

REPORT

NAME: Ahmed ali elsayed saber khalil

ID: 7

NAME: Islam mostafa Abdelaziz gaber

ID: 13

IMPORTANT DECISIONS:

- We implemented the red black tree as a binary search tree with it's attributes such as: key , left , right , parent , value.
- We added one attribute per node which is : color ; which is either Red or Black.
- As we insert or delete a node in the tree we fixup it by rotation methods.
- We implemented an additional class "RBtreePrinter" has a method to print the red black tree.

- We applied all the properties of the Red-Black trees such as:
 - Every node is either red or black.
 - The root is black.
 - Every leaf is black.
 - If a node is red, then both its children are black.
 - For each node, all paths from the node to descendant leaves contain the same number of black nodes.

- We implemented the TreeMap as it is based on Red-Black tree.

The time analysis of implemented methods:

- The insertion has $O(\log n)$ time.
- Also , the deletion has $O(\log n)$ time.
- We passed all the tests of this lab and this is the time of them.



✓ Tests passed: 69 of 69 tests - 24s 657ms

Sample Runs:

➤ Example 1:

Here we inserted some nodes in the red black tree , then we print the tree.

The screenshot shows an IDE with several tabs: Main.java, RedBlackTree.java, RBTreePrinter.java, INode.java, and IRedBlackTree.java. The Main.java tab is active, displaying the following code:

```
public static void main(String[] args) {  
    RedBlackTree<Integer,String> RB = new RedBlackTree();  
    RB.insert(1,"1");  
    RB.insert(0,"0");  
    RB.insert(3,"3");  
    RB.insert(6,"6");  
    RB.insert(4,"4");  
    RB.insert(13,"13");  
    RB.insert(16,"16");  
    RB.insert(10,"10");  
    RBTreePrinter.print(RB.getRoot());  
}
```

Below the code editor, the output window shows the command executed: "C:\Program Files\Java\jdk-11.0.2\bin\java.exe" "-javaagent:C:\Program File:". The output is a diagram of a Red-Black Tree:

```
graph TD  
    4["4 (B)"] --- 1["1 (R)"]  
    4 --- 13["13 (R)"]  
    1 --- 0["0 (B)"]  
    1 --- 3["3 (B)"]  
    13 --- 6["6 (B)"]  
    13 --- 16["16 (B)"]  
    6 --- 10["10 (R)"]
```

The diagram illustrates a Red-Black Tree structure. The root node is 4 (B). It has two children: 1 (R) and 13 (R). Node 1 (R) has two children: 0 (B) and 3 (B). Node 13 (R) has two children: 6 (B) and 16 (B). Node 6 (B) has one child: 10 (R).

➤ Example 2:

Here we inserted some nodes ,then delete a node with the key “6” ,then print the tree.

The screenshot shows a Java IDE with four tabs: Main.java, RedBlackTree.java, RBTreePrinter.java, and INode.java. The Main.java tab is active, displaying the following code:

```
9
10 public static void main(String[] args) {
11
12
13     RedBlackTree<Integer,String> RB = new RedBlackTree();
14     RB.insert(1,"1");
15     RB.insert(0,"0");
16     RB.insert(3,"3");
17     RB.insert(6,"6");
18     RB.insert(4,"4");
19     RB.insert(13,"13");
20     RB.insert(16,"16");
21     RB.insert(10,"10");
22     RB.delete( key: 6);
23     RBTreePrinter.print(RB.getRoot());
```

Below the code editor, the output window shows the command prompt path and a diagram of the Red-Black Tree structure:

```
"C:\Program Files\Java\jdk-11.0.2\bin\java
```

The tree diagram is as follows:

```
graph TD
    4["4 (B)"] --- 1["1 (R)"]
    4 --- 13["13 (R)"]
    1 --- 0["0 (B)"]
    1 --- 3["3 (B)"]
    13 --- 10["10 (B)"]
    13 --- 16["16 (B)"]
```

The diagram illustrates a Red-Black Tree with root node 4 (Black). Node 4 has two children: 1 (Red) and 13 (Red). Node 1 has two children: 0 (Black) and 3 (Black). Node 13 has two children: 10 (Black) and 16 (Black). The colors in parentheses indicate the color of each node.

➤ Example 3:

Here we inserted some nodes , then delete an node with key “6”, then search the node with key “13” which return it’s value = 13 , then check if the tree contains the key “1” which is true , then check if the tree contains the key “6” that we deleted before which is false , then we check if the tree is empty which is false , then we print it.

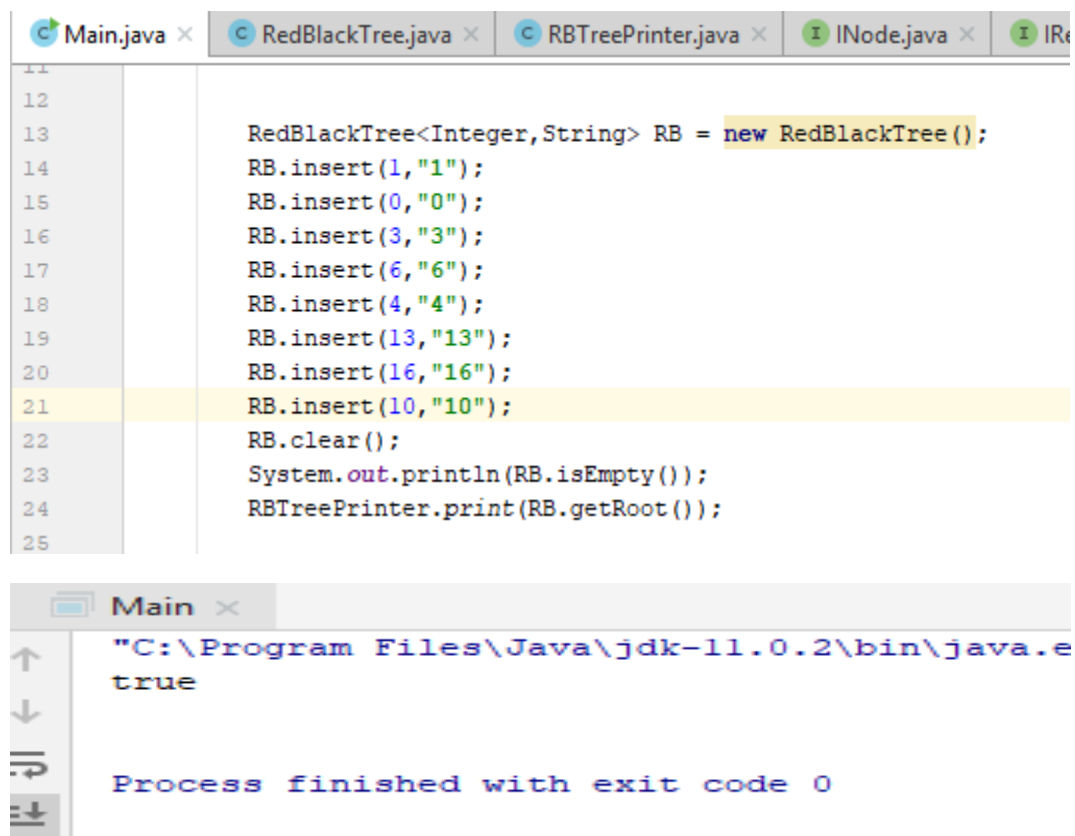
```
Main.java x RedBlackTree.java x RBTreePrinter.java x INode.java x I
11
12
13 RedBlackTree<Integer,String> RB = new RedBlackTree();
14 RB.insert(1,"1");
15 RB.insert(0,"0");
16 RB.insert(3,"3");
17 RB.insert(6,"6");
18 RB.insert(4,"4");
19 RB.insert(13,"13");
20 RB.insert(16,"16");
21 RB.insert(10,"10");
22 RB.delete(key: 6);
23 System.out.println(RB.search(key: 13));
24 System.out.println(RB.contains(1));
25 System.out.println(RB.contains(6));
26 System.out.println(RB.isEmpty());
27 RBTreePrinter.print(RB.getRoot());
```

```
Main x
"C:\Program Files\Java\jdk-11.0.2\bin\java.exe" "-java
13
true
false
false
```

```
graph TD
    4["4 (B)"] --- 1["1 (R)"]
    4 --- 13["13 (R)"]
    1 --- 0["0 (B)"]
    1 --- 3["3 (B)"]
    13 --- 10["10 (B)"]
    13 --- 16["16 (B)"]
```

➤ Example 4:

Here we inserted some nodes , then we clear the tree , then we check if the tree is empty which is true , then we print it which is empty.



The image shows a screenshot of an IDE with two windows. The top window, titled 'Main.java', contains the following Java code:

```
11
12
13     RedBlackTree<Integer,String> RB = new RedBlackTree();
14     RB.insert(1,"1");
15     RB.insert(0,"0");
16     RB.insert(3,"3");
17     RB.insert(6,"6");
18     RB.insert(4,"4");
19     RB.insert(13,"13");
20     RB.insert(16,"16");
21     RB.insert(10,"10");
22     RB.clear();
23     System.out.println(RB.isEmpty());
24     RBTreePrinter.print(RB.getRoot());
25
```

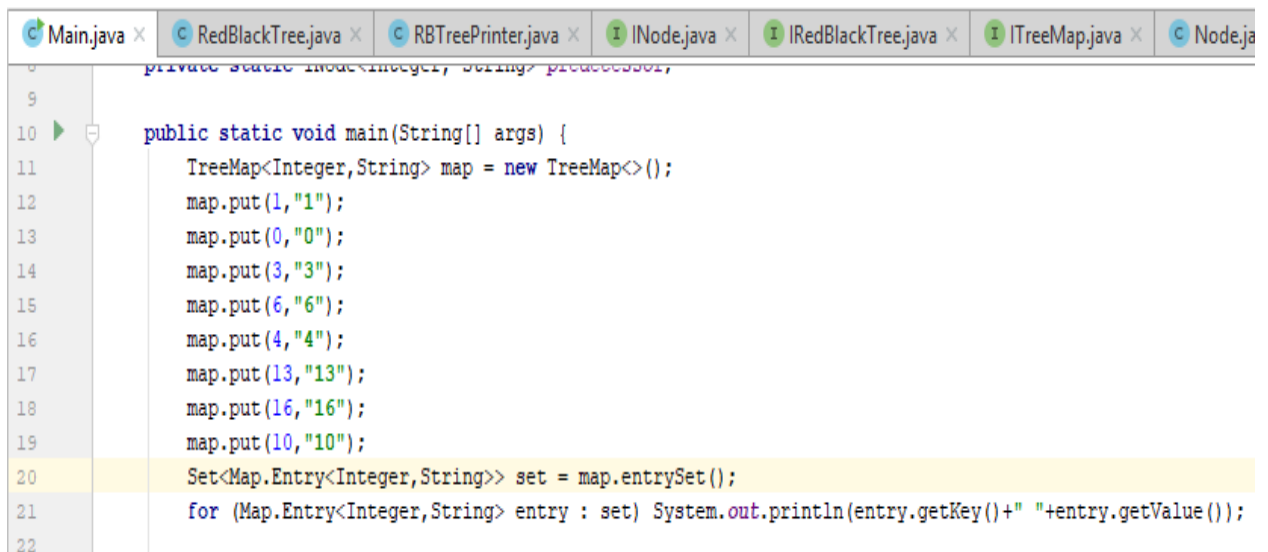
The bottom window, titled 'Main', shows the execution output:

```
"C:\Program Files\Java\jdk-11.0.2\bin\java.e
true

Process finished with exit code 0
```

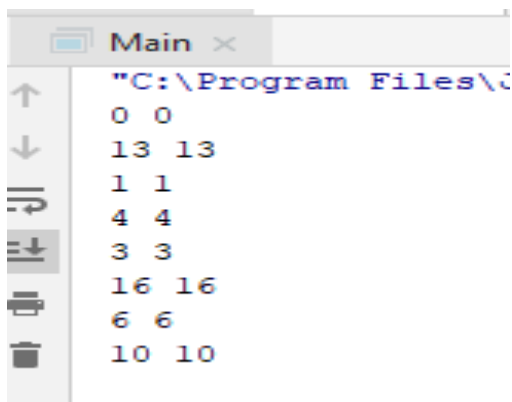

➤ Example 5:

Here we put nodes in the treeMap , then we called “entrySet” method to return the contents of the mapTree then we print the keys and the values of the contents.



```
private static Node<Integer, String> predecessor, successor;

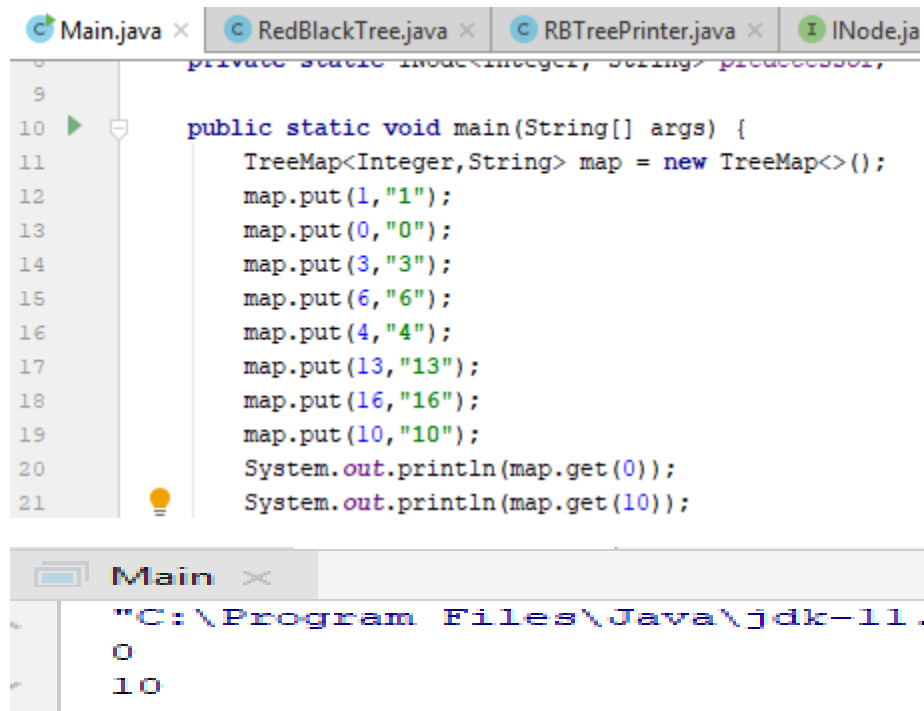
9
10 public static void main(String[] args) {
11     TreeMap<Integer, String> map = new TreeMap<>();
12     map.put(1, "1");
13     map.put(0, "0");
14     map.put(3, "3");
15     map.put(6, "6");
16     map.put(4, "4");
17     map.put(13, "13");
18     map.put(16, "16");
19     map.put(10, "10");
20     Set<Map.Entry<Integer, String>> set = map.entrySet();
21     for (Map.Entry<Integer, String> entry : set) System.out.println(entry.getKey()+" "+entry.getValue());
22 }
```



```
Main x
"C:\Program Files\
0 0
13 13
1 1
4 4
3 3
16 16
6 6
10 10
```

➤ Example 6:

Here we put some elements in the mapTree ,
then we print the values of the keys “0” , “10”.



The screenshot shows an IDE with four tabs: Main.java, RedBlackTree.java, RBTreePrinter.java, and INode.java. The Main.java tab is active, displaying the following code:

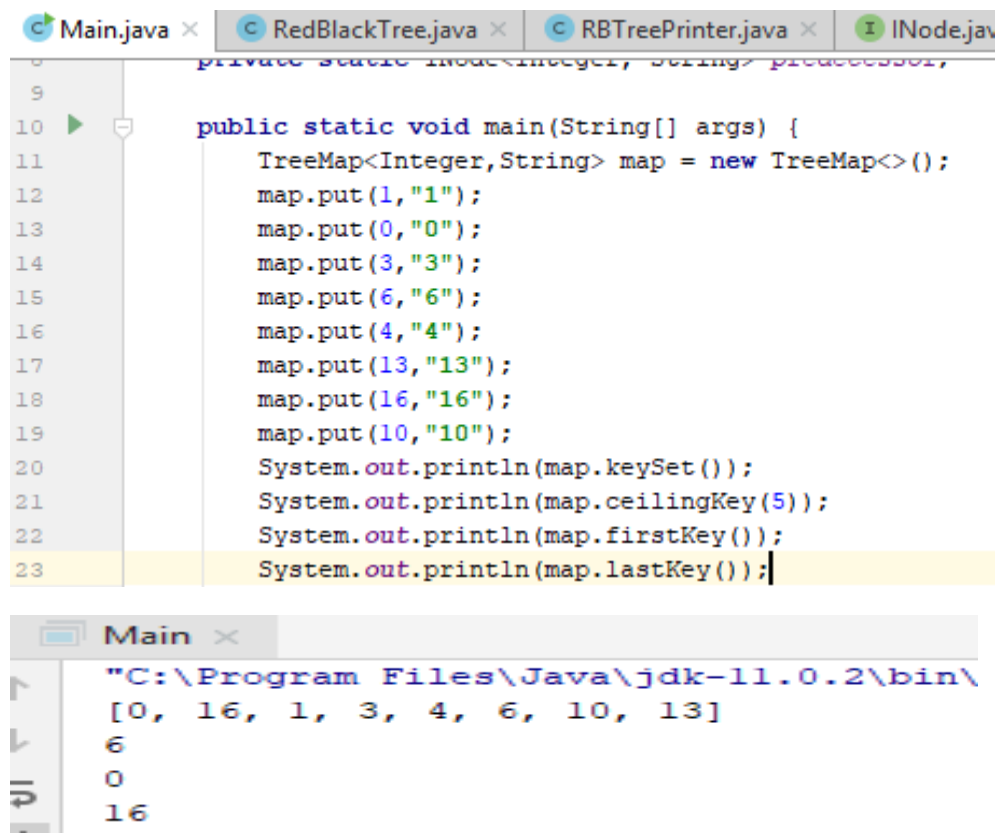
```
8 private static Inode<Integer, String> predecessor,  
9  
10 public static void main(String[] args) {  
11     TreeMap<Integer,String> map = new TreeMap<>();  
12     map.put(1,"1");  
13     map.put(0,"0");  
14     map.put(3,"3");  
15     map.put(6,"6");  
16     map.put(4,"4");  
17     map.put(13,"13");  
18     map.put(16,"16");  
19     map.put(10,"10");  
20     System.out.println(map.get(0));  
21     System.out.println(map.get(10));
```

Below the code editor, the output window titled "Main" shows the execution results:

```
"C:\Program Files\Java\jdk-11.  
0  
10
```

➤ Example 7:

Here we put some elements in the treeMap , then we print the set of keys , then we print the ceiling key(5) which is 6 in the treeMap , then we print the first key in the treeMap which is 0 , then we print the last key in the treeMap which is 16.



The screenshot shows a Java IDE with four tabs: Main.java, RedBlackTree.java, RBTreePrinter.java, and INode.java. The Main.java tab is active, displaying the following code:

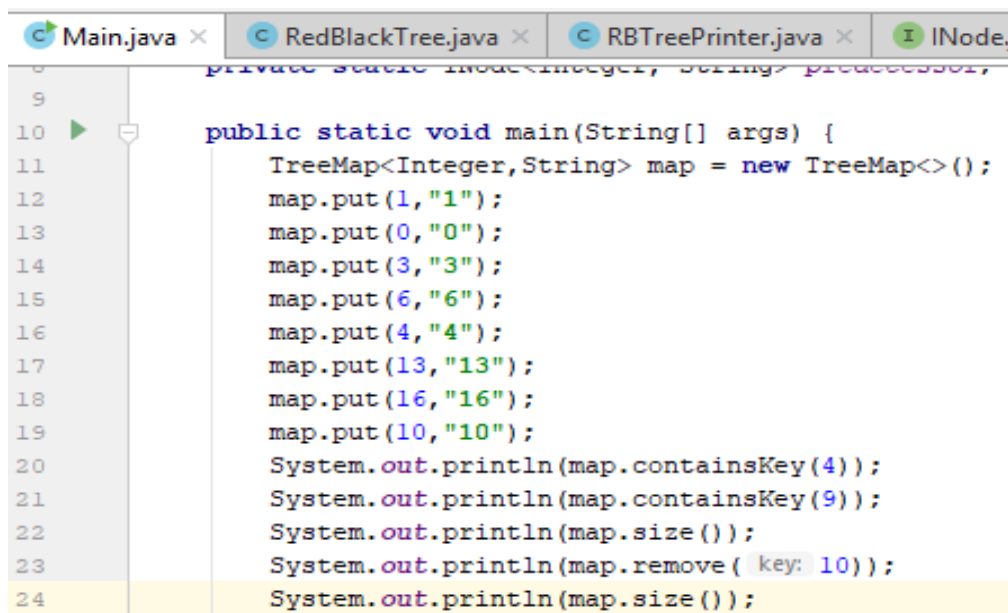
```
10 public static void main(String[] args) {
11     TreeMap<Integer,String> map = new TreeMap<>();
12     map.put(1,"1");
13     map.put(0,"0");
14     map.put(3,"3");
15     map.put(6,"6");
16     map.put(4,"4");
17     map.put(13,"13");
18     map.put(16,"16");
19     map.put(10,"10");
20     System.out.println(map.keySet());
21     System.out.println(map.ceilingKey(5));
22     System.out.println(map.firstKey());
23     System.out.println(map.lastKey());
}
```

Below the code editor, the output window titled "Main" shows the following output:

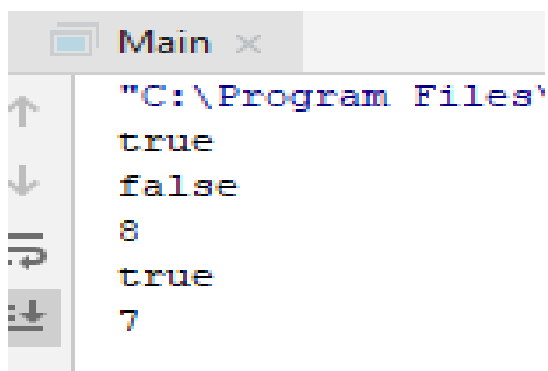
```
"C:\Program Files\Java\jdk-11.0.2\bin\
[0, 16, 1, 3, 4, 6, 10, 13]
6
0
16
```

➤ Example 8:

Here we put some nodes in the treeMap , then we check if it contains key(4) which is true , then we check if it contains key(9) which is false, then we print the size of the treeMap , then we remove the element of key 10 ,then we print the size again.



```
9
10 public static void main(String[] args) {
11     TreeMap<Integer,String> map = new TreeMap<>();
12     map.put(1,"1");
13     map.put(0,"0");
14     map.put(3,"3");
15     map.put(6,"6");
16     map.put(4,"4");
17     map.put(13,"13");
18     map.put(16,"16");
19     map.put(10,"10");
20     System.out.println(map.containsKey(4));
21     System.out.println(map.containsKey(9));
22     System.out.println(map.size());
23     System.out.println(map.remove(key: 10));
24     System.out.println(map.size());
```



```
Main x
"C:\Program Files"
true
false
8
true
7
```