Problem A

Data structure

Implement linked binary search tree with:

- **`void insert(int key) {}`**
- **`boolean delete(int key) {}`**
- **`YourNodeClass find(int key) {}`**

operations. Assume keys are integer numbers. For **delete()** operation use **predecessor**s.

Tests

Input file **bst.in** contains 3 lines of numbers. **First** line is integer data for **insertion**, **second** line is data for **deletion**, and **third** line is data for **testing**:

1) Read the values from the first line as integers and insert them one-by-one into your linked binary search tree.
2) Delete the values of the second line from your tree one-by-one.
3) Read values from the third row, find nodes and for each found node print right child's value or null (if not present):
   a. `Node q = find(value);`
   b. `Node right = q == null ? null : q.getRight();`
   c. `if (right == null) print "null"`
      `else print right.getKey()`

Separate your output values with spaces and write them to **bst.out** file.

bst.in	**bst.out**
7 5 19 4 6 15 22 13 16 45 17 77 76 17 22 5 7 16 19	6 19 null 45
7 5 19 4 6 15 22 13 16 45 17 77 19 7 77 76 17 6 5	null null 22 17 null

Problem B

Data structure

Implement AVL tree with:

- **`void insert(int key) {}`**
- **`boolean delete(int key) {}`**
- **`YourNodeClass find(int key) {}`**

operations. Assume keys are integer numbers. For **delete()** operation use **predecessor**s.

Tests

Input file **avl.in** contains 3 lines of numbers. **First** line is integer data for **insertion**, **second** line is data for **deletion**, and **third** line is data for **testing**:

1) Read the values from the first line as integers and insert them one-by-one into your AVL tree.
2) Delete the values of the second line from your tree one-by-one.
3) Read values from the third row, find nodes and for each found node print right child's value or null (if not present):
   a. `Node q = find(value);`
   b. `Node right = q == null ? null : q.getRight();`
   c. `if (right == null) print "null"`
      `else print right.getKey()`

Separate your output values with spaces and write them to **avl.out**.

avl.in	**avl.out**
1 2 3 1 2 3	null 3 null
10 15 5 11 16 17 10 15	11 16
10 15 11 16 14 13 10 15 11 16 14 13	null 16 13 null 15 null
1 2 3 4 1 2 3 4	null 4 null
1 2 3 4 5 6 7 4 3 2 6	6 null 7

Problem C

Data structure

Implements Red-Black tree with:

- **`void insert(int key) {}`**
- **`YourNodeClass find(int key) {}`**

operations.

Tests

Input file **rbt.in** contains 2 lines of numbers. **First** line is integer data for **insertion** and **second** line is data for **testing**:

4) Read the values from the first line as integers and insert them one-by-one into your AVL tree.
5) Read values from the second row, find nodes and for each found node print right child's value or null (if not present):
   a. `Node q = find(value);`
   b. `Node right = q == null ? null : q.getRight();`
   c. `if (right == null) print "null"`
      `else print right.getKey()`

Separate your output values with spaces and write them to **rbt.out**.

rbt.in	**rbt.out**
10 5 15 20 10	15
10 5 15 20 25 10	20
10 5 15 20 25 16 15	16
10 5 15 20 25 16 17 15 16	null 17