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 Неделя №7

OpenEdu

Группа: P3217
Выполнил: Минин Александр

г. Санкт-Петербург
2018г.

Задача №1

Условие

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

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

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

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

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

Формат входного файла

Входной файл содержит описание двоичного дерева. В первой строке файла находится число N $(1 \le N \le 2 \cdot 10^5)$ — число вершин в дереве. В последующих N строках файла находятся описания вершин дерева. В $(i+1)$-ой строке файла $(1 \le i \le N)$ находится описание i-ой вершины, состоящее из трех чисел K_i, L_i, R_i, разделенных пробелами — ключа в i-ой вершине $(|K_i| \le 10^9)$, номера левого ребенка i-ой вершины ($i < L_i \le N$ или $L_i = 0$, если левого ребенка нет) и номера правого ребенка i-ой вершины ($i < R_i \le N$ или $R_i = 0$, если правого ребенка нет).

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

Формат выходного файла

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

Решение

```
package week7;

import mooc.EdxIO;

import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class Week7_1 {
    private static EdxIO edxIO;

    public static void main(String[] args) {
        edxIO = EdxIO.create();

        int n = edxIO.nextInt();
```

```
        AVLTree tree = new AVLTree(n);
        for (int i = 0; i < n; i++) {
            int key = edxIO.nextInt();
            int left = edxIO.nextInt() - 1;
            int right = edxIO.nextInt() - 1;

            Node<Long> currentNode = tree.getNodes()[i];
            // it can happen that it was created before if its' parent has already been parsed
            if (currentNode == null) {
                currentNode = new Node(key);
                tree.getNodes()[i] = currentNode;
            }

            if (left != -1) {
                if (tree.getNodes()[left] == null) {
                    Node node = new Node<Long>();
                    tree.getNodes()[left] = node;
                    node.setParent(tree.getNodes()[i]);
                }
                tree.getNodes()[i].setLeft(tree.getNodes()[left]);
            }

            if (right != -1) {
                if (tree.getNodes()[right] == null) {
                    Node node = new Node<Long>();
                    tree.getNodes()[right] = node;
                    node.setParent(tree.getNodes()[i]);
                }
                tree.getNodes()[i].setRight(tree.getNodes()[right]);
            }

            if (left == -1 && right == -1) {
                LinkedList<Node<Long>> curr = new LinkedList<>();
                while (currentNode != null) {
                    curr.push(currentNode);
                    currentNode = currentNode.getParent();
                }
                // setting height for all of the nodes on the path from
                // the first leaf
                while (curr.size() != 0) {
                    currentNode = curr.pop();
                    if (currentNode.getHeight() < curr.size())
                        currentNode.setHeight(curr.size());
                }
            }
        }

        for (int i = 0; i < n; i++) {
            edxIO.println(tree.getNodes()[i].getBalance());
        }

        edxIO.close();

    }

    static class AVLTree<T> {
        private Node<T>[] nodes;
        List<Node<T>> leaves = new ArrayList<>();
```

```
    private long size;

    public AVLTree(int n) {
        size = n;
        nodes = new Node[n];
    }

    public Node<T>[] getNodes() {
        return nodes;
    }

    public long getSize() {
        return size;
    }

    public void setSize(long size) {
        this.size = size;
    }
}

static class Node<T> {
    private T key;
    private Node<T> parent;
    private Node<T> left;
    private Node<T> right;

    private long height;

    Node() {

    }

    Node(T key) {
        this.key = key;
    }

    public long getHeight() {
        return height;
    }

    public void setHeight(long height) {
        this.height = height;
    }

    public T getKey() {
        return key;
    }

    public void setKey(T key) {
        this.key = key;
    }

    public Node<T> getParent() {
        return parent;
    }

    public void setParent(Node<T> parent) {
        this.parent = parent;
    }
```

```
        public Node<T> getLeft() {
            return left;
        }

        public void setLeft(Node<T> left) {
            this.left = left;
        }

        public Node<T> getRight() {
            return right;
        }

        public void setRight(Node<T> right) {
            this.right = right;
        }

        public long getBalance() {
            if (left != null && right != null) {
                return right.height - left.height;
            } else if (left == null && right != null) {
                return right.height + 1;
            } else if (left != null) {
                // right == null
                return -1 - left.height;
            }

            return 0;
        }
    }
}
```

Результаты

№ теста	Результат	Время, мс	Память	Размер входного файла	Размер выходного файла
Max		515	69279744	3986010	1688889
1	OK	109	21655552	46	19
2	OK	125	21688320	10	3
3	OK	125	21667840	17	6
4	OK	125	21692416	17	7
5	OK	140	21610496	24	9
6	OK	109	21663744	24	10
7	OK	109	21622784	24	9
8	OK	125	21696512	24	10
9	OK	109	21635072	24	11
10	OK	125	21655552	31	12
11	OK	140	21626880	31	13
12	OK	109	21655552	31	12

13	OK	125	21659648	31	13
14	OK	109	21622784	31	14
15	OK	109	21667840	31	12
16	OK	156	21630976	31	13
17	OK	125	21622784	31	13
18	OK	125	21643264	31	14
19	OK	140	21630976	31	13
20	OK	109	21626880	31	14
21	OK	171	21671936	31	13
22	OK	125	21676032	31	14
23	OK	125	21676032	31	15
24	OK	156	21659648	38	15
25	OK	125	21643264	38	16
26	OK	140	21626880	38	15
27	OK	156	21614592	38	16
28	OK	109	21643264	38	17
29	OK	109	21626880	38	15
30	OK	109	21643264	38	16
31	OK	156	21647360	38	16
32	OK	125	21622784	38	17
33	OK	109	21643264	38	16
34	OK	187	21647360	38	17
35	OK	109	21659648	38	16
36	OK	140	21635072	38	17
37	OK	140	21712896	38	18
38	OK	109	21622784	38	15
39	OK	125	21688320	38	16
40	OK	109	21622784	38	15
41	OK	125	21643264	38	16
42	OK	109	21643264	38	17
43	OK	109	21639168	38	15
44	OK	125	21626880	38	16
45	OK	109	21635072	38	16
46	OK	109	21647360	38	17
47	OK	156	21667840	38	16

48	OK	109	21639168	38	17
49	OK	109	21663744	38	16
50	OK	109	21630976	38	17
51	OK	140	21708800	38	18
52	OK	140	21635072	38	16
53	OK	125	21671936	38	17
54	OK	109	21651456	38	16
55	OK	140	21688320	38	17
56	OK	125	21647360	38	18
57	OK	93	21688320	38	16
58	OK	109	21647360	38	17
59	OK	125	21635072	38	17
60	OK	125	21708800	38	18
61	OK	140	21647360	38	17
62	OK	125	21635072	38	18
63	OK	125	21667840	38	17
64	OK	140	21647360	38	18
65	OK	109	21659648	38	19
66	OK	140	21671936	45	18
67	OK	125	21671936	45	19
68	OK	140	21635072	45	18
69	OK	93	21643264	45	19
70	OK	125	21639168	45	20
71	OK	125	21639168	45	18
72	OK	109	21618688	45	19
73	OK	109	21676032	45	19
74	OK	125	21676032	45	20
75	OK	125	21630976	45	19
76	OK	109	21663744	45	20
77	OK	125	21635072	45	19
78	OK	125	21618688	45	20
79	OK	125	21663744	45	21
80	OK	109	21630976	45	18
81	OK	109	21659648	45	19
82	OK	125	21659648	45	18

83	OK	140	21647360	45	19
84	OK	109	21659648	45	20
85	OK	109	21626880	45	18
86	OK	109	21610496	45	19
87	OK	125	21659648	45	19
88	OK	109	21651456	45	20
89	OK	109	21630976	45	19
90	OK	125	21635072	45	20
91	OK	156	21639168	45	19
92	OK	125	21651456	45	20
93	OK	125	21647360	45	21
94	OK	109	21651456	45	19
95	OK	125	21643264	45	20
96	OK	156	21663744	45	19
97	OK	156	21651456	45	20
98	OK	93	21635072	45	21
99	OK	125	21655552	45	19
100	OK	140	21630976	45	20
101	OK	125	21667840	45	20
102	OK	171	21630976	45	21
103	OK	109	21680128	45	20
104	OK	93	21647360	45	21
105	OK	125	21667840	45	20
106	OK	109	21602304	45	21
107	OK	140	21643264	45	22
108	OK	140	21712896	45	18
109	OK	125	21655552	45	19
110	OK	125	21626880	45	18
111	OK	109	21630976	45	19
112	OK	109	21614592	45	20
113	OK	109	21659648	45	18
114	OK	140	21659648	45	19
115	OK	109	21626880	45	19
116	OK	109	21622784	45	20
117	OK	109	21651456	45	19

118	OK	109	21680128	45	20
119	OK	156	21630976	45	19
120	OK	140	21708800	45	20
121	OK	125	21655552	45	21
122	OK	109	21643264	45	18
123	OK	140	21647360	45	19
124	OK	109	21630976	45	18
125	OK	125	21659648	45	19
126	OK	109	21647360	45	20
127	OK	125	21647360	45	19
128	OK	125	21647360	45	20
129	OK	125	21647360	45	19
130	OK	93	21626880	45	20
131	OK	109	21622784	45	21
132	OK	109	21704704	45	19
133	OK	125	21659648	45	20
134	OK	109	21626880	45	20
135	OK	109	21626880	45	21
136	OK	171	21639168	45	18
137	OK	140	21659648	45	19
138	OK	125	21680128	45	20
139	OK	140	21667840	45	21
140	OK	93	21647360	45	21
141	OK	109	21680128	45	22
142	OK	140	21622784	45	19
143	OK	125	21647360	45	20
144	OK	140	21688320	45	19
145	OK	171	21622784	45	20
146	OK	93	21594112	45	21
147	OK	125	21667840	45	19
148	OK	109	21630976	45	20
149	OK	125	21639168	45	20
150	OK	93	21647360	45	21
151	OK	109	21647360	45	20
152	OK	93	21630976	45	21

153	OK	109	21643264	45	20
154	OK	125	21647360	45	21
155	OK	109	21659648	45	22
156	OK	109	21696512	45	19
157	OK	109	21639168	45	20
158	OK	109	21651456	45	19
159	OK	125	21692416	45	20
160	OK	171	21647360	45	21
161	OK	109	21639168	45	19
162	OK	109	21663744	45	20
163	OK	125	21647360	45	20
164	OK	125	21643264	45	21
165	OK	140	21700608	45	20
166	OK	109	21622784	45	21
167	OK	109	21676032	45	20
168	OK	125	21704704	45	21
169	OK	125	21606400	45	22
170	OK	109	21635072	45	19
171	OK	109	21667840	45	20
172	OK	109	21667840	45	19
173	OK	140	21630976	45	20
174	OK	125	21647360	45	21
175	OK	109	21655552	45	19
176	OK	125	21643264	45	20
177	OK	125	21663744	45	20
178	OK	125	21643264	45	21
179	OK	109	21614592	45	20
180	OK	125	21630976	45	21
181	OK	125	21594112	45	20
182	OK	109	21622784	45	21
183	OK	109	21635072	45	22
184	OK	125	21647360	45	20
185	OK	125	21667840	45	21
186	OK	125	21643264	45	20
187	OK	125	21639168	45	21

188	OK	125	21630976	45	22
189	OK	109	21635072	45	20
190	OK	125	21622784	45	21
191	OK	125	21635072	45	21
192	OK	125	21696512	45	22
193	OK	125	21635072	45	21
194	OK	125	21626880	45	22
195	OK	109	21618688	45	21
196	OK	156	21635072	45	22
197	OK	109	21639168	45	23
198	OK	109	21663744	221	55
199	OK	140	21651456	220	59
200	OK	109	21647360	220	46
201	OK	125	21655552	223	48
202	OK	109	21635072	226	45
203	OK	109	21684224	1786	502
204	OK	109	21626880	1785	555
205	OK	140	21704704	1785	445
206	OK	140	21659648	1845	365
207	OK	109	21680128	1847	363
208	OK	125	21880832	9555	3006
209	OK	125	21868544	9554	3297
210	OK	125	21909504	9554	2730
211	OK	125	21905408	9303	1888
212	OK	125	21897216	9984	1877
213	OK	109	23392256	37691	12907
214	OK	140	23339008	37690	13974
215	OK	125	23371776	37690	11820
216	OK	125	23638016	39602	7150
217	OK	171	23707648	38744	7125
218	OK	187	24080384	178903	63876
219	OK	140	24104960	178902	68889
220	OK	140	24010752	178902	58890
221	OK	171	25681920	185712	33049
222	OK	140	25960448	180580	33013

223	OK	234	33005568	1853240	724890
224	OK	250	32903168	1853239	773873
225	OK	218	32694272	1853239	675751
226	OK	296	52658176	1855624	324156
227	OK	296	51408896	1856715	324455
228	OK	312	42127360	3473125	1412256
229	OK	296	42262528	3473124	1501788
230	OK	296	42242048	3473124	1322578
231	OK	500	68329472	3603994	592172
232	OK	421	67821568	3646224	592525
233	OK	296	44027904	3888905	1589032
234	OK	281	43855872	3888904	1688889
235	OK	328	43966464	3888904	1488890
236	OK	515	69279744	3890628	661024
237	OK	406	68743168	3986010	661067

Задача №2

Условие

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.

Решение

package week7;

```java
import mooc.EdxIO;

import java.util.LinkedList;
import java.util.Queue;

public class Week7_2 {
    private static EdxIO edxIO;

    public static void main(String[] args) {
        edxIO = EdxIO.create();

        int n = edxIO.nextInt();

        Node<Long>[] nodes = new Node[n];

        AVLTree tree = new AVLTree(n);
        for (int i = 0; i < n; i++) {
            int key = edxIO.nextInt();
            int left = edxIO.nextInt() - 1;
            int right = edxIO.nextInt() - 1;

            Node<Long> currentNode = nodes[i];
            // it can happen that it was created before if its' parent has already been parsed
            if (currentNode == null) {
                currentNode = new Node(key);
                nodes[i] = currentNode;
            } else {
                nodes[i].setKey((long) key);
            }

            if (left != -1) {
                if (nodes[left] == null) {
                    Node node = new Node<Long>();
                    nodes[left] = node;
                    node.setParent(nodes[i]);
                }
                nodes[i].setLeft(nodes[left]);
            }

            if (right != -1) {
                if (nodes[right] == null) {
                    Node node = new Node<Long>();
                    nodes[right] = node;
                    node.setParent(nodes[i]);
                }
                nodes[i].setRight(nodes[right]);
            }
            if (i == 0) {
                tree.setRoot(nodes[i]);
            }


            if (left == -1 && right == -1) {
                LinkedList<Node<Long>> curr = new LinkedList<>();
                while (currentNode != null) {
                    curr.push(currentNode);
                    currentNode = currentNode.getParent();
                }
```

```
            // setting height for all of the nodes on the path from
            // the first leaf
            while (curr.size() != 0) {
                currentNode = curr.pop();
                if (currentNode.getHeight() < curr.size())
                    currentNode.setHeight(curr.size());
            }
        }
    }

//      for (int i = 0; i < n; i++) {
//          edxIO.println(tree.getNodes()[i].getBalance());
//      }
    System.out.println(tree.getRoot().getBalance());
    if (tree.getRoot().getRight().getBalance() == -1) {
        tree.bigLeftTurn(tree.getRoot());
    } else {
        tree.leftRotation(tree.getRoot());
    }

//      tree.print(edxIO);
    edxIO.println(n);
    tree.print(edxIO);
    edxIO.close();
  }

  static class AVLTree<T extends Comparable> {
    private long size;
    private Node<T> root;

    public AVLTree(int n) {
        size = n;
    }

    public long getSize() {
        return size;
    }

    public void setSize(long size) {
        this.size = size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public Node lookup(T key) {
        if (isEmpty()) {
            return null;
        }

        Node<T> currNode = root;
        while (
                currNode.getKey().compareTo(key) != 0
                        && currNode.getKey().compareTo(key) < 0 && currNode.getRight() != null
                        && currNode.getKey().compareTo(key) > 0 && currNode.getLeft() != null
        ) {
            if (currNode.getKey().compareTo(key) < 0) {
```

```
            currNode = currNode.getRight();
        } else if (currNode.getKey().compareTo(key) > 0) {
            currNode = currNode.getLeft();
        } else {
            throw new IllegalStateException("[DEBUG] Just can't happen");
        }
    }
    return currNode;
}


public void print(EdxIO edxIO) {
    Node node = root;
    Queue<Node> queue = new LinkedList<>();
    queue.add(node);
    int c = 1;
    while (queue.size() != 0) {
        node = queue.poll();
        edxIO.print(node.getKey() + " ");
        if (node.hasLeft()) {
            queue.add(node.getLeft());
            c++;
            edxIO.print(c + " ");
        } else {
            edxIO.print("0 ");
        }
        if (node.hasRight()) {
            queue.add(node.getRight());
            c++;
            edxIO.print(c);
        } else {
            edxIO.print("0");
        }
        edxIO.println();
    }
}

private void leftRotation(Node x) {
    Node parent = x.getParent();
    Node T1 = x.getLeft();
    Node y = x.getRight();
    Node T2 = y.getLeft();
    Node T3 = y.getRight();

    y.setParent(parent);
    if (parent != null) {
        if (x.equals(parent.getLeft())) {
            parent.setLeft(y);
        } else {
            parent.setRight(y);
        }
    } else {
        root = y;
    }

    y.setLeft(x);
    x.setParent(y);
```

```
        x.setLeft(T1);
        if (T1 != null) {
            T1.setParent(x);
        }

        x.setRight(T2);
        if (T2 != null) {
            T2.setParent(x);
        }

        y.setRight(T3);
        if (T3 != null) {
            T3.setParent(y);
        }

        y.setHeight(max(y));
        x.setHeight(max(x));
    }

    private long max(Node node) {
        return Long.max(
                node.getLeft() == null ? 0 : node.getLeft().getHeight(),
                node.getRight() == null ? 1 : node.getRight().getHeight() + 1
        );
    }

    private void rightRotation(Node y) {
        Node parent = y.getParent();
        Node x = y.getLeft();
        Node T3 = y.getRight();
        Node T1 = x.getLeft();
        Node T2 = x.getRight();

        x.setParent(parent);
        if (parent != null) {
            if (y.equals(parent.getLeft())) {
                parent.setLeft(x);
            } else {
                parent.setRight(x);
            }
        } else {
            root = x;
        }

        x.setRight(y);
        y.setParent(x);

        x.setLeft(T1);
        if (T1 != null) {
            T1.setParent(x);
        }

        y.setLeft(T2);
        if (T2 != null) {
            T2.setParent(y);
        }

        y.setRight(T3);
        if (T3 != null) {
```

```
            T3.setParent(y);
        }
    }

    public Node<T> getRoot() {
        return root;
    }

    public void setRoot(Node<T> root) {
        this.root = root;
    }

    private void bigLeftTurn(Node z) {
        Node y = z.getRight();
        rightRotation(y);
        leftRotation(z);
    }
}

static class Node<T> {
    private T key;
    private Node<T> parent;
    private Node<T> left;
    private Node<T> right;

    private long height;

    Node() {

    }

    Node(T key) {
        this.key = key;
    }

    public long getHeight() {
        return height;
    }

    public void setHeight(long height) {
        this.height = height;
    }

    public T getKey() {
        return key;
    }

    public void setKey(T key) {
        this.key = key;
    }

    public Node<T> getParent() {
        return parent;
    }

    public void setParent(Node<T> parent) {
        this.parent = parent;
    }
```

```
        public Node<T> getLeft() {
            return left;
        }

        public void setLeft(Node<T> left) {
            this.left = left;
        }

        public Node<T> getRight() {
            return right;
        }

        public void setRight(Node<T> right) {
            this.right = right;
        }

        public boolean hasLeft() {
            return left != null;
        }

        public boolean hasRight() {
            return right != null;
        }

        public long getBalance() {
            if (left != null && right != null) {
                return right.height - left.height;
            } else if (left == null && right != null) {
                return right.height + 1;
            } else if (left != null) {
                // right == null
                return -1 - left.height;
            }

            return 0;
        }
    }
}
```

Результаты

№ теста	Результат	Время, мс	Память	Размер входного файла	Размер выходного файла
Max		1031	96489472	3986416	3986416
1	OK	125	21639168	54	54
2	OK	125	21696512	24	24
3	OK	156	21721088	24	24
4	OK	125	21696512	31	31
5	OK	109	21741568	45	45
6	OK	109	21716992	45	45
7	OK	125	21696512	45	45
8	OK	140	21700608	45	45
9	OK	140	21671936	52	52
10	OK	109	21700608	52	52
11	OK	140	21688320	52	52
12	OK	125	21614592	52	52
13	OK	109	21659648	52	52
14	OK	187	21745664	52	52
15	OK	140	21716992	59	59
16	OK	109	21659648	59	59
17	OK	93	21696512	59	59
18	OK	109	21663744	59	59
19	OK	109	21708800	66	66
20	OK	125	21692416	75	75
21	OK	140	21700608	75	75
22	OK	140	21688320	75	75
23	OK	140	21774336	75	75
24	OK	109	21635072	75	75
25	OK	125	21630976	75	75
26	OK	156	21700608	75	75
27	OK	187	21700608	75	75
28	OK	109	21692416	75	75
29	OK	125	21684224	75	75
30	OK	125	21643264	75	75
31	OK	109	21655552	75	75

32	OK	125	21708800	75	75
33	OK	125	21741568	75	75
34	OK	125	21725184	75	75
35	OK	156	21704704	75	75
36	OK	125	21692416	75	75
37	OK	125	21688320	75	75
38	OK	125	21680128	75	75
39	OK	140	21729280	75	75
40	OK	93	21704704	75	75
41	OK	125	21684224	75	75
42	OK	171	21676032	75	75
43	OK	109	21708800	75	75
44	OK	125	21716992	75	75
45	OK	125	21659648	75	75
46	OK	156	21671936	75	75
47	OK	125	21757952	75	75
48	OK	140	21659648	75	75
49	OK	140	21671936	75	75
50	OK	125	21712896	75	75
51	OK	156	21762048	75	75
52	OK	125	21688320	84	84
53	OK	140	21704704	84	84
54	OK	109	21684224	84	84
55	OK	109	21680128	84	84
56	OK	125	21692416	84	84
57	OK	125	21708800	84	84
58	OK	125	21725184	84	84
59	OK	156	21729280	84	84
60	OK	109	21684224	84	84
61	OK	125	21630976	84	84
62	OK	140	21684224	84	84
63	OK	171	21663744	84	84
64	OK	125	21692416	84	84
65	OK	125	21733376	84	84
66	OK	109	21688320	84	84

67	OK	93	21680128	84	84
68	OK	140	21704704	84	84
69	OK	140	21659648	84	84
70	OK	140	21671936	84	84
71	OK	125	21696512	84	84
72	OK	140	21696512	84	84
73	OK	140	21622784	84	84
74	OK	125	21676032	84	84
75	OK	125	21721088	84	84
76	OK	93	21716992	84	84
77	OK	125	21700608	84	84
78	OK	125	21712896	84	84
79	OK	109	21630976	84	84
80	OK	125	21692416	84	84
81	OK	156	21688320	84	84
82	OK	125	21737472	84	84
83	OK	109	21651456	84	84
84	OK	125	21737472	84	84
85	OK	125	21712896	84	84
86	OK	140	21753856	84	84
87	OK	125	21659648	84	84
88	OK	140	21696512	84	84
89	OK	125	21684224	84	84
90	OK	125	21774336	84	84
91	OK	125	21725184	84	84
92	OK	125	21643264	84	84
93	OK	125	21721088	84	84
94	OK	125	21725184	84	84
95	OK	140	21659648	84	84
96	OK	125	21708800	84	84
97	OK	109	21741568	84	84
98	OK	125	21692416	84	84
99	OK	140	21655552	84	84
100	OK	109	21676032	84	84
101	OK	109	21688320	84	84

102	OK	125	21667840	84	84
103	OK	171	21721088	84	84
104	OK	171	21663744	84	84
105	OK	156	21729280	84	84
106	OK	140	21729280	84	84
107	OK	109	21712896	84	84
108	OK	140	21716992	84	84
109	OK	125	21704704	84	84
110	OK	125	21680128	84	84
111	OK	125	21696512	84	84
112	OK	125	21712896	84	84
113	OK	109	21696512	84	84
114	OK	125	21708800	84	84
115	OK	125	21700608	84	84
116	OK	125	21655552	84	84
117	OK	109	21729280	84	84
118	OK	140	21659648	84	84
119	OK	140	21688320	84	84
120	OK	125	21733376	84	84
121	OK	109	21712896	84	84
122	OK	109	21700608	84	84
123	OK	125	21712896	84	84
124	OK	125	21721088	84	84
125	OK	125	21671936	84	84
126	OK	140	21671936	84	84
127	OK	125	21692416	84	84
128	OK	109	21659648	84	84
129	OK	140	21696512	84	84
130	OK	125	21704704	84	84
131	OK	109	21712896	84	84
132	OK	125	21737472	93	93
133	OK	109	21684224	93	93
134	OK	156	21626880	93	93
135	OK	140	21696512	93	93
136	OK	125	21712896	93	93

137	OK	109	21671936	93	93
138	OK	156	21671936	93	93
139	OK	125	21708800	93	93
140	OK	125	21667840	93	93
141	OK	125	21700608	93	93
142	OK	140	21712896	93	93
143	OK	125	21663744	93	93
144	OK	109	21696512	93	93
145	OK	109	21680128	93	93
146	OK	125	21671936	93	93
147	OK	156	21659648	93	93
148	OK	125	21663744	93	93
149	OK	109	21696512	93	93
150	OK	109	21712896	93	93
151	OK	156	21692416	93	93
152	OK	93	21618688	93	93
153	OK	125	21700608	93	93
154	OK	171	21676032	93	93
155	OK	109	21684224	93	93
156	OK	125	21716992	93	93
157	OK	140	21680128	93	93
158	OK	125	21680128	93	93
159	OK	156	21688320	93	93
160	OK	109	21721088	93	93
161	OK	109	21712896	93	93
162	OK	140	21712896	93	93
163	OK	125	21716992	93	93
164	OK	156	21684224	93	93
165	OK	140	21671936	93	93
166	OK	109	21655552	93	93
167	OK	109	21700608	93	93
168	OK	156	21762048	93	93
169	OK	109	21708800	93	93
170	OK	140	21667840	93	93
171	OK	125	21700608	93	93

172	OK	109	21692416	93	93
173	OK	125	21692416	93	93
174	OK	109	21708800	93	93
175	OK	125	21712896	93	93
176	OK	156	21647360	93	93
177	OK	125	21696512	93	93
178	OK	125	21692416	93	93
179	OK	109	21680128	93	93
180	OK	125	21757952	93	93
181	OK	93	21700608	93	93
182	OK	125	21655552	93	93
183	OK	109	21721088	93	93
184	OK	140	21725184	93	93
185	OK	125	21700608	93	93
186	OK	125	21696512	93	93
187	OK	140	21700608	93	93
188	OK	140	21663744	93	93
189	OK	140	21708800	93	93
190	OK	140	21729280	93	93
191	OK	109	21692416	93	93
192	OK	125	21708800	93	93
193	OK	109	21704704	93	93
194	OK	125	21659648	93	93
195	OK	156	21708800	93	93
196	OK	140	21692416	93	93
197	OK	109	21692416	93	93
198	OK	109	21696512	93	93
199	OK	156	21712896	93	93
200	OK	109	21680128	93	93
201	OK	125	21712896	93	93
202	OK	125	21692416	93	93
203	OK	93	21684224	93	93
204	OK	125	21737472	93	93
205	OK	171	21692416	93	93
206	OK	140	21667840	93	93

207	OK	109	21708800	93	93
208	OK	109	21712896	93	93
209	OK	125	21688320	93	93
210	OK	125	21729280	93	93
211	OK	140	21721088	93	93
212	OK	109	21696512	93	93
213	OK	125	21741568	93	93
214	OK	125	21688320	93	93
215	OK	125	21684224	93	93
216	OK	140	21725184	93	93
217	OK	109	21708800	93	93
218	OK	109	21655552	93	93
219	OK	140	21725184	93	93
220	OK	125	21696512	93	93
221	OK	109	21716992	93	93
222	OK	140	21708800	93	93
223	OK	140	21700608	93	93
224	OK	125	21663744	93	93
225	OK	109	21671936	93	93
226	OK	187	21721088	93	93
227	OK	125	21671936	93	93
228	OK	125	21700608	93	93
229	OK	109	21688320	93	93
230	OK	109	21671936	93	93
231	OK	125	21692416	93	93
232	OK	125	21716992	93	93
233	OK	140	21676032	93	93
234	OK	125	21708800	93	93
235	OK	125	21716992	93	93
236	OK	109	21708800	93	93
237	OK	109	21692416	93	93
238	OK	140	21696512	93	93
239	OK	156	21651456	93	93
240	OK	125	21749760	93	93
241	OK	125	21671936	93	93

242	OK	125	21671936	93	93
243	OK	125	21700608	93	93
244	OK	125	21684224	93	93
245	OK	171	21680128	93	93
246	OK	156	21708800	93	93
247	OK	109	21721088	93	93
248	OK	125	21676032	93	93
249	OK	125	21684224	93	93
250	OK	125	21712896	93	93
251	OK	125	21700608	93	93
252	OK	125	21716992	220	220
253	OK	125	21807104	1798	1798
254	OK	140	21729280	1842	1842
255	OK	140	22188032	9606	9606
256	OK	156	22204416	9569	9569
257	OK	140	22192128	9662	9662
258	OK	140	22171648	9777	9777
259	OK	171	24535040	37717	37717
260	OK	171	24649728	37874	37874
261	OK	171	24702976	39268	39268
262	OK	156	24621056	39470	39470
263	OK	187	29683712	181024	181024
264	OK	250	29155328	183405	183405
265	OK	203	29663232	180784	180784
266	OK	187	29339648	183152	183152
267	OK	218	31436800	181246	181246
268	OK	203	28778496	180886	180886
269	OK	640	67743744	1843051	1843051
270	OK	656	68390912	1888721	1888721
271	OK	703	67817472	1866794	1866794
272	OK	656	68308992	1898864	1898864
273	OK	656	68743168	1908693	1908693
274	OK	640	67612672	1881384	1881384
275	OK	906	79777792	3526463	3526463
276	OK	984	79847424	3559824	3559824

277	OK	921	80211968	3598289	3598289
278	OK	953	79781888	3590402	3590402
279	OK	968	80973824	3567894	3567894
280	OK	875	80379904	3566660	3566660
281	OK	875	80736256	3487414	3487414
282	OK	1031	96489472	3874088	3874088
283	OK	921	89333760	3978208	3978208
284	OK	906	90828800	3957801	3957801
285	OK	1000	90329088	3978349	3978349
286	OK	968	88092672	3986416	3986416
287	OK	906	84172800	3933437	3933437
288	OK	906	83226624	3926735	3926735

Задача №3

Условие

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.

Решение

```
package week7;

import mooc.EdxIO;

import java.util.LinkedList;
import java.util.Queue;

public class Week7_3 {
```

```
private static EdxIO edxIO;

public static void main(String[] args) {
    edxIO = EdxIO.create();

    int n = edxIO.nextInt();

    Node<Long>[] nodes = new Node[n];

    AVLTree tree = new AVLTree(n);
    for (int i = 0; i < n; i++) {
        long key = edxIO.nextLong();
        int left = edxIO.nextInt() - 1;
        int right = edxIO.nextInt() - 1;

        Node<Long> currentNode = nodes[i];
        // it can happen that it was created before if its' parent has already been parsed
        if (currentNode == null) {
            currentNode = new Node(key);
            nodes[i] = currentNode;
        } else {
            nodes[i].setKey((long) key);
        }

        if (left != -1) {
            if (nodes[left] == null) {
                Node<Long> node = new Node<Long>();
                nodes[left] = node;
                node.setParent(nodes[i]);
            }
            nodes[i].setLeft(nodes[left]);
        }

        if (right != -1) {
            if (nodes[right] == null) {
                Node<Long> node = new Node<Long>();
                nodes[right] = node;
                node.setParent(nodes[i]);
            }
            nodes[i].setRight(nodes[right]);
        }
        if (i == 0) {
            tree.setRoot(nodes[i]);
        }


        if (left == -1 && right == -1) {
            LinkedList<Node<Long>> curr = new LinkedList<>();
            while (currentNode != null) {
                curr.push(currentNode);
                currentNode = currentNode.getParent();
            }
            // setting height for all of the nodes on the path from
            // the first leaf
            while (curr.size() != 0) {
                currentNode = curr.pop();
                if (currentNode.getHeight() < curr.size())
                    currentNode.setHeight(curr.size());
```

```
            }
        }
    }

    long key = edxIO.nextLong();
    tree.insert(key);

    edxIO.println(tree.getSize());
    tree.print(edxIO);
    edxIO.close();
}

static class AVLTree<T extends Comparable> {
    private long size;
    private Node<T> root;

    public AVLTree(int n) {
        size = n;
    }

    public long getSize() {
        return size;
    }

    public void setSize(long size) {
        this.size = size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public Node<T> lookup(T key) {
        Node<T> currNode = root;
        while (currNode != null &&
                currNode.getKey().compareTo(key) != 0
                && (currNode.getKey().compareTo(key) < 0 && currNode.getRight() != null
                || currNode.getKey().compareTo(key) > 0 && currNode.getLeft() != null)
        ) {
            if (currNode.getKey().compareTo(key) < 0) {
                currNode = currNode.getRight();
            } else if (currNode.getKey().compareTo(key) > 0) {
                currNode = currNode.getLeft();
            } else {
                throw new IllegalStateException("[DEBUG] Just can't happen");
            }
        }
        return currNode;
    }

    public boolean contains(T key) {
        Node<T> node = lookup(key);
        return node != null && key.compareTo(node.getKey()) == 0;
    }

    public void insert(T key) {
        if (isEmpty()) {
            root = new Node<>(key);
```

```
    } else {
        Node<T> lookupNode = lookup(key);
        Node<T> newNode = new Node<>(key, lookupNode);
        if (lookupNode.getKey().compareTo(key) < 0) {
            lookupNode.setRight(newNode);
        } else if (lookupNode.getKey().compareTo(key) > 0) {
            lookupNode.setLeft(newNode);
        } else {
            return;
        }

        recalculateHeightsStartingFrom(newNode);

        Node currNode = lookupNode;
        while (currNode != null) {
            currNode.setHeight(max(currNode));
            int balance = getBalance(currNode);

            if (balance < -1 && key.compareTo(currNode.getLeft().getKey()) < 0) {
                rightRotation(currNode);
            } else if (balance > 1 && key.compareTo(currNode.getRight().getKey()) > 0) {
                leftRotation(currNode);
            } else if (balance < -1 && key.compareTo(currNode.getLeft().getKey()) > 0) {
                leftRotation(currNode.getLeft());
                rightRotation(currNode);
            } else if (balance > 1 && key.compareTo(currNode.getRight().getKey()) < 0) {
                rightRotation(currNode.getRight());
                leftRotation(currNode);
            }
            currNode = currNode.getParent();
        }
    }
    size++;
}

private void balanceStartingFrom(Node<T> node) {
    recalculateHeightsStartingFrom(node);

    Node currNode = node;
    while (currNode != null) {
        currNode.setHeight(max(currNode));
        int balance = getBalance(currNode);

        if (balance < -1 && getBalance(currNode.getLeft()) <= 0) {
            rightRotation(currNode);
        } else if (balance > 1 && getBalance(currNode.getRight()) >= 0) {
            leftRotation(currNode);
        } else if (balance < -1 && getBalance(currNode.getLeft()) > 0) {
            leftRotation(currNode.getLeft());
            rightRotation(currNode);
        } else if (balance > 1 && getBalance(currNode.getRight()) < 0) {
            rightRotation(currNode.getRight());
            leftRotation(currNode);
        }
        currNode = currNode.getParent();
    }
}

private void replaceChild(Node deleted, Node replace) {
```

```
    if (deleted.hasParent()) {
        if (deleted.equals(deleted.getParent().getLeft())) {
            deleted.getParent().setLeft(replace);
        } else {
            deleted.getParent().setRight(replace);
        }
    } else {
        root = replace;
    }
    replace.setParent(deleted.getParent());
}

private void deleteLeaf(Node leaf) {
    if (leaf.hasParent()) {
        if (leaf.equals(leaf.getParent().getLeft())) {
            leaf.getParent().setLeft(null);
        } else {
            leaf.getParent().setRight(null);
        }
    } else {
        root = null;
    }
}

private Node<T> findRightmost(Node<T> node) {
    while (node != null && node.hasRight()) {
        node = node.getRight();
    }
    return node;
}

public int getBalance(Node node) {
    if (node != null) {
        if (node.hasLeft() && node.hasRight()) {
            return node.getRight().getHeight() - node.getLeft().getHeight();
        } else if (!node.hasLeft() && node.hasRight()) {
            return node.getRight().getHeight() + 1;
        } else if (node.hasLeft()) {
            // right == null
            return -1 - node.getLeft().getHeight();
        }
    }
    return 0;
}


private void recalculateHeightsStartingFrom(Node<T> currentNode) {
    while (currentNode != null) {
        currentNode.setHeight(1 + Integer.max(
                currentNode.getLeft() == null ? -1 : currentNode.getLeft().getHeight(),
                currentNode.getRight() == null ? -1 : currentNode.getRight().getHeight()));
        currentNode = currentNode.getParent();
    }
}

public void print(EdxIO edxIO) {
    Node<T> node = root;
```

```
    Queue<Node<T>> queue = new LinkedList<>();
    if (node != null) {
        queue.add(node);
    }
    int c = 1;
    while (queue.size() != 0) {
        node = queue.poll();
        edxIO.print(node.getKey() + " ");
        if (node.hasLeft()) {
            queue.add(node.getLeft());
            c++;
            edxIO.print(c + " ");
        } else {
            edxIO.print("0 ");
        }
        if (node.hasRight()) {
            queue.add(node.getRight());
            c++;
            edxIO.print(c);
        } else {
            edxIO.print("0");
        }
        edxIO.println();
    }
}

private void leftRotation(Node x) {
    Node parent = x.getParent();
    Node T1 = x.getLeft();
    Node y = x.getRight();
    Node T2 = y.getLeft();
    Node T3 = y.getRight();

    y.setParent(parent);
    if (parent != null) {
        if (x.equals(parent.getLeft())) {
            parent.setLeft(y);
        } else {
            parent.setRight(y);
        }
    } else {
        root = y;
    }

    y.setLeft(x);
    x.setParent(y);

    x.setLeft(T1);
    if (T1 != null) {
        T1.setParent(x);
    }

    x.setRight(T2);
    if (T2 != null) {
        T2.setParent(x);
    }

    y.setRight(T3);
    if (T3 != null) {
```

```
            T3.setParent(y);
        }

        y.setHeight(max(y));
        x.setHeight(max(x));
    }

    private int max(Node node) {
        return 1 + Integer.max(
                node.getLeft() == null ? -1 : node.getLeft().getHeight(),
                node.getRight() == null ? -1 : node.getRight().getHeight()
        );
    }

    private void rightRotation(Node y) {
        Node parent = y.getParent();
        Node x = y.getLeft();
        Node T3 = y.getRight();
        Node T1 = x.getLeft();
        Node T2 = x.getRight();

        x.setParent(parent);
        if (parent != null) {
            if (y.equals(parent.getLeft())) {
                parent.setLeft(x);
            } else {
                parent.setRight(x);
            }
        } else {
            root = x;
        }

        x.setRight(y);
        y.setParent(x);

        x.setLeft(T1);
        if (T1 != null) {
            T1.setParent(x);
        }

        y.setLeft(T2);
        if (T2 != null) {
            T2.setParent(y);
        }

        y.setRight(T3);
        if (T3 != null) {
            T3.setParent(y);
        }

        y.setHeight(max(y));
        x.setHeight(max(x));
    }

    public Node<T> getRoot() {
        return root;
    }

    public void setRoot(Node<T> root) {
```

```
        this.root = root;
    }
}

static class Node<T> {
    private T key;
    private Node<T> parent;
    private Node<T> left;
    private Node<T> right;

    private int height;

    Node() {

    }

    Node(T key) {
        this.key = key;
    }

    Node(T key, Node<T> parent) {
        this.key = key;
        this.parent = parent;
    }

    public boolean hasParent() {
        return parent != null;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public T getKey() {
        return key;
    }

    public void setKey(T key) {
        this.key = key;
    }

    public Node<T> getParent() {
        return parent;
    }

    public void setParent(Node<T> parent) {
        this.parent = parent;
    }

    public Node<T> getLeft() {
        return left;
    }

    public void setLeft(Node<T> left) {
        this.left = left;
```

```
        }

        public Node<T> getRight() {
            return right;
        }

        public void setRight(Node<T> right) {
            this.right = right;
        }

        public boolean hasLeft() {
            return left != null;
        }

        public boolean hasRight() {
            return right != null;
        }

    }
}
```

Результаты

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1000	94277632	4011957	4011966
1	OK	125	21667840	20	24
2	OK	125	21692416	6	10
3	OK	140	21766144	14	18
4	OK	171	21725184	13	17
5	OK	109	21721088	21	25
6	OK	125	21716992	20	24
7	OK	125	21671936	20	24
8	OK	125	21684224	21	25
9	OK	125	21712896	20	24
10	OK	125	21680128	20	24
11	OK	125	21725184	28	32
12	OK	109	21680128	27	31
13	OK	125	21692416	27	31
14	OK	125	21696512	27	31
15	OK	109	21684224	35	39
16	OK	125	21753856	34	38
17	OK	125	21659648	34	38
18	OK	125	21688320	34	38
19	OK	140	21688320	34	38

20	OK	140	21663744	35	39
21	OK	140	21708800	34	38
22	OK	125	21688320	34	38
23	OK	109	21721088	34	38
24	OK	125	21647360	34	38
25	OK	109	21700608	35	39
26	OK	109	21688320	34	38
27	OK	109	21696512	34	38
28	OK	125	21716992	34	38
29	OK	125	21688320	34	38
30	OK	125	21712896	35	39
31	OK	140	21729280	34	38
32	OK	125	21716992	34	38
33	OK	125	21700608	34	38
34	OK	125	21676032	34	38
35	OK	140	21667840	42	46
36	OK	109	21680128	41	45
37	OK	109	21635072	41	45
38	OK	109	21663744	41	45
39	OK	125	21700608	41	45
40	OK	140	21753856	41	45
41	OK	109	21655552	42	46
42	OK	109	21659648	41	45
43	OK	125	21704704	41	45
44	OK	156	21692416	41	45
45	OK	109	21704704	41	45
46	OK	109	21712896	41	45
47	OK	109	21721088	42	46
48	OK	109	21663744	41	45
49	OK	125	21651456	41	45
50	OK	109	21688320	41	45
51	OK	109	21704704	41	45
52	OK	125	21766144	41	45
53	OK	125	21651456	42	46
54	OK	109	21721088	41	45

55	OK	125	21716992	41	45
56	OK	125	21651456	41	45
57	OK	156	21704704	41	45
58	OK	109	21737472	41	45
59	OK	109	21712896	42	46
60	OK	109	21659648	41	45
61	OK	140	21696512	41	45
62	OK	109	21667840	41	45
63	OK	109	21684224	41	45
64	OK	140	21766144	41	45
65	OK	125	21676032	42	46
66	OK	125	21671936	41	45
67	OK	125	21692416	41	45
68	OK	140	21704704	41	45
69	OK	125	21721088	41	45
70	OK	125	21741568	41	45
71	OK	140	21721088	50	54
72	OK	125	21680128	49	53
73	OK	125	21663744	49	53
74	OK	109	21749760	49	53
75	OK	125	21696512	49	53
76	OK	156	21762048	49	53
77	OK	125	21676032	50	54
78	OK	125	21684224	50	54
79	OK	125	21704704	49	53
80	OK	125	21704704	49	53
81	OK	109	21700608	49	53
82	OK	109	21684224	49	53
83	OK	125	21749760	49	53
84	OK	109	21688320	50	54
85	OK	140	21696512	50	54
86	OK	140	21725184	49	53
87	OK	109	21671936	49	53
88	OK	109	21680128	49	53
89	OK	140	21721088	49	53

90	OK	109	21688320	49	53
91	OK	109	21721088	50	54
92	OK	140	21688320	50	54
93	OK	156	21680128	49	53
94	OK	156	21721088	49	53
95	OK	109	21700608	49	53
96	OK	125	21741568	49	53
97	OK	109	21741568	49	53
98	OK	109	21725184	50	54
99	OK	109	21708800	58	62
100	OK	140	21700608	57	61
101	OK	171	21651456	57	61
102	OK	140	21704704	57	61
103	OK	125	21737472	57	61
104	OK	125	21692416	57	61
105	OK	125	21700608	58	62
106	OK	109	21712896	58	62
107	OK	125	21688320	58	62
108	OK	109	21745664	57	61
109	OK	109	21716992	57	61
110	OK	140	21708800	57	61
111	OK	109	21676032	57	61
112	OK	125	21725184	57	61
113	OK	109	21663744	58	62
114	OK	109	21712896	58	62
115	OK	140	21704704	58	62
116	OK	187	21725184	57	61
117	OK	109	21721088	57	61
118	OK	109	21684224	57	61
119	OK	140	21716992	57	61
120	OK	156	21708800	57	61
121	OK	125	21708800	58	62
122	OK	109	21692416	58	62
123	OK	140	21700608	58	62
124	OK	125	21647360	57	61

125	OK	125	21692416	57	61
126	OK	140	21749760	57	61
127	OK	109	21663744	57	61
128	OK	125	21745664	57	61
129	OK	109	21708800	58	62
130	OK	109	21700608	58	62
131	OK	125	21663744	58	62
132	OK	156	21725184	57	61
133	OK	109	21684224	57	61
134	OK	140	21725184	57	61
135	OK	109	21725184	57	61
136	OK	109	21684224	57	61
137	OK	125	21716992	58	62
138	OK	156	21725184	58	62
139	OK	125	21708800	58	62
140	OK	125	21655552	57	61
141	OK	125	21684224	57	61
142	OK	93	21680128	57	61
143	OK	109	21667840	57	61
144	OK	109	21733376	57	61
145	OK	109	21708800	58	62
146	OK	109	21688320	58	62
147	OK	171	21712896	58	62
148	OK	109	21635072	57	61
149	OK	109	21692416	57	61
150	OK	156	21766144	57	61
151	OK	156	21733376	57	61
152	OK	125	21688320	57	61
153	OK	109	21692416	58	62
154	OK	125	21667840	58	62
155	OK	140	21704704	58	62
156	OK	125	21721088	57	61
157	OK	140	21741568	57	61
158	OK	109	21688320	57	61
159	OK	109	21716992	57	61

160	OK	140	21651456	57	61
161	OK	125	21733376	58	62
162	OK	125	21716992	58	62
163	OK	125	21651456	58	62
164	OK	156	21708800	57	61
165	OK	125	21663744	57	61
166	OK	125	21696512	57	61
167	OK	109	21729280	57	61
168	OK	125	21696512	57	61
169	OK	125	21712896	58	62
170	OK	156	21725184	58	62
171	OK	125	21716992	58	62
172	OK	140	21684224	57	61
173	OK	109	21688320	57	61
174	OK	109	21774336	57	61
175	OK	140	21721088	57	61
176	OK	125	21688320	57	61
177	OK	109	21659648	58	62
178	OK	140	21680128	58	62
179	OK	125	21708800	58	62
180	OK	171	21704704	57	61
181	OK	125	21721088	57	61
182	OK	109	21708800	57	61
183	OK	125	21700608	57	61
184	OK	125	21663744	57	61
185	OK	140	21716992	58	62
186	OK	125	21729280	58	62
187	OK	156	21712896	58	62
188	OK	125	21712896	57	61
189	OK	140	21684224	57	61
190	OK	125	21684224	57	61
191	OK	109	21745664	57	61
192	OK	125	21704704	57	61
193	OK	125	21692416	58	62
194	OK	171	21671936	58	62

195	OK	125	21688320	58	62
196	OK	125	21655552	57	61
197	OK	125	21704704	57	61
198	OK	109	21790720	57	61
199	OK	93	21708800	57	61
200	OK	125	21725184	57	61
201	OK	125	21729280	58	62
202	OK	109	21688320	58	62
203	OK	140	21696512	58	62
204	OK	140	21749760	57	61
205	OK	125	21696512	57	61
206	OK	140	21671936	57	61
207	OK	156	21692416	57	61
208	OK	109	21680128	57	61
209	OK	125	21704704	58	62
210	OK	125	21729280	58	62
211	OK	109	21680128	58	62
212	OK	125	21700608	57	61
213	OK	109	21712896	57	61
214	OK	140	21630976	57	61
215	OK	156	21753856	57	61
216	OK	140	21721088	57	61
217	OK	140	21684224	58	62
218	OK	125	21725184	58	62
219	OK	140	21639168	58	62
220	OK	125	21716992	57	61
221	OK	109	21737472	57	61
222	OK	109	21721088	57	61
223	OK	125	21725184	57	61
224	OK	125	21667840	57	61
225	OK	125	21696512	58	62
226	OK	109	21704704	58	62
227	OK	109	21655552	58	62
228	OK	156	21700608	57	61
229	OK	125	21721088	57	61

230	OK	109	21684224	57	61
231	OK	140	21680128	57	61
232	OK	109	21712896	57	61
233	OK	125	21753856	58	62
234	OK	140	21737472	58	62
235	OK	109	21721088	240	245
236	OK	125	21688320	243	248
237	OK	125	21766144	1835	1841
238	OK	109	21749760	1815	1821
239	OK	125	21737472	1834	1840
240	OK	140	22261760	9951	9957
241	OK	140	22364160	9831	9837
242	OK	125	22237184	9854	9860
243	OK	156	23715840	38301	38308
244	OK	171	23781376	37664	37671
245	OK	171	23629824	39108	39115
246	OK	156	23703552	39190	39197
247	OK	203	31916032	183695	183703
248	OK	250	30330880	184258	184266
249	OK	265	30138368	185065	185073
250	OK	218	30474240	185428	185436
251	OK	250	31690752	185741	185749
252	OK	640	66633728	1900094	1900102
253	OK	640	66805760	1860225	1860233
254	OK	640	67497984	1899455	1899463
255	OK	656	67678208	1861088	1861096
256	OK	703	67526656	1942127	1942135
257	OK	687	67538944	1930200	1930208
258	OK	671	66785280	1861244	1861252
259	OK	937	78565376	3510448	3510457
260	OK	890	79699968	3650901	3650910
261	OK	921	78499840	3552374	3552383
262	OK	906	78114816	3435983	3435992
263	OK	937	78438400	3562689	3562698
264	OK	906	80306176	3521159	3521168

265	OK	875	78884864	3539149	3539158
266	OK	953	92127232	3985264	3985273
267	OK	1000	94277632	3866892	3866901
268	OK	921	90288128	3942753	3942762
269	OK	937	88158208	3824263	3824272
270	OK	937	85499904	4011957	4011966
271	OK	859	82968576	3955420	3955429
272	OK	953	82653184	3946583	3946592
273	OK	968	80154624	3891536	3891545

Задача №4

Условие

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.

Решение

```
package week7;

import mooc.EdxIO;

import java.util.LinkedList;
import java.util.Queue;

public class Week7_4 {
    private static EdxIO edxIO;

    public static void main(String[] args) {
        edxIO = EdxIO.create();
```

```
int n = edxIO.nextInt();

Node<Long>[] nodes = new Node[n];

AVLTree tree = new AVLTree(n);
for (int i = 0; i < n; i++) {
    long key = edxIO.nextLong();
    int left = edxIO.nextInt() - 1;
    int right = edxIO.nextInt() - 1;

    Node<Long> currentNode = nodes[i];
    // it can happen that it was created before if its' parent has already been parsed
    if (currentNode == null) {
        currentNode = new Node(key);
        nodes[i] = currentNode;
    } else {
        nodes[i].setKey((long) key);
    }

    if (left != -1) {
        if (nodes[left] == null) {
            Node<Long> node = new Node<Long>();
            nodes[left] = node;
            node.setParent(nodes[i]);
        }
        nodes[i].setLeft(nodes[left]);
    }

    if (right != -1) {
        if (nodes[right] == null) {
            Node<Long> node = new Node<Long>();
            nodes[right] = node;
            node.setParent(nodes[i]);
        }
        nodes[i].setRight(nodes[right]);
    }
    if (i == 0) {
        tree.setRoot(nodes[i]);
    }


    if (left == -1 && right == -1) {
        LinkedList<Node<Long>> curr = new LinkedList<>();
        while (currentNode != null) {
            curr.push(currentNode);
            currentNode = currentNode.getParent();
        }
        // setting height for all of the nodes on the path from
        // the first leaf
        while (curr.size() != 0) {
            currentNode = curr.pop();
            if (currentNode.getHeight() < curr.size())
                currentNode.setHeight(curr.size());
        }
    }
}

long key = edxIO.nextLong();
```

```
        tree.delete(key);

        edxIO.println(tree.getSize());
        tree.print(edxIO);
        edxIO.close();
    }

    static class AVLTree<T extends Comparable> {
        private long size;
        private Node<T> root;

        public AVLTree(int n) {
            size = n;
        }

        public long getSize() {
            return size;
        }

        public void setSize(long size) {
            this.size = size;
        }

        public boolean isEmpty() {
            return size == 0;
        }

        public Node<T> lookup(T key) {
            Node<T> currNode = root;
            while (currNode != null &&
                    currNode.getKey().compareTo(key) != 0
                    && (currNode.getKey().compareTo(key) < 0 && currNode.getRight() != null
                    || currNode.getKey().compareTo(key) > 0 && currNode.getLeft() != null)
            ) {
                if (currNode.getKey().compareTo(key) < 0) {
                    currNode = currNode.getRight();
                } else if (currNode.getKey().compareTo(key) > 0) {
                    currNode = currNode.getLeft();
                } else {
                    throw new IllegalStateException("[DEBUG] Just can't happen");
                }
            }
            return currNode;
        }

        private void balanceStartingFrom(Node<T> node) {
            Node<T> currNode = node;
            while (currNode != null) {
                currNode.setHeight(calculateHeight(currNode));
                int balance = getBalance(currNode);

                if (balance < -1 && getBalance(currNode.getLeft()) <= 0) {
                    rightRotation(currNode);
                } else if (balance > 1 && getBalance(currNode.getRight()) >= 0) {
                    leftRotation(currNode);
                } else if (balance < -1 && getBalance(currNode.getLeft()) > 0) {
                    leftRotation(currNode.getLeft());
                    rightRotation(currNode);
```

```
            } else if (balance > 1 && getBalance(currNode.getRight()) < 0) {
                rightRotation(currNode.getRight());
                leftRotation(currNode);
            }
            currNode = currNode.getParent();
        }
    }

    private void deleteLeaf(Node<T> leaf) {
        if (leaf.hasParent()) {
            if (leaf.equals(leaf.getParent().getLeft())) {
                leaf.getParent().setLeft(null);
            } else {
                leaf.getParent().setRight(null);
            }
        } else {
            root = null;
        }
    }

    private Node<T> findRightmost(Node<T> node) {
        while (node != null && node.hasRight()) {
            node = node.getRight();
        }
        return node;
    }

    public boolean delete(T key) {
        Node<T> node = lookup(key);
        if (node == null || node.getKey().compareTo(key) != 0) {
            return false;
        }

        if (!node.hasLeft() && !node.hasRight()) {
            deleteLeaf(node);
            if (node.hasParent()) {
                balanceStartingFrom(node.getParent());
            } else {
                // no children...
                // no parents...
                // root and leaf at the same time...
                // left dying alone in this severe world...
                // no replace...
            }
        } else if (!node.hasLeft()) {
            node.setKey(node.getRight().getKey());
            deleteLeaf(node.getRight());
            balanceStartingFrom(node);
        } else {
            Node<T> rightmostInLeftSubtree = findRightmost(node.getLeft());
            node.setKey(rightmostInLeftSubtree.getKey());
            if (rightmostInLeftSubtree.hasLeft()) {
                // then it's the left child of deleted node (there's no rightmost) and his left part should
be current node's left
                if (rightmostInLeftSubtree.equals(rightmostInLeftSubtree.getParent().getLeft())) {
                    rightmostInLeftSubtree.getParent().setLeft(rightmostInLeftSubtree.getLeft());
                } else {
                    rightmostInLeftSubtree.getParent().setRight(rightmostInLeftSubtree.getLeft());
```

```
            }
            rightmostInLeftSubtree.getLeft().setParent(rightmostInLeftSubtree.getParent());
        } else {
            deleteLeaf(rightmostInLeftSubtree);
        }
        balanceStartingFrom(rightmostInLeftSubtree.getParent());
    }

    size--;
    return true;
}

public int getBalance(Node node) {
    if (node != null) {
        if (node.hasLeft() && node.hasRight()) {
            return node.getRight().getHeight() - node.getLeft().getHeight();
        } else if (!node.hasLeft() && node.hasRight()) {
            return node.getRight().getHeight() + 1;
        } else if (node.hasLeft()) {
            // right == null
            return -1 - node.getLeft().getHeight();
        }
    }
    return 0;
}

public void print(EdxIO edxIO) {
    Node<T> node = root;
    Queue<Node<T>> queue = new LinkedList<>();
    if (node != null) {
        queue.add(node);
    }
    int c = 1;
    while (queue.size() != 0) {
        node = queue.poll();
        edxIO.print(node.getKey() + " ");
        if (node.hasLeft()) {
            queue.add(node.getLeft());
            c++;
            edxIO.print(c + " ");
        } else {
            edxIO.print("0 ");
        }
        if (node.hasRight()) {
            queue.add(node.getRight());
            c++;
            edxIO.print(c);
        } else {
            edxIO.print("0");
        }
        edxIO.println();
    }
}

private void leftRotation(Node<T> x) {
    Node<T> parent = x.getParent();
    Node<T> T1 = x.getLeft();
    Node<T> y = x.getRight();
```

```
    Node<T> T2 = y.getLeft();
    Node<T> T3 = y.getRight();

    y.setParent(parent);
    if (parent != null) {
        if (x.equals(parent.getLeft())) {
            parent.setLeft(y);
        } else {
            parent.setRight(y);
        }
    } else {
        root = y;
    }

    y.setLeft(x);
    x.setParent(y);

    x.setLeft(T1);
    if (T1 != null) {
        T1.setParent(x);
    }

    x.setRight(T2);
    if (T2 != null) {
        T2.setParent(x);
    }

    y.setRight(T3);
    if (T3 != null) {
        T3.setParent(y);
    }

    y.setHeight(calculateHeight(y));
    x.setHeight(calculateHeight(x));
}

private int calculateHeight(Node<T> node) {
    return 1 + Integer.max(
            node.getLeft() == null ? -1 : node.getLeft().getHeight(),
            node.getRight() == null ? -1 : node.getRight().getHeight()
    );
}

private void rightRotation(Node<T> y) {
    Node<T> parent = y.getParent();
    Node<T> x = y.getLeft();
    Node<T> T3 = y.getRight();
    Node<T> T1 = x.getLeft();
    Node<T> T2 = x.getRight();

    x.setParent(parent);
    if (parent != null) {
        if (y.equals(parent.getLeft())) {
            parent.setLeft(x);
        } else {
            parent.setRight(x);
        }
    } else {
        root = x;
```

```
        }

        x.setRight(y);
        y.setParent(x);

        x.setLeft(T1);
        if (T1 != null) {
            T1.setParent(x);
        }

        y.setLeft(T2);
        if (T2 != null) {
            T2.setParent(y);
        }

        y.setRight(T3);
        if (T3 != null) {
            T3.setParent(y);
        }

        y.setHeight(calculateHeight(y));
        x.setHeight(calculateHeight(x));
    }

    public Node<T> getRoot() {
        return root;
    }

    public void setRoot(Node<T> root) {
        this.root = root;
    }
}

static class Node<T> {
    private T key;
    private Node<T> parent;
    private Node<T> left;
    private Node<T> right;

    private int height;

    Node() {

    }

    Node(T key) {
        this.key = key;
    }

    Node(T key, Node<T> parent) {
        this.key = key;
        this.parent = parent;
    }

    public boolean hasParent() {
        return parent != null;
    }

    public int getHeight() {
```

```
            return height;
        }

        public void setHeight(int height) {
            this.height = height;
        }

        public T getKey() {
            return key;
        }

        public void setKey(T key) {
            this.key = key;
        }

        public Node<T> getParent() {
            return parent;
        }

        public void setParent(Node<T> parent) {
            this.parent = parent;
        }

        public Node<T> getLeft() {
            return left;
        }

        public void setLeft(Node<T> left) {
            this.left = left;
        }

        public Node<T> getRight() {
            return right;
        }

        public void setRight(Node<T> right) {
            this.right = right;
        }

        public boolean hasLeft() {
            return left != null;
        }

        public boolean hasRight() {
            return right != null;
        }
    }
}
```

Результаты

№ теста	Результат	Время, мс	Память	Размер входного файла	Размер выходного файла
Max		1109	92463104	4077288	4077255
1	OK	140	21684224	27	17

2	OK	125	21700608	13	3
3	OK	125	21712896	20	10
4	OK	109	21659648	20	10
5	OK	109	21704704	20	10
6	OK	140	21671936	20	10
7	OK	109	21704704	27	17
8	OK	140	21716992	27	17
9	OK	125	21729280	27	17
10	OK	156	21700608	34	24
11	OK	125	21663744	34	24
12	OK	140	21729280	34	24
13	OK	125	21696512	34	24
14	OK	109	21643264	34	24
15	OK	125	21680128	34	24
16	OK	125	21667840	34	24
17	OK	125	21651456	34	24
18	OK	125	21725184	34	24
19	OK	109	21741568	34	24
20	OK	156	21684224	34	24
21	OK	140	21651456	34	24
22	OK	156	21692416	34	24
23	OK	140	21659648	34	24
24	OK	109	21733376	34	24
25	OK	125	21708800	34	24
26	OK	140	21651456	41	31
27	OK	140	21692416	41	31
28	OK	125	21712896	41	31
29	OK	109	21737472	41	31
30	OK	109	21733376	41	31
31	OK	125	21733376	41	31
32	OK	125	21680128	41	31
33	OK	156	21725184	41	31
34	OK	109	21684224	41	31
35	OK	125	21680128	41	31
36	OK	125	21659648	41	31

37	OK	125	21716992	41	31
38	OK	125	21635072	41	31
39	OK	125	21708800	41	31
40	OK	125	21700608	41	31
41	OK	109	21725184	41	31
42	OK	109	21684224	41	31
43	OK	109	21696512	41	31
44	OK	171	21708800	41	31
45	OK	140	21712896	41	31
46	OK	109	21712896	41	31
47	OK	140	21725184	41	31
48	OK	109	21671936	41	31
49	OK	125	21684224	41	31
50	OK	125	21696512	41	31
51	OK	156	21708800	41	31
52	OK	125	21684224	41	31
53	OK	125	21733376	41	31
54	OK	109	21712896	41	31
55	OK	156	21721088	41	31
56	OK	140	21696512	48	38
57	OK	156	21688320	48	38
58	OK	109	21688320	48	38
59	OK	125	21680128	48	38
60	OK	125	21696512	48	38
61	OK	125	21630976	48	38
62	OK	140	21688320	48	38
63	OK	140	21663744	48	38
64	OK	140	21643264	48	38
65	OK	109	21745664	48	38
66	OK	125	21696512	48	38
67	OK	109	21721088	48	38
68	OK	125	21676032	48	38
69	OK	125	21696512	48	38
70	OK	109	21680128	48	38
71	OK	109	21700608	48	38

72	OK	125	21725184	48	38
73	OK	140	21700608	48	38
74	OK	125	21667840	48	38
75	OK	125	21635072	48	38
76	OK	125	21708800	48	38
77	OK	140	21745664	48	38
78	OK	140	21704704	48	38
79	OK	125	21688320	48	38
80	OK	125	21733376	55	45
81	OK	125	21676032	55	45
82	OK	140	21712896	55	45
83	OK	140	21704704	55	45
84	OK	125	21684224	55	45
85	OK	125	21684224	55	45
86	OK	171	21671936	55	45
87	OK	125	21663744	55	45
88	OK	109	21704704	55	45
89	OK	125	21725184	55	45
90	OK	125	21716992	55	45
91	OK	140	21712896	55	45
92	OK	140	21692416	55	45
93	OK	140	21671936	55	45
94	OK	109	21700608	55	45
95	OK	109	21708800	55	45
96	OK	125	21700608	55	45
97	OK	109	21704704	55	45
98	OK	109	21684224	55	45
99	OK	156	21651456	55	45
100	OK	125	21663744	55	45
101	OK	109	21745664	55	45
102	OK	109	21684224	55	45
103	OK	125	21708800	55	45
104	OK	156	21659648	55	45
105	OK	171	21684224	55	45
106	OK	109	21692416	55	45

107	OK	109	21708800	55	45
108	OK	109	21704704	55	45
109	OK	109	21659648	55	45
110	OK	125	21667840	55	45
111	OK	140	21700608	55	45
112	OK	125	21671936	55	45
113	OK	109	21749760	55	45
114	OK	109	21733376	55	45
115	OK	125	21692416	55	45
116	OK	140	21684224	55	45
117	OK	125	21680128	55	45
118	OK	125	21680128	55	45
119	OK	125	21712896	55	45
120	OK	140	21667840	55	45
121	OK	109	21655552	55	45
122	OK	140	21696512	55	45
123	OK	125	21655552	55	45
124	OK	125	21680128	55	45
125	OK	125	21712896	55	45
126	OK	125	21684224	55	45
127	OK	125	21712896	55	45
128	OK	109	21712896	55	45
129	OK	140	21725184	55	45
130	OK	140	21696512	55	45
131	OK	125	21741568	55	45
132	OK	125	21721088	55	45
133	OK	109	21622784	55	45
134	OK	140	21684224	55	45
135	OK	109	21667840	55	45
136	OK	140	21708800	55	45
137	OK	109	21712896	55	45
138	OK	125	21676032	55	45
139	OK	109	21696512	55	45
140	OK	109	21696512	55	45
141	OK	109	21684224	55	45

142	OK	156	21684224	55	45
143	OK	125	21737472	55	45
144	OK	125	21671936	55	45
145	OK	125	21643264	55	45
146	OK	109	21696512	55	45
147	OK	109	21626880	55	45
148	OK	125	21667840	55	45
149	OK	125	21745664	55	45
150	OK	109	21725184	55	45
151	OK	109	21708800	55	45
152	OK	109	21729280	55	45
153	OK	109	21737472	55	45
154	OK	140	21700608	55	45
155	OK	140	21712896	55	45
156	OK	140	21716992	55	45
157	OK	109	21667840	55	45
158	OK	125	21700608	55	45
159	OK	125	21651456	55	45
160	OK	109	21725184	55	45
161	OK	125	21700608	55	45
162	OK	125	21716992	55	45
163	OK	125	21655552	55	45
164	OK	93	21708800	55	45
165	OK	125	21704704	55	45
166	OK	125	21704704	55	45
167	OK	109	21704704	55	45
168	OK	140	21688320	55	45
169	OK	125	21639168	55	45
170	OK	125	21667840	55	45
171	OK	125	21622784	55	45
172	OK	109	21696512	55	45
173	OK	109	21770240	55	45
174	OK	140	21651456	55	45
175	OK	125	21745664	55	45
176	OK	109	21704704	55	45

177	OK	125	21692416	55	45
178	OK	125	21684224	55	45
179	OK	109	21716992	55	45
180	OK	187	21692416	55	45
181	OK	171	21647360	55	45
182	OK	125	21692416	55	45
183	OK	140	21655552	55	45
184	OK	109	21696512	55	45
185	OK	93	21708800	55	45
186	OK	125	21692416	55	45
187	OK	140	21716992	55	45
188	OK	156	21692416	55	45
189	OK	171	21680128	55	45
190	OK	140	21684224	55	45
191	OK	125	21688320	55	45
192	OK	140	21684224	55	45
193	OK	140	21659648	55	45
194	OK	140	21700608	55	45
195	OK	140	21667840	55	45
196	OK	125	21696512	55	45
197	OK	109	21712896	55	45
198	OK	140	21704704	55	45
199	OK	109	21696512	239	210
200	OK	125	21684224	235	208
201	OK	140	21770240	1797	1769
202	OK	140	21753856	1809	1781
203	OK	125	21766144	1831	1803
204	OK	125	22233088	9625	9597
205	OK	140	22233088	10026	9996
206	OK	125	22224896	9672	9642
207	OK	156	23629824	39459	39428
208	OK	140	23871488	39672	39641
209	OK	156	23646208	38780	38749
210	OK	187	23646208	38392	38361
211	OK	218	31703040	179425	179394

212	OK	203	31891456	182878	182845
213	OK	218	31981568	185986	185953
214	OK	296	31666176	186275	186244
215	OK	281	30085120	179082	179051
216	OK	640	66945024	1912349	1912319
217	OK	625	67108864	1864033	1864003
218	OK	656	67432448	1913940	1913908
219	OK	718	67440640	1879988	1879958
220	OK	656	67858432	1887512	1887482
221	OK	656	67084288	1932558	1932526
222	OK	640	67788800	1875036	1875006
223	OK	937	78630912	3597394	3597361
224	OK	890	78323712	3569973	3569940
225	OK	890	79466496	3512224	3512193
226	OK	953	79130624	3624329	3624296
227	OK	937	78045184	3523352	3523321
228	OK	953	78815232	3638077	3638044
229	OK	953	79159296	3512844	3512813
230	OK	968	91787264	4001320	4001287
231	OK	1109	92463104	3954282	3954249
232	OK	890	87511040	3907814	3907783
233	OK	968	89096192	3983014	3982983
234	OK	906	88915968	4029895	4029862
235	OK	921	86069248	4046188	4046157
236	OK	921	84795392	4077288	4077255
237	OK	890	80924672	3902092	3902061

Задача №5

Условие

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.

Решение

```
package week7;

import mooc.EdxIO;

import java.util.LinkedList;
import java.util.Queue;

public class Week7_5 {
    private static EdxIO edxIO;

    public static void main(String[] args) {
        edxIO = EdxIO.create();

        int n = edxIO.nextInt();

        AVLTree<Integer> tree = new AVLTree<>(0);
        for (int i = 0; i < n; i++) {
            switch (edxIO.nextChar()) {
                case 'A':
                    tree.insert(edxIO.nextInt());
                    edxIO.println(tree.getBalance(tree.getRoot()));
                    break;
                case 'C':
                    edxIO.println(tree.contains(edxIO.nextInt()) ? 'Y' : 'N');
                    break;
```

```
            case 'D':
                tree.delete(edxIO.nextInt());
                edxIO.println(tree.getBalance(tree.getRoot()));
                break;
            default:
                throw new IllegalStateException();
        }
    }
    edxIO.close();
}

static class AVLTree<T extends Comparable> {
    private long size;
    private Node<T> root;

    public AVLTree(int n) {
        size = n;
    }

    public long getSize() {
        return size;
    }

    public void setSize(long size) {
        this.size = size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public Node<T> lookup(T key) {
        Node<T> currNode = root;
        while (currNode != null &&
                currNode.getKey().compareTo(key) != 0
                && (currNode.getKey().compareTo(key) < 0 && currNode.getRight() != null
                || currNode.getKey().compareTo(key) > 0 && currNode.getLeft() != null)
        ) {
            if (currNode.getKey().compareTo(key) < 0) {
                currNode = currNode.getRight();
            } else if (currNode.getKey().compareTo(key) > 0) {
                currNode = currNode.getLeft();
            } else {
                throw new IllegalStateException("[DEBUG] Just can't happen");
            }
        }
        return currNode;
    }

    public boolean contains(T key) {
        Node<T> node = lookup(key);
        return node != null && key.compareTo(node.getKey()) == 0;
    }

    public void insert(T key) {
        if (isEmpty()) {
            root = new Node<>(key);
        } else {
```

```
            Node<T> lookupNode = lookup(key);
            Node<T> newNode = new Node<>(key, lookupNode);
            if (lookupNode.getKey().compareTo(key) < 0) {
                lookupNode.setRight(newNode);
            } else if (lookupNode.getKey().compareTo(key) > 0) {
                lookupNode.setLeft(newNode);
            } else {
                return;
            }

            balanceStartingFrom(lookupNode);
        }
        size++;
    }

    private void balanceStartingFrom(Node<T> node) {
        Node<T> currNode = node;
        while (currNode != null) {
            currNode.setHeight(calculateHeight(currNode));
            int balance = getBalance(currNode);

            if (balance < -1 && getBalance(currNode.getLeft()) <= 0) {
                rightRotation(currNode);
            } else if (balance > 1 && getBalance(currNode.getRight()) >= 0) {
                leftRotation(currNode);
            } else if (balance < -1 && getBalance(currNode.getLeft()) > 0) {
                leftRotation(currNode.getLeft());
                rightRotation(currNode);
            } else if (balance > 1 && getBalance(currNode.getRight()) < 0) {
                rightRotation(currNode.getRight());
                leftRotation(currNode);
            }
            currNode = currNode.getParent();
        }
    }

    private void deleteLeaf(Node<T> leaf) {
        if (leaf.hasParent()) {
            if (leaf.equals(leaf.getParent().getLeft())) {
                leaf.getParent().setLeft(null);
            } else {
                leaf.getParent().setRight(null);
            }
        } else {
            root = null;
        }
    }

    private Node<T> findRightmost(Node<T> node) {
        while (node != null && node.hasRight()) {
            node = node.getRight();
        }
        return node;
    }

    public boolean delete(T key) {
        Node<T> node = lookup(key);
        if (node == null || node.getKey().compareTo(key) != 0) {
```

```
            return false;
        }

        if (!node.hasLeft() && !node.hasRight()) {
            deleteLeaf(node);
            if (node.hasParent()) {
                balanceStartingFrom(node.getParent());
            } else {
                // no children...
                // no parents...
                // root and leaf at the same time...
                // left dying alone in this severe world...
                // no replace...
            }
        } else if (!node.hasLeft()) {
            node.setKey(node.getRight().getKey());
            deleteLeaf(node.getRight());
            balanceStartingFrom(node);
        } else {
            Node<T> rightmostInLeftSubtree = findRightmost(node.getLeft());
            node.setKey(rightmostInLeftSubtree.getKey());
            if (rightmostInLeftSubtree.hasLeft()) {
                // then it's the left child of deleted node (there's no rightmost) and his left part should
be current node's left
                if (rightmostInLeftSubtree.equals(rightmostInLeftSubtree.getParent().getLeft())) {
                    rightmostInLeftSubtree.getParent().setLeft(rightmostInLeftSubtree.getLeft());
                } else {
                    rightmostInLeftSubtree.getParent().setRight(rightmostInLeftSubtree.getLeft());
                }
                rightmostInLeftSubtree.getLeft().setParent(rightmostInLeftSubtree.getParent());
            } else {
                deleteLeaf(rightmostInLeftSubtree);
            }
            balanceStartingFrom(rightmostInLeftSubtree.getParent());
        }

        size--;
        return true;
    }

    public int getBalance(Node node) {
        if (node != null) {
            if (node.hasLeft() && node.hasRight()) {
                return node.getRight().getHeight() - node.getLeft().getHeight();
            } else if (!node.hasLeft() && node.hasRight()) {
                return node.getRight().getHeight() + 1;
            } else if (node.hasLeft()) {
                // right == null
                return -1 - node.getLeft().getHeight();
            }
        }
        return 0;
    }

    public void print(EdxIO edxIO) {
        Node<T> node = root;
        Queue<Node<T>> queue = new LinkedList<>();
        if (node != null) {
```

```
            queue.add(node);
        }
        int c = 1;
        while (queue.size() != 0) {
            node = queue.poll();
            edxIO.print(node.getKey() + " ");
            if (node.hasLeft()) {
                queue.add(node.getLeft());
                c++;
                edxIO.print(c + " ");
            } else {
                edxIO.print("0 ");
            }
            if (node.hasRight()) {
                queue.add(node.getRight());
                c++;
                edxIO.print(c);
            } else {
                edxIO.print("0");
            }
            edxIO.println();
        }
    }

    private void leftRotation(Node<T> x) {
        Node<T> parent = x.getParent();
        Node<T> T1 = x.getLeft();
        Node<T> y = x.getRight();
        Node<T> T2 = y.getLeft();
        Node<T> T3 = y.getRight();

        y.setParent(parent);
        if (parent != null) {
            if (x.equals(parent.getLeft())) {
                parent.setLeft(y);
            } else {
                parent.setRight(y);
            }
        } else {
            root = y;
        }

        y.setLeft(x);
        x.setParent(y);

        x.setLeft(T1);
        if (T1 != null) {
            T1.setParent(x);
        }

        x.setRight(T2);
        if (T2 != null) {
            T2.setParent(x);
        }

        y.setRight(T3);
        if (T3 != null) {
            T3.setParent(y);
        }
```

```
    y.setHeight(calculateHeight(y));
    x.setHeight(calculateHeight(x));
}

private int calculateHeight(Node<T> node) {
    return 1 + Integer.max(
            node.getLeft() == null ? -1 : node.getLeft().getHeight(),
            node.getRight() == null ? -1 : node.getRight().getHeight()
    );
}

private void rightRotation(Node<T> y) {
    Node<T> parent = y.getParent();
    Node<T> x = y.getLeft();
    Node<T> T3 = y.getRight();
    Node<T> T1 = x.getLeft();
    Node<T> T2 = x.getRight();

    x.setParent(parent);
    if (parent != null) {
        if (y.equals(parent.getLeft())) {
            parent.setLeft(x);
        } else {
            parent.setRight(x);
        }
    } else {
        root = x;
    }

    x.setRight(y);
    y.setParent(x);

    x.setLeft(T1);
    if (T1 != null) {
        T1.setParent(x);
    }

    y.setLeft(T2);
    if (T2 != null) {
        T2.setParent(y);
    }

    y.setRight(T3);
    if (T3 != null) {
        T3.setParent(y);
    }

    y.setHeight(calculateHeight(y));
    x.setHeight(calculateHeight(x));
}

public Node<T> getRoot() {
    return root;
}

public void setRoot(Node<T> root) {
    this.root = root;
}
```

```
}

static class Node<T> {
    private T key;
    private Node<T> parent;
    private Node<T> left;
    private Node<T> right;

    private int height;

    Node(T key) {
        this.key = key;
    }

    Node(T key, Node<T> parent) {
        this.key = key;
        this.parent = parent;
    }

    public boolean hasParent() {
        return parent != null;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public T getKey() {
        return key;
    }

    public void setKey(T key) {
        this.key = key;
    }

    public Node<T> getParent() {
        return parent;
    }

    public void setParent(Node<T> parent) {
        this.parent = parent;
    }

    public Node<T> getLeft() {
        return left;
    }

    public void setLeft(Node<T> left) {
        this.left = left;
    }

    public Node<T> getRight() {
        return right;
    }
```

```
        public void setRight(Node<T> right) {
            this.right = right;
        }

        public boolean hasLeft() {
            return left != null;
        }

        public boolean hasRight() {
            return right != null;
        }
    }
}
```

Результаты

№ теста	Результат	Время, мс	Память	Размер входного файла	Размер выходного файла
Max		437	42491904	2678110	731071
1	OK	109	21635072	33	19
2	OK	125	21602304	114	66
3	OK	125	21630976	154	90
4	OK	171	21606400	154	91
5	OK	125	21630976	154	90
6	OK	109	21643264	154	95
7	OK	125	21614592	154	91
8	OK	140	21610496	154	94
9	OK	125	21626880	154	95
10	OK	125	21647360	154	90
11	OK	109	21635072	154	90
12	OK	171	21635072	154	90
13	OK	109	21667840	154	95
14	OK	109	21610496	154	97
15	OK	187	21622784	154	94
16	OK	125	21606400	154	93
17	OK	140	21655552	154	90
18	OK	125	21635072	154	90
19	OK	125	21647360	154	98
20	OK	109	21581824	154	93
21	OK	156	21671936	154	92
22	OK	156	21651456	154	98
23	OK	250	29065216	1000008	616458

24	OK	281	34992128	1000008	622272
25	OK	265	32759808	1000008	625335
26	OK	281	33738752	1000008	628546
27	OK	328	32530432	1000008	631472
28	OK	343	32542720	1000008	632217
29	OK	312	32620544	1000008	631772
30	OK	265	30765056	1000008	631071
31	OK	281	30621696	1000008	630132
32	OK	328	31825920	1017957	630451
33	OK	281	29741056	1000008	616595
34	OK	265	31174656	1000008	622199
35	OK	265	33292288	1000008	625057
36	OK	296	32972800	1000008	628040
37	OK	281	34496512	1000008	631495
38	OK	343	34156544	1000008	632086
39	OK	265	32518144	1000008	631753
40	OK	312	32436224	1000008	630849
41	OK	312	33611776	1000008	630110
42	OK	312	33370112	1018151	630800
43	OK	125	21606400	756	369
44	OK	125	21659648	758	432
45	OK	125	21712896	1659	408
46	OK	125	21688320	723	383
47	OK	125	21680128	723	385
48	OK	109	21671936	723	415
49	OK	125	21635072	723	415
50	OK	156	21700608	1668	377
51	OK	125	21684224	1660	396
52	OK	125	22290432	5348	2337
53	OK	109	22290432	5350	2848
54	OK	156	22310912	10439	2648
55	OK	140	22233088	5238	2343
56	OK	125	22167552	5238	2465
57	OK	156	22392832	5238	2719
58	OK	109	22319104	5238	2719

59	OK	171	22396928	10450	2421
60	OK	125	22089728	10439	2405
61	OK	171	23486464	32784	12708
62	OK	171	23609344	32787	14896
63	OK	156	23736320	56716	12715
64	OK	187	24055808	31674	12778
65	OK	156	24350720	31674	13220
66	OK	203	23949312	31674	14383
67	OK	171	24051712	31674	14825
68	OK	156	24264704	56748	13671
69	OK	171	23789568	56716	13193
70	OK	156	25567232	162462	57855
71	OK	171	25513984	162466	68948
72	OK	203	26787840	258205	71756
73	OK	218	24936448	152067	59306
74	OK	234	25071616	152067	59903
75	OK	187	25804800	152067	66900
76	OK	203	26042368	152067	67497
77	OK	265	26394624	258312	70001
78	OK	187	25268224	258332	58111
79	OK	250	29888512	811002	274035
80	OK	250	29872128	811006	332612
81	OK	281	31322112	1222794	299942
82	OK	328	29712384	799892	286940
83	OK	296	30445568	799892	282227
84	OK	296	31326208	799892	324420
85	OK	281	28839936	799892	319707
86	OK	359	33210368	1222871	284516
87	OK	296	29093888	1223246	288111
88	OK	312	38060032	1888898	600000
89	OK	343	38227968	1888903	731071
90	OK	421	37904384	2677526	600067
91	OK	312	38842368	1777788	601696
92	OK	390	42491904	1777788	632768
93	OK	390	41652224	1777788	698302

94	OK	328	37605376	1777788	698303
95	OK	437	41066496	2678110	611713
96	OK	406	36724736	2677266	600286