COP 5536 Programming Project

Submitted by Srinivas Gubbala (2131 7376)

Program Flow:

bbst :: Main ()

- ✓ Read input from the file provided as argument.
- ✓ Create RedBlackTree.class object which is used to create given events as a Red Black Tree.
- ✓ Build a Red Black Tree.
- ✓ Read Standard input from commands file.
- ✓ Call the appropriate methods in RedBlackTree.class based on each command's first argument.

RedBlackTree :: increase(), reduce(), count(), inRange(), next(), previous()

findNode()

insertNode(), fixRBTreeInsert(), rotateRight(), rotateLeft(),

deleteNode(), fixRBTreeDelete(), transplant(), treeMinimum()

- ✓ Inserts nodes that read from input into the Red Black Tree.
- ✓ Fix the Red Black tree incase if the tree violates RB Tree properties after insertion.
- ✓ Implements increase, reduce, count, inrange, next, previous functionalities.

Node :: Node()

- ✓ Initialize id, count, parent, right and left children.

Functions:

Increase (int eventID, int m) {}: Increases count of given eventID by m. It will search for the given eventID in the constructed Red Black Tree using BST traversal. If found, the eventID count will be increased by m. Otherwise a new node will be inserted with the given eventID and count. Run time for searching the node and for insertion will be O(log n) + O(log n) which will be O(log n).

Reduce (int eventID, int count) {}: Reduces count of given eventID by m. It will search for the given eventID in the constructed Red Black Tree using BST traversal. If found, the eventID count will be reduced by m. If the count becomes less than zero after reducing it by m, the eventID will be deleted. Run time for searching the node and for deletion will be O (log n) + O (log n) which will be O (log n).

Count (int eventID) {}: Returns the count of the given eventID. It will search for the given eventID in the constructed Red Black Tree using BST traversal. If found, the eventID count will be returned. Otherwise it returns zero. Run time for searching will be O (log n).

InRange(int eventID1, int eventID2) {}: Returns the sum of count of event ID's lying in between eventID1 and eventID2 including eventID1 and eventID2. It will traverse the paths that fall between eventID1 and eventID2 using BST traversal. Run time for finding the height and visiting the event ID's in that interval = O (log n) + O(s) = O (log n +s) where s will be the number of events between eventID1 and eventID2.

Next (int eventID) {}: Returns the least eventID greater than the given eventID along with its count. Run time will be finding the utmost left of right child or tracing back which is O (log n) + O (log n) which in turn is O (log n).

Previous (int eventID) {}: Returns the greatest eventID less than the given eventID along with its count. Run time will be finding the utmost right of left child or tracing back which is O (log n) + O (log n) which in turn is O (log n).

Implementation:

Increase:

- ✓ This is implemented internally using insertNode method. A node will be inserted or updated into the Red Black Tree. First it finds for the node with the given id using findNode method.
- ✓ If the node is present it will update the count. Otherwise, the node will be inserted into the tree as a normal BST. By default, the node will be colored red.
    - Now if the parent is black, there won't be any violation of RB Tree properties.
    - If the parent is red, then double red violation occurs. Here there will be two cases to consider and it was implemented as fixRBTreeInsert method.
        - If the sibling of parent is red or black/absent. In first case, simply parent and parent sibling are recolored. Then check for the same with parent's parent and go till the root as it may propagate up in the tree.
        - In the second case the node will be rotated. The direction of rotation will be decided based on whether it is right or left child to its parent.

Reduce:

- ✓ This is implemented internally using deleteNode method. A node will be deleted or updated into the Red Black Tree.
- ✓ If the count becomes less than zero, then update. Else the node will be deleted. Here deletion may violate RB Tree properties.
- ✓ If the node to be deleted has no or one child, then the node will be deleted or replaced with child respectively.

If the node to be deleted has two children, first we will rotate to make it to have one child.
Then if the color of node is Black, then fixRBTreeDelete method will handle the violation.

Count: This was implemented internally by calling findNode method. For the given id findNode will search for the node in RB Tree using BST traversal. If the node is present, then it returns and prints the count. Otherwise it will return 0.

InRange: This was implemented internally by calling the inRangeHelper function recursively based on the id1 and id2 starting from root. If value of root's key is greater than id1, then recursively call in left subtree. If value of root's key is in range, then add the root id to the count. If value of root's key is smaller than id2, then recursively call in right subtree. Finally, the count will be returned and printed.

Next: This was implemented internally by calling the next Helper method. If there is a right child present to the given node, then it returns the left most node of that right child which will be the next greater element for the given element. If there is no right child it traverses until it becomes the right child to its parent. Then it returns its parent.

Previous: This was implemented internally by calling the previous Helper method. If there is a left child present to the given node, then it returns the right most node of that right child which will be the next greater element for the given element. If there is no left child it traverses until it becomes the left child to its parent. Then it returns its parent.

Compiler Used:

Java – Version (1.8.0_74)

Compiler – javac (J2SE 8)

Files Included:

bbst.java

makefile

Report.pdf

my_out_100.txt , my_out_1000000.txt, my_out_10000000.txt, my_out_100000000.txt

How to run:

time java –Xms8g bbst input_file < commands.txt > output_file

-Xms8g will ensure the project to run without any out of memory exceptions.

Ex: time java –Xms8g bbst test_100000000.txt < commands.txt > my_out_100000000.txt

Run Times:

lin114-10:15% time java bbst test_100.txt < commands.txt > myOut_100.txt

0.131u 0.024s 0:00.12 125.0% 0+0k 0+80io 0pf+0w

lin114-10:16% time java com bbst test_1000000.txt < commands.txt > myOut_1000000.txt

1.009u 0.220s 0:00.61 200.0% 0+0k 19008+80io 0pf+0w

lin114-10:17% time java bbst test_10000000.txt < commands.txt > myOut_10000000.txt

7.026u 3.056s 0:04.22 238.6% 0+0k 209568+80io 0pf+0w

lin114-10:18% time java bbst test_100000000.txt < commands.txt > myOut_100000000.txt

215.808u 17.955s 0:54.14 431.7% 0+0k 0+120io 0pf+0w

References:

Introduction to Algorithms CLRS 3rd edition

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-10-red-black-trees-rotations-insertions-deletions/lec10.pdf

https://www.cs.usfca.edu/~galles/visualization/RedBlack.html