COP5536 Project Report
UFID #13103195
Compiler Used:

```
machine:~ bishalgautam$ java -version
java version "1.6.0_26"
Java(TM) SE Runtime Environment (build 1.6.0_26-b03-384-10M3425)
Java HotSpot(TM) 64-Bit Server VM (build 20.1-b02-384, mixed mode)
```

Structure of Program:

I have used following classes in the implementation:

1. RedBlackTree.java
2. RedBlackNode.java
3. bbst.java

RedBlackTree Class has following methods:

buildTree(ArrayList<T> eventList, ArrayList<Integer> countList, **int** start, **int** end)

It takes two arraylists where the eventIds and count for the eventIds are stored. Since we are recursively building the tree we are also passing the start and end index. This function will initially build the tree in O(n) time.

levelOrderTraversal(RedBlackNode<T> node)

I am following the pseudo code from the text book(CLRS) hence I need to add the parent pointer for the nodes in the tree. So this method adds the parent pointer and the color to the nodes by traversing the tree in levelOrder. It also takes O(n) time.

So the total time to build a tree with the colors and parent pointers will take O(n) + O(n) => O(n) time.

rangeCount(RedBlackNode<T> root, T lo, T hi)

Using the logic that the path will remain same till some point traversing down the tree; only after the split node we need to sum up the counts of the nodes that lies in the given range, it takes O(logn + S) time. But if the split happens at the root it will be of the order of n. It returns the total number of count of all the events lying in the given range(lo<hi).

getClosestNode(T value)

To get the **previous**(*the*ID) and **next**(*the*ID) values it is very helpful to find the closest node for a given eventId. It is also implemented in O(logn) time.

Search(T value)

It returns a node if the value passed matches any of the nodes evenId else returns null. It is helpful for the Count(*the*ID) command. It takes O(logn) time.

Apart, from these **leftRotate** , **rightRotate**, **insert** , **insertFixup** , **remove** , **removeFixup** are implemented as per the pseudocode in the textbook(CLRS). Apart from them **treeSuccessor** and **treePredecessor** are implemented to find the inorder **next** and inorder **previous** node from a tree.

RedBlackNode Class has two constructor defined one with the eventId and count to create the internal nodes and one without any arguments to create the leaf nodes which is by default black and is named **nil** throughout the program.

bbst Class has the code that reads the file passed as argument and store the eventIds and count in two different arraylist. Here the commands are read from the stdin and corresponding action is initiated.

Commands	Longest Function Sequence
Increase(*theID*, *m*)	Search and Insert Complexity : O(lgn) + O(lgn) => O(lgn)
Reduce(*theID*, *m)*	Search and Delete Complexity : O(lgn) + O(lgn) => O(lgn)
Count(*theID*)	Search Complexity : O(lgn)
InRange(*ID1*, *ID2*)	rangeCount Complexity : O(lgn + S)
Next(*theID*)	getClosetNode and treeSuccessor Complexity : O(lgn) + O(lgn) => O(lgn)
Previous*(theID)*	getClosestNode and treePredecessor Complexity : O(lgn) + O(lgn) => O(lgn)

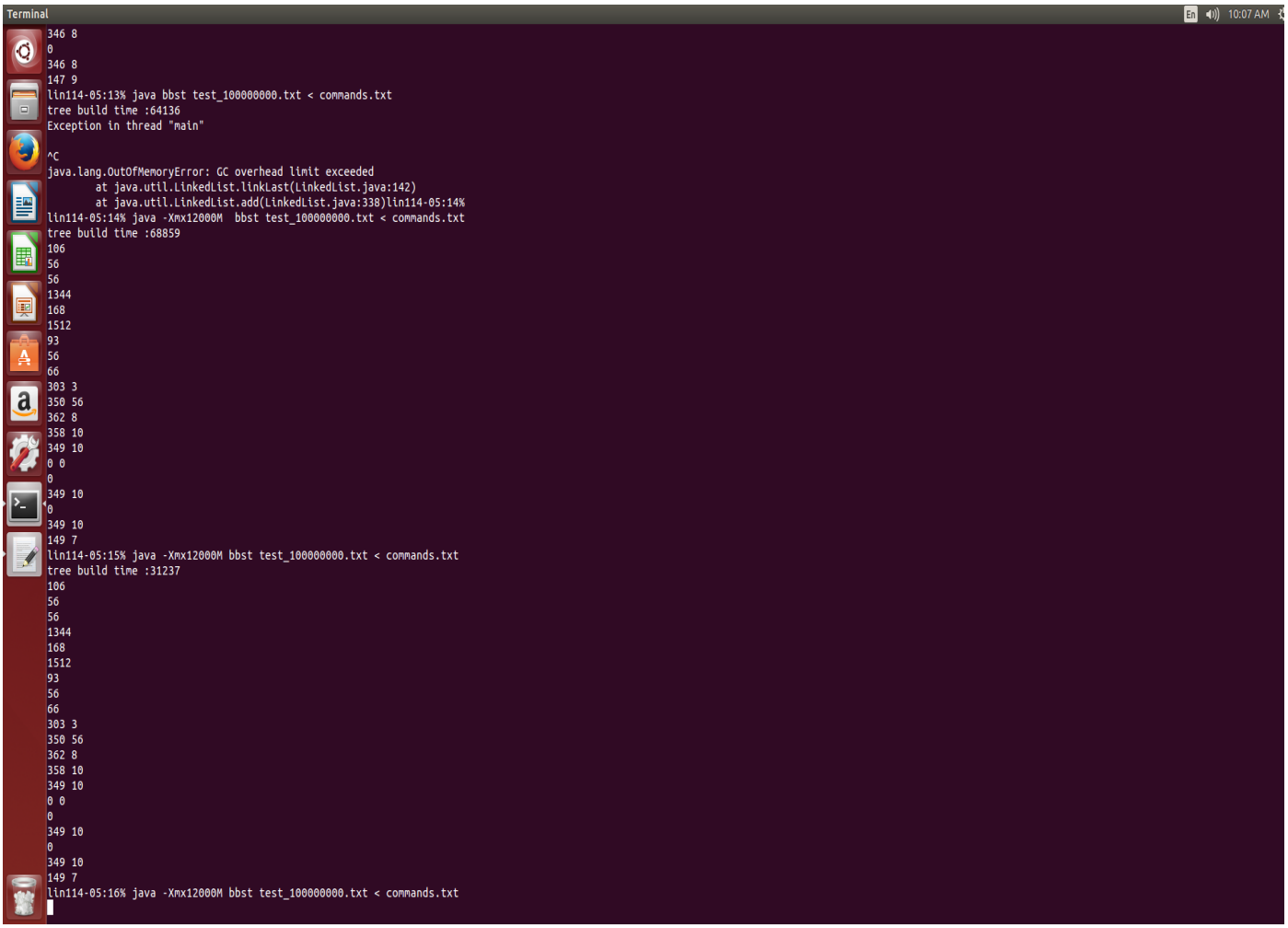
Flow of the Program

In the bbst main is defined and in the main first a tree is created from the RebBlackTree class.Then the treebuild function is called that builds the tree using divide and conquer with child pointers. To assign the parent pointer and color the function inOrderTraversal is called.

After this the scanner waits for the input in stdin if it gets any input it checks for the first word and accordingly execute the commands.

Space Complexity and Improvements

For running the large dataset of the order of 10^8 nodes the heap size of the VM must be increased. I am attaching the screenshot where I was able to run the program in linear time with correct output only after increasing the heap size.



In my logic I am building the tree once and again traversing the tree to assign the parent pointer. Since both the steps are of the order of n the program runs little slow than many other students. I wrote the program using the comparable interface and because some generics type casting is not allowed in Java, I couldn't change my logic and build the tree in one pass.