	0.000	2383872	58	62
188	OK	0.015	2367488	57	61
189	OK	0.000	2367488	57	61
190	OK	0.000	2371584	57	61
191	OK	0.000	2367488	57	61
192	OK	0.000	2363392	57	61
193	OK	0.015	2367488	58	62
194	OK	0.000	2367488	58	62
195	OK	0.000	2371584	58	62
196	OK	0.000	2379776	57	61
197	OK	0.000	2367488	57	61
198	OK	0.000	2371584	57	61
199	OK	0.015	2379776	57	61
200	OK	0.000	2367488	57	61
201	OK	0.000	2363392	58	62
202	OK	0.000	2375680	58	62
203	OK	0.000	2367488	58	62
204	OK	0.000	2379776	57	61
205	OK	0.000	2367488	57	61
206	OK	0.000	2367488	57	61
207	OK	0.000	2371584	57	61
208	OK	0.000	2379776	57	61
209	OK	0.000	2367488	58	62
210	OK	0.000	2371584	58	62
211	OK	0.000	2371584	58	62
212	OK	0.000	2363392	57	61
213	OK	0.000	2367488	57	61

214	OK	0.000	2371584	57	61
215	OK	0.000	2379776	57	61
216	OK	0.000	2379776	57	61
217	OK	0.015	2367488	58	62
218	OK	0.000	2379776	58	62
219	OK	0.015	2371584	58	62
220	OK	0.015	2367488	57	61
221	OK	0.000	2367488	57	61
222	OK	0.000	2383872	57	61
223	OK	0.015	2367488	57	61
224	OK	0.015	2363392	57	61
225	OK	0.031	2367488	58	62
226	OK	0.015	2371584	58	62
227	OK	0.000	2367488	58	62
228	OK	0.000	2383872	57	61
229	OK	0.000	2363392	57	61
230	OK	0.000	2367488	57	61
231	OK	0.015	2371584	57	61
232	OK	0.000	2367488	57	61
233	OK	0.000	2367488	58	62
234	OK	0.015	2371584	58	62
235	OK	0.000	2371584	240	245
236	OK	0.000	2367488	243	248
237	OK	0.000	2375680	1835	1841
238	OK	0.000	2379776	1815	1821
239	OK	0.000	2371584	1834	1840
240	OK	0.015	2400256	9951	9957
241	OK	0.000	2400256	9831	9837
242	OK	0.015	2412544	9854	9860

243	OK	0.015	2502656	38301	38308
244	OK	0.015	2498560	37664	37671
245	OK	0.015	2502656	39108	39115
246	OK	0.015	2506752	39190	39197
247	OK	0.062	3018752	183695	183703
248	OK	0.062	3014656	184258	184266
249	OK	0.062	3018752	185065	185073
250	OK	0.046	3002368	185428	185436
251	OK	0.062	2998272	185741	185749
252	OK	0.515	7540736	1900094	1900102
253	OK	0.484	7540736	1860225	1860233
254	OK	0.500	7540736	1899455	1899463
255	OK	0.500	7544832	1861088	1861096
256	OK	0.500	7540736	1942127	1942135
257	OK	0.500	7540736	1930200	1930208
258	OK	0.484	7540736	1861244	1861252
259	OK	0.890	12099584	3510448	3510457
260	OK	0.906	12095488	3650901	3650910
261	OK	0.906	12099584	3552374	3552383
262	OK	0.875	12103680	3435983	3435992
263	OK	0.906	12099584	3562689	3562698
264	OK	0.906	12099584	3521159	3521168
265	OK	0.890	12099584	3539149	3539158
266	OK	1.015	13238272	3985264	3985273
267	OK	1.000	13242368	3866892	3866901
268	OK	1.015	13238272	3942753	3942762
269	OK	0.984	13238272	3824263	3824272
270	OK	1.015	13242368	4011957	4011966
271	OK	1.000	13238272	3955420	3955429

272	OK	1.015	13238272	3946583	3946592
273	OK	1.000	13238272	3891536	3891545

Удаление из АВЛ-дерева

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

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

Исходный код к задаче 4

```
#include <fstream>
#include <vector>

using namespace std;

int maximum(int a, int b) {
   if (a > b)
      return a;
   else
      return b;
}

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 = maximum(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 = maximum(h, height(node->left_child));
    }
    if (node->right_child != nullptr) {
        h = maximum(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 input("input.txt");
    ofstream output("output.txt");

    int n, x;
    char command;
    input >> n;

    AVL_tree tree(input, n);
    input >> x;
    tree.remove(x);
    tree.print_tree(output);
```

Бенчмарк к задаче 4

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.015	13246464	4077288	4077255
1	OK	0.000	2371584	27	17
2	OK	0.000	2367488	13	3
3	OK	0.031	2383872	20	10
4	OK	0.000	2375680	20	10
5	OK	0.000	2387968	20	10
6	OK	0.015	2371584	20	10
7	OK	0.000	2367488	27	17
8	OK	0.000	2387968	27	17
9	OK	0.000	2383872	27	17
10	OK	0.000	2379776	34	24
11	OK	0.000	2371584	34	24
12	OK	0.015	2371584	34	24
13	OK	0.015	2371584	34	24
14	OK	0.000	2371584	34	24
15	OK	0.000	2387968	34	24
16	OK	0.000	2383872	34	24
17	OK	0.000	2387968	34	24
18	OK	0.000	2371584	34	24
19	OK	0.000	2371584	34	24
20	OK	0.000	2367488	34	24
21	OK	0.000	2383872	34	24
22	OK	0.015	2387968	34	24
23	OK	0.000	2371584	34	24
24	OK	0.000	2371584	34	24
25	OK	0.015	2367488	34	24
26	OK	0.000	2375680	41	31

27	OK	0.000	2371584	41	31
28	OK	0.000	2383872	41	31
29	OK	0.000	2371584	41	31
30	OK	0.015	2371584	41	31
31	OK	0.000	2371584	41	31
32	OK	0.015	2375680	41	31
33	OK	0.000	2367488	41	31
34	OK	0.000	2379776	41	31
35	OK	0.000	2371584	41	31
36	OK	0.015	2383872	41	31
37	OK	0.000	2383872	41	31
38	OK	0.015	2387968	41	31
39	OK	0.031	2379776	41	31
40	OK	0.015	2371584	41	31
41	OK	0.000	2371584	41	31
42	OK	0.000	2371584	41	31
43	OK	0.000	2371584	41	31
44	OK	0.000	2375680	41	31
45	OK	0.015	2383872	41	31
46	OK	0.000	2375680	41	31
47	OK	0.000	2371584	41	31
48	OK	0.000	2383872	41	31
49	OK	0.000	2371584	41	31
50	OK	0.015	2371584	41	31
51	OK	0.015	2371584	41	31
52	OK	0.000	2371584	41	31
53	OK	0.000	2367488	41	31
54	OK	0.000	2375680	41	31
55	OK	0.000	2371584	41	31

56	OK	0.000	2371584	48	38
57	OK	0.015	2379776	48	38
58	OK	0.031	2371584	48	38
59	OK	0.015	2371584	48	38
60	OK	0.000	2371584	48	38
61	OK	0.000	2387968	48	38
62	OK	0.015	2371584	48	38
63	OK	0.000	2371584	48	38
64	OK	0.000	2375680	48	38
65	OK	0.000	2371584	48	38
66	OK	0.000	2375680	48	38
67	OK	0.015	2392064	48	38
68	OK	0.000	2371584	48	38
69	OK	0.015	2371584	48	38
70	OK	0.015	2371584	48	38
71	OK	0.000	2383872	48	38
72	OK	0.000	2371584	48	38
73	OK	0.015	2371584	48	38
74	OK	0.015	2375680	48	38
75	OK	0.000	2371584	48	38
76	OK	0.000	2375680	48	38
77	OK	0.015	2375680	48	38
78	OK	0.000	2379776	48	38
79	OK	0.000	2371584	48	38
80	OK	0.000	2371584	55	45
81	OK	0.000	2371584	55	45
82	OK	0.015	2387968	55	45
83	OK	0.000	2371584	55	45
84	OK	0.000	2383872	55	45

85	OK	0.015	2375680	55	45
86	OK	0.000	2375680	55	45
87	OK	0.000	2371584	55	45
88	OK	0.015	2379776	55	45
89	OK	0.015	2383872	55	45
90	OK	0.015	2371584	55	45
91	OK	0.015	2383872	55	45
92	OK	0.000	2383872	55	45
93	OK	0.000	2367488	55	45
94	OK	0.000	2375680	55	45
95	OK	0.031	2383872	55	45
96	OK	0.015	2375680	55	45
97	OK	0.015	2371584	55	45
98	OK	0.000	2371584	55	45
99	OK	0.000	2375680	55	45
100	OK	0.000	2371584	55	45
101	OK	0.000	2371584	55	45
102	OK	0.015	2371584	55	45
103	OK	0.000	2379776	55	45
104	OK	0.015	2383872	55	45
105	OK	0.000	2371584	55	45
106	OK	0.000	2387968	55	45
107	OK	0.000	2371584	55	45
108	OK	0.015	2371584	55	45
109	OK	0.015	2375680	55	45
110	OK	0.015	2371584	55	45
111	OK	0.015	2387968	55	45
112	OK	0.000	2363392	55	45
113	OK	0.000	2371584	55	45

114	OK	0.015	2375680	55	45
115	OK	0.015	2371584	55	45
116	OK	0.000	2375680	55	45
117	OK	0.000	2367488	55	45
118	OK	0.000	2383872	55	45
119	OK	0.015	2371584	55	45
120	OK	0.000	2375680	55	45
121	OK	0.000	2371584	55	45
122	OK	0.031	2367488	55	45
123	OK	0.015	2375680	55	45
124	OK	0.000	2383872	55	45
125	OK	0.000	2371584	55	45
126	OK	0.015	2375680	55	45
127	OK	0.000	2363392	55	45
128	OK	0.000	2387968	55	45
129	OK	0.000	2383872	55	45
130	OK	0.000	2371584	55	45
131	OK	0.015	2371584	55	45
132	OK	0.015	2379776	55	45
133	OK	0.000	2367488	55	45
134	OK	0.000	2371584	55	45
135	OK	0.000	2375680	55	45
136	OK	0.000	2371584	55	45
137	OK	0.000	2371584	55	45
138	OK	0.000	2371584	55	45
139	OK	0.015	2375680	55	45
140	OK	0.015	2375680	55	45
141	OK	0.015	2371584	55	45
142	OK	0.015	2383872	55	45

143	OK	0.000	2367488	55	45
144	OK	0.015	2371584	55	45
145	OK	0.000	2383872	55	45
146	OK	0.000	2375680	55	45
147	OK	0.015	2375680	55	45
148	OK	0.015	2383872	55	45
149	OK	0.015	2371584	55	45
150	OK	0.000	2383872	55	45
151	OK	0.000	2367488	55	45
152	OK	0.015	2375680	55	45
153	OK	0.031	2371584	55	45
154	OK	0.000	2379776	55	45
155	OK	0.031	2371584	55	45
156	OK	0.015	2371584	55	45
157	OK	0.000	2367488	55	45
158	OK	0.015	2379776	55	45
159	OK	0.031	2375680	55	45
160	OK	0.000	2383872	55	45
161	OK	0.000	2367488	55	45
162	OK	0.000	2371584	55	45
163	OK	0.000	2387968	55	45
164	OK	0.000	2387968	55	45
165	OK	0.000	2371584	55	45
166	OK	0.000	2383872	55	45
167	OK	0.015	2367488	55	45
168	OK	0.000	2371584	55	45
169	OK	0.015	2375680	55	45
170	OK	0.000	2371584	55	45
171	OK	0.000	2375680	55	45

172	OK	0.015	2367488	55	45
173	OK	0.015	2371584	55	45
174	OK	0.031	2379776	55	45
175	OK	0.000	2375680	55	45
176	OK	0.015	2375680	55	45
177	OK	0.015	2371584	55	45
178	OK	0.000	2367488	55	45
179	OK	0.015	2371584	55	45
180	OK	0.000	2371584	55	45
181	OK	0.000	2379776	55	45
182	OK	0.015	2383872	55	45
183	OK	0.000	2387968	55	45
184	OK	0.000	2387968	55	45
185	OK	0.000	2383872	55	45
186	OK	0.015	2371584	55	45
187	OK	0.000	2375680	55	45
188	OK	0.015	2375680	55	45
189	OK	0.000	2383872	55	45
190	OK	0.015	2367488	55	45
191	OK	0.031	2367488	55	45
192	OK	0.015	2379776	55	45
193	OK	0.000	2379776	55	45
194	OK	0.015	2383872	55	45
195	OK	0.000	2371584	55	45
196	OK	0.000	2375680	55	45
197	OK	0.000	2371584	55	45
198	OK	0.000	2371584	55	45
199	OK	0.000	2371584	239	210
200	OK	0.000	2387968	235	208

201	OK	0.015	2392064	1797	1769
202	OK	0.015	2379776	1809	1781
203	OK	0.000	2371584	1831	1803
204	OK	0.000	2404352	9625	9597
205	OK	0.015	2420736	10026	9996
206	OK	0.000	2400256	9672	9642
207	OK	0.031	2506752	39459	39428
208	OK	0.015	2510848	39672	39641
209	OK	0.015	2523136	38780	38749
210	OK	0.015	2502656	38392	38361
211	OK	0.062	3022848	179425	179394
212	OK	0.046	3022848	182878	182845
213	OK	0.046	3014656	185986	185953
214	OK	0.046	3018752	186275	186244
215	OK	0.062	3018752	179082	179051
216	OK	0.484	7544832	1912349	1912319
217	OK	0.484	7544832	1864033	1864003
218	OK	0.484	7544832	1913940	1913908
219	OK	0.484	7540736	1879988	1879958
220	OK	0.484	7544832	1887512	1887482
221	OK	0.500	7544832	1932558	1932526
222	OK	0.500	7544832	1875036	1875006
223	OK	0.890	12107776	3597394	3597361
224	OK	0.890	12107776	3569973	3569940
225	OK	0.921	12103680	3512224	3512193
226	OK	0.906	12103680	3624329	3624296
227	OK	0.890	12103680	3523352	3523321
228	OK	0.921	12103680	3638077	3638044
229	OK	0.890	12107776	3512844	3512813

230	OK	1.015	13242368	4001320	4001287
231	OK	1.000	13242368	3954282	3954249
232	OK	1.000	13242368	3907814	3907783
233	OK	1.015	13242368	3983014	3982983
234	OK	1.000	13238272	4029895	4029862
235	OK	1.015	13246464	4046188	4046157
236	OK	1.000	13246464	4077288	4077255
237	OK	0.984	13242368	3902092	3902061

Упорядоченное множество на АВЛ-дереве

1.0 возможный балл (оценивается)

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 ($1 \le N \le 2 \cdot 10^5$) — 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	0 1 0 Y

C 4 C 6 D 5	N -1

Исходный код к задаче 5

```
#include <fstream>
#include <vector>

using namespace std;

int maximum(int a, int b) {
   if (a > b)
      return a;
   else
      return b;
}

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 = maximum(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 = maximum(h, height(node->left_child));
    }
    if (node->right_child != nullptr) {
        h = maximum(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 input("input.txt");
    ofstream output("output.txt");

    int n, x;
    char command;
    input >> n;

    AVL_tree tree;
    for (int i = 0; i < n; i++) {
        input >> command >> x;
        if (command == 'A') {
            tree.insert(x);
            output << tree.root_balance() << '\n';
        } else if (command == 'D') {
            tree.remove(x);
```

```
            output << tree.root_balance() << '\n';
        } else {
            if (tree.contains(x)) {
                output << 'Y' << '\n';
            } else {
                output << 'N' << '\n';
            }
        }

    }
}
```

Бенчмарк к задаче 5

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.500	12079104	2678110	731071
1	OK	0.000	2359296	33	19
2	OK	0.000	2359296	114	66
3	OK	0.000	2568192	154	90
4	OK	0.000	2580480	154	91
5	OK	0.015	2580480	154	90
6	OK	0.015	2588672	154	95
7	OK	0.015	2572288	154	91
8	OK	0.015	2568192	154	94
9	OK	0.015	2584576	154	95
10	OK	0.000	2359296	154	90
11	OK	0.000	2572288	154	90
12	OK	0.000	2592768	154	90
13	OK	0.015	2359296	154	95
14	OK	0.015	2355200	154	97
15	OK	0.000	2359296	154	94
16	OK	0.000	2363392	154	93
17	OK	0.000	2359296	154	90
18	OK	0.031	2371584	154	90
19	OK	0.000	2359296	154	98
20	OK	0.000	2359296	154	93

21	OK	0.000	2359296	154	92
22	OK	0.000	2359296	154	98
23	OK	0.265	4042752	1000008	616458
24	OK	0.265	4042752	1000008	622272
25	OK	0.265	4046848	1000008	625335
26	OK	0.281	4042752	1000008	628546
27	OK	0.265	4046848	1000008	631472
28	OK	0.281	4046848	1000008	632217
29	OK	0.281	4042752	1000008	631772
30	OK	0.281	4038656	1000008	631071
31	OK	0.265	4042752	1000008	630132
32	OK	0.281	4042752	1017957	630451
33	OK	0.265	4034560	1000008	616595
34	OK	0.265	4038656	1000008	622199
35	OK	0.265	4042752	1000008	625057
36	OK	0.281	4042752	1000008	628040
37	OK	0.281	4046848	1000008	631495
38	OK	0.265	4038656	1000008	632086
39	OK	0.265	4046848	1000008	631753
40	OK	0.281	4042752	1000008	630849
41	OK	0.281	4042752	1000008	630110
42	OK	0.281	4046848	1018151	630800
43	OK	0.015	2363392	756	369
44	OK	0.000	2363392	758	432
45	OK	0.000	2367488	1659	408
46	OK	0.000	2359296	723	383
47	OK	0.015	2359296	723	385
48	OK	0.000	2359296	723	415
49	OK	0.000	2371584	723	415

50	OK	0.000	2355200	1668	377
51	OK	0.000	2363392	1660	396
52	OK	0.000	2404352	5348	2337
53	OK	0.000	2404352	5350	2848
54	OK	0.015	2408448	10439	2648
55	OK	0.000	2375680	5238	2343
56	OK	0.000	2379776	5238	2465
57	OK	0.000	2379776	5238	2719
58	OK	0.015	2379776	5238	2719
59	OK	0.000	2392064	10450	2421
60	OK	0.015	2367488	10439	2405
61	OK	0.015	2613248	32784	12708
62	OK	0.015	2605056	32787	14896
63	OK	0.015	2605056	56716	12715
64	OK	0.000	2482176	31674	12778
65	OK	0.015	2469888	31674	13220
66	OK	0.000	2473984	31674	14383
67	OK	0.015	2469888	31674	14825
68	OK	0.015	2469888	56748	13671
69	OK	0.015	2453504	56716	13193
70	OK	0.046	3350528	162462	57855
71	OK	0.046	3346432	162466	68948
72	OK	0.046	3334144	258205	71756
73	OK	0.031	2850816	152067	59306
74	OK	0.031	2854912	152067	59903
75	OK	0.046	2842624	152067	66900
76	OK	0.031	2842624	152067	67497
77	OK	0.031	2838528	258312	70001
78	OK	0.046	2703360	258332	58111

79	OK	0.187	6844416	811002	274035
80	OK	0.187	6844416	811006	332612
81	OK	0.234	6844416	1222794	299942
82	OK	0.171	4612096	799892	286940
83	OK	0.171	4603904	799892	282227
84	OK	0.187	4612096	799892	324420
85	OK	0.187	4608000	799892	319707
86	OK	0.218	4608000	1222871	284516
87	OK	0.203	3903488	1223246	288111
88	OK	0.390	12079104	1888898	600000
89	OK	0.406	12075008	1888903	731071
90	OK	0.500	12079104	2677526	600067
91	OK	0.375	7262208	1777788	601696
92	OK	0.375	7262208	1777788	632768
93	OK	0.375	7262208	1777788	698302
94	OK	0.375	7266304	1777788	698303
95	OK	0.468	7262208	2678110	611713
96	OK	0.406	5660672	2677266	600286