

Project Report

Introduction

I implemented an event counter using red-black tree. Each event has two fields: *id* and *count*, where *count* is the number of active events with the given *id*. The event counter stores only those *id*'s whose *count* is > 0 . Once a *count* drops below 1, that *ID* is removed. My program initially builds a red-black tree from a sorted list of n events (in ascending order of *id*) in $O(n)$ time. My program takes a list of commands from standard input stream and gives the appropriate output. I ran the program on all the 4 test files and got the correct result each time. Furthermore, the redblack tree has many other functions that are described in detail later in the report.

Machine Details

1. Processor
 - 2.7GHz dual-core Intel Core i5 processor with Turbo Boost up to 3.1GHz
 - 3MB shared L3 cache
2. Memory
 - 8GB of 1866MHz LPDDR3 (RAM)
 - 256GB PCIe-based flash storage (Internal Memory)
3. Graphics
 - Intel Iris Graphics 6100

Compiler Used

Java Compiler version : 1.7.0_79

Java Runtime Environment Version : 1.7.0_79-b15

Compiling and Running

make

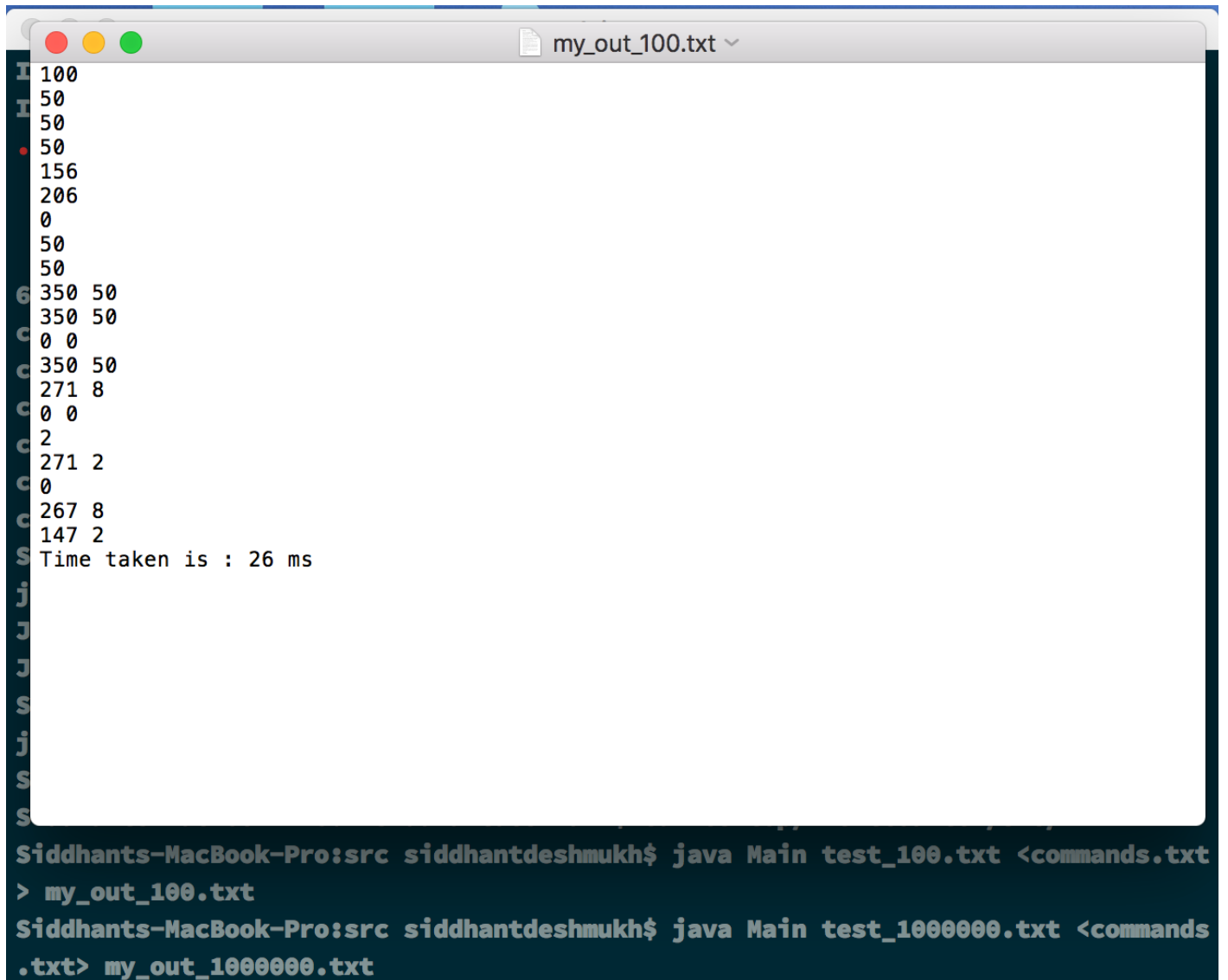
```
java bbst test_100.txt <commands.txt> my_out_100.txt
```

```
java bbst test_1000000.txt <commands.txt> my_out_1000000.txt
```

```
java bbst test_10000000.txt <commands.txt> my_out_10000000.txt
```

```
java -Xms1024m -Xmx8000m bbst test_100000000.txt <commands.txt> my_out_100000000.txt
```

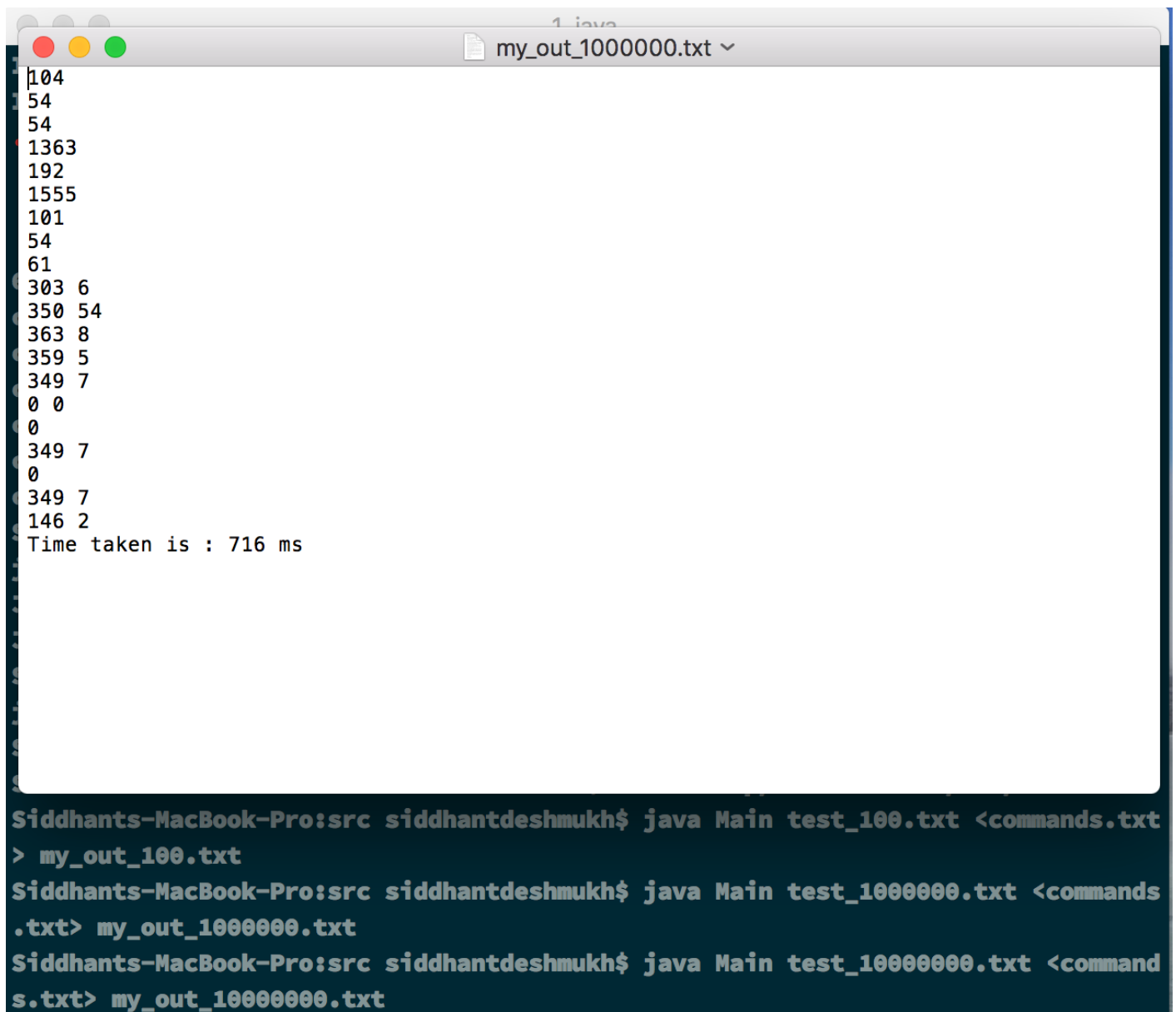
Results



```
100
50
50
50
156
206
0
50
50
350 50
350 50
0 0
350 50
271 8
0 0
2
271 2
0
267 8
147 2
Time taken is : 26 ms

Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_100.txt <commands.txt
> my_out_100.txt
Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_1000000.txt <commands
.txt> my_out_1000000.txt
```

Figure 1: Output for test_100.txt with the time in ms.

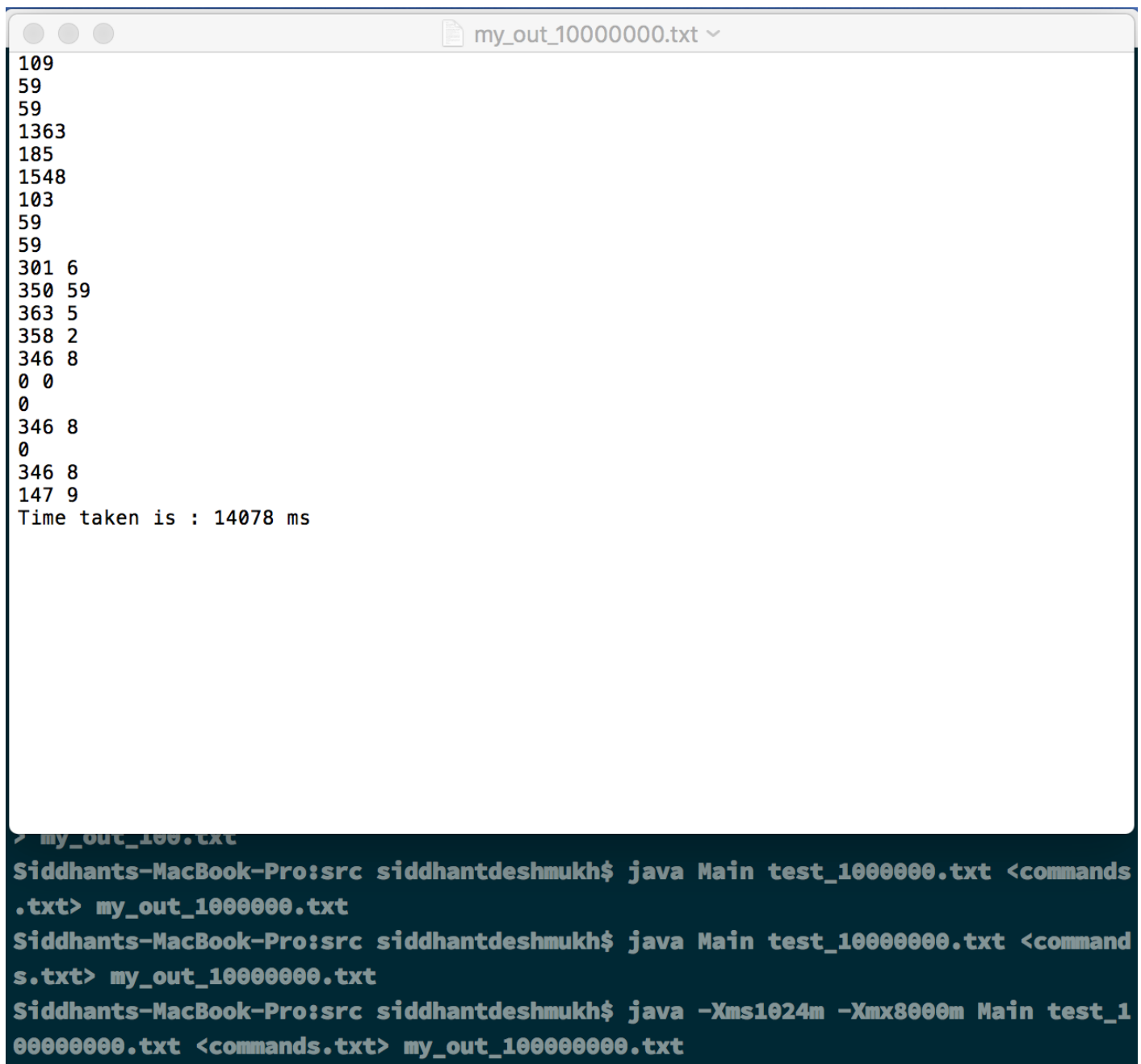


The image shows a Java application window titled "1.java" with a file named "my_out_1000000.txt" open. The window displays a list of numbers: 104, 54, 54, 1363, 192, 1555, 101, 54, 61, 303 6, 350 54, 363 8, 359 5, 349 7, 0 0, 0, 349 7, 0, 349 7, 146 2, and "Time taken is : 716 ms". Below the window is a terminal window showing the commands used to run the program: "Siddhants-MacBook-Pro:src siddhantdeshmukh\$ java Main test_100.txt <commands.txt > my_out_100.txt", "Siddhants-MacBook-Pro:src siddhantdeshmukh\$ java Main test_1000000.txt <commands.txt> my_out_1000000.txt", and "Siddhants-MacBook-Pro:src siddhantdeshmukh\$ java Main test_10000000.txt <command s.txt> my_out_10000000.txt".

```
104
54
54
1363
192
1555
101
54
61
303 6
350 54
363 8
359 5
349 7
0 0
0
349 7
0
349 7
146 2
Time taken is : 716 ms

Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_100.txt <commands.txt
> my_out_100.txt
Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_1000000.txt <commands
.txt> my_out_1000000.txt
Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_10000000.txt <command
s.txt> my_out_10000000.txt
```

Figure 2: Output for test_1000000.txt with the time in ms.



```
109
59
59
1363
185
1548
103
59
59
301 6
350 59
363 5
358 2
346 8
0 0
0
346 8
0
346 8
147 9
Time taken is : 14078 ms
```

> my_out_100.txt

```
Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_1000000.txt <commands.txt> my_out_1000000.txt
Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_10000000.txt <command s.txt> my_out_10000000.txt
Siddhants-MacBook-Pro:src siddhantdeshmukh$ java -Xms1024m -Xmx8000m Main test_100000000.txt <commands.txt> my_out_100000000.txt
```

Figure 3: Output for test_10000000.txt with the time in ms.

```
my_out_100000000.txt
106
56
56
1344
168
1512
93
56
66
303 3
350 56
362 8
358 10
349 10
0 0
0
349 10
0
349 10
149 7
Time taken is : 211249 ms

Siddhants-MacBook-Pro:src siddhantdeshmukh$ java Main test_100000000.txt <command
s.txt> my_out_100000000.txt
Siddhants-MacBook-Pro:src siddhantdeshmukh$ java -Xms1024m -Xmx8000m Main test_1
00000000.txt <commands.txt> my_out_100000000.txt
Siddhants-MacBook-Pro:src siddhantdeshmukh$
```

Figure 4: Output for test_100000000.txt with the time in ms.

Structure of the Program

- I) **Class Node** : Contains the definitions and constructors to create a node for the Red Black Tree. Also contains pretty() function to print the tree. The class implements the Sentinel Interface.
- 1) Instance variables :
 - i) Node left : stores left child of the node.
 - ii) Node right : stores right child of the node.
 - iii) Node parent : stores parent of the node.
 - iv) String color : stores color of the node("red" or "black").
 - v) String type : type can be either "node" or "nil". Type is "nil" if node is the sentinel(see class Sentinel)
 - vi) int count : stores the number of active events for the node id
 - vii) int id : node id acts as an identifier and a key in the tree. Tree is arranged based on the id value of the nodes.
 - 2) Node() : Constructor used to initialize node to default values when no arguments given.
 - 3) Node(String str) : Constructor used to initialize a "nil" node and makes the type of the node "str"
 - 4) Node(int k) : Constructor used to initialize a node with the given id as k.
 - 5) Node(int k,int c) : Constructor used to initialize a node with the given id as k and count as c.
 - 6) void pretty(int l) : Function used to recursively print the tree in a pretty format where the number of dots (".") before the node indicates the level of that particular node.
- II) **Class RedBlackTree** : Contains the definitions and constructors to create a Red Black Tree. The class implements the Sentinel Interface.
- 1) Instance Variables :
 - i) Node root : stores the root node of the Red Black Tree.
 - ii) int sum : variable is used by the inRange function to compute the sum of all the counts in the given range. Is set to 0 each time inRange is called.
 - 2) RedBlackTree(Node r) : Constructor used to initialize a Red Black Tree with the given node as root. Rest of the fields for the root node and Red Black Tree object are set to default values. Parent and the children are set to "nil" and color is set to "black".
 - 3) RedBlackTree() : Constructor used to initialize a Red Black Tree with the default values. Parent, root and the children are set to "nil" and color is set to "black".
 - 4) int insert(Node n) : Standard Red Black Tree insert algorithm the initially inserts as if inserting in a BST and calls rbInsertFix(Node z) to check whether any RBT violations have occurred and fix them.

- 5) void rblInsertFix(Node z) : Fixes all the RBT violations that occurred after inserting by using left rotations, right rotations and colorings. There are separate functions built for left rotate and right rotate.
- 6) void transplant(Node u,Node v) : replaces the subtree rooted at node u with the subtree rooted at node v. Used for RBT delete.
- 7) void delete(Node z) : Performs the RBT delete node operation. Calls the deleteFixup (Node x) function to check whether any RBT violations have occurred and fix them.
- 8) void deleteFixup(Node x) : Fixes all the RBT violations that occurred after deleting by using left rotations, right rotations and colorings. There are separate functions built for left rotate and right rotate.
- 9) Node treeMinimum(Node n) : Returns the node with the minimum id value in the tree rooted at Node n.
- 10) void prettyPrint() : Prints the RBT by calling the recursive function pretty(). This function is used to recursively print the tree in a pretty format where the number of dots (".") before the node indicates the level of that particular node.
- 11) void leftRotate(Node x) : Assumes that right child of Node x is not "nil". It performs a left rotate operation at Node x. Right child of Node x becomes its new parent as a result.
- 12) void rightRotate(Node x) : Assumes that left child of Node x is not "nil". It performs a right rotate operation at Node x. Left child of Node x becomes its new parent as a result.
- 13) Node search(int theid) : Returns the node with id "theid". Used by many of the other functions as search can be very useful.
- 14) void Increase(int theid,int m) : Increases the count of the event "theid" by "m". If theid is not present it inserts it. Prints the count of "theid" after the addition.
- 15) void Reduce(int theid,int m) : Decreases the count of "theid" by "m". If count of "theid" becomes less than or equal to 0, removes "theid" from the counter. Prints the count of "theid" after the deletion, or prints 0 if "theid" is removed or not present.
- 16) void Count(int theid) : Prints the count of "theid". If not present, prints 0.
- 17) void InRange(int id1,int id2) : Prints the total count for id's between "id1" and "id2" inclusively. Assumes that "id1" \leq "id2".
- 18) void recAdd(int id1,int id2,Node n) : InRange uses recAdd to recursively calculate the sum of count in between "id1" and "id2". Node n is the current node that is being processed.
- 19) void Next(int theid) : Prints the id and the count of the event with the lowest id that is greater than "theid". Prints "0 0", if there is no next id.
- 20) void Previous(int theid) : Prints the id and the count of the event with the greatest key that is less than "theid". Prints "0 0", if there is no previous id.
- 21) Node Successor(int theid) : Returns the inorder successor of node with id "theid".
- 22) Node genTree(int[][] arr,int start,int end,Node par) : Function is used to initially generate the tree from a sorted input in O(n) time. "arr" contains the the sorted input ids and count for the corresponding ids. "start" and "end" are used to recursively generate the tree by continuously inserting the middle element from the given array. "par" contains the parent node to ensure that all parent node values are correctly set. genTree() uses the following approach: Get the middle element of the array and make it the root. Recursively get the middle element of the left half and make it the left child of the root and get the middle

element of the right half and make it the right child of the root. Hence, the tree is successfully created in $O(n)$ time.

III) **Class Sentinel** : contains an interface “Sentinel” that defines the sentinel node “nil” that is used for the children of the leaf nodes and the parent of the root nodes.

IV) **Class bbst** : Contains public static void main. Main program flow starts from here. Also contains all the unit test functions for each of the project functions implemented. The class also implements the Sentinel Interface.

- 1) public static void main(String[] args) : Main program flow starts from here. It reads in the file and makes the calls to initialize the RBT and insert all the nodes using genTree(). It handles all the possible commands(commands.txt) using a switch case. It closes the file and looks out for exceptions while dealing with the file and I/o.
- 2) static void testSearch(int x) : tests the Search() function.
- 3) static void testPrint() : tests the prettyPrint() function.
- 4) static void testCount() : tests the Count() function.
- 5) static void testNext(int x) : tests the Next() function.
- 6) static void testPrevious(int x) : tests the Previous() function.
- 7) static void testIncrease(int id,int c) : tests the Increase() function.
- 8) static void testReduce(int id,int c) : tests the Reduce () function.
- 9) static void testInRange(int id1,int id2) : tests the InRange() function.