Project Description

Requirement:

Implementing an event counter using red-black tree and the counter supports the following operations: increase(theID, m), reduce(theID, m), count(theID), inrange(ID1, ID2), next(theID), previous(theID).

Develop Environment:

Language: Java
java version "1.8.0_20"
Java(TM) SE Runtime Environment (build 1.8.0_20-b26)
Java HotSpot(TM) 64-Bit Server VM (build 25.20-b23, mixed mode)
Complier: javac 1.8.0_20
Note: When running the huge test file bigger than 1GB, the program should be run with a bigger heap size, such as java –Xmx8000m bbst testfile

Project Structure

The project is consisted of three main source files:
bbst.java, RBtree.java, RBTnode.java

1. bbst.java

This file containing the main function is used to support redirected input from a file "file-name" which contains the initial sorted list. When running this function, it will create an object “tree” using RBtree.class. Then the object will perform the corresponding functions to initialize a balanced binary search(BBST) and change it to be red-black tree in O(n) time with the data in test file inputted.

After creating the red-black tree, the program is waiting for inputting commands. When command is inputted, the object “tree” will perform the according functions such as increase(), reduce(), count() etc. to print the results according to the commands and quit the program when meet “quit”.

2. RBTnode.java

This file creates RBTnode class which contains nodes in the red-black tree. These nodes can each present the event in the requirement and they have such attributes: ID, count, color, leftchild，rightchild, parent. Some functions are used to create new attributes such as uncle(node), grandparent(node), Right_Minimum(node), Left_Maxmum(node), sibling(node), tree_successor(node), tree_predecessor(node). These attributes will be used when new RBTnode objects created in RBtree.java.

3. RBtree.java

This file is the main source file to implement all the functions this project needed. When this file is performed, an object “tree” will be created to using all kinds of functions of this file to implement a serious requirements of the project. There are following main functions in this file: insertNode(), setNode(), LL/RR/RL/LRrotation(), find/findLost(), deletNode(), setDelete(), ArrayToRBTree(), findLCA() , increase(theID, m), reduce(theID, m), count(theID), inrange(ID1, ID2), next(theID), previous(theID).
The whole program can be divided into parts: creating the red-black tree using data of test file; serious operations the project inquires; insert node to red-black tree; delete node from red-black tree.

a. Period of creating red-black tree:

It happens at the beginning of the program and object “tree” will perform ArrayToRBTree() function to create a red-black tree using sorted array in the test file.

b. performing increase(theID,m), reduce(theID,m), count(theID), inrange(ID1,ID2), next(theID), previous(theID) functions.

For increase(theID, m), “tree” will perform find() function to find the node(event), if the node doesn’t exist then it will go to insert node period.

For reduce(theID, m), “tree” will perform find() function to find the node(event), reduce the count, and If the node’s count becomes less than or equal to 0, go to delete node period.

For count(theID), “tree” will perform find() function to find the node(event), print the count or 0 when find null node.

For inrange(ID1, ID2), “tree” will perform find() function to find two nodes with ID1 and ID2, then perform findLCA() function to find their lowest common ancestor. From the ancestor node, traversal the sub-tree to get the sum of the nodes’ count when the node’s ID is in the range of ID1 and ID2. If nodes do not exist, perform next() and previous() to find the nodes, then continue inrange(ID1,ID2). This kind of algorithm will ensure the time complexity is O(logn + s) where s is the number of IDs in the range.

For next(theID), “tree” will perform find() function to find the node(event), perform tree_ successor (node) function to find the next node. If node with theID does not exist, “tree” will perform findLost() and then perform tree_ successor (node) function to find the next node. Then print the node’s ID.

For previous(theID), “tree” will perform find() function to find the node(event), perform tree_predecessor(node) function to find the previous node. If node with theID does not exist, “tree” will perform findLost() and then perform tree_predecessor(node) function to find the previous node. Then print the node’s ID.

c. Insert node period:

“tree” will use inerstNode() to insert the node to red-black tree, and then use setNode() to rebalance the new tree to be red-black tree according to the propriety of the red-black tree.

d. Delete node period:
“tree” will use find() to find the node in the tree and then perform deleteNode() to delete the node to red-black tree, then use setDelete() to rebalance the new tree to be red-black tree according to the propriety of the red-black tree.

Function prototypes

In this program, there are some fundamental functions to implement basic functions.
ArrayToRBTree(): Initialize the sorted array to be a red-black tree. From the mid of the array to create the nodes recursively, this will create a BBST and then paint the node with the biggest height to red. This will create a red-black tree in O(n) time complexity.

find/findLost(): From the root of the tree, compare the ID with the node, if ID is smaller, go leftchild of the node else go to right of the node until to find the node with the same ID. If it goes to the end of the tree and cannot still find the node, then return null. findLost() is similar with find(), it will return the parent of the ID if the ID is not in the tree.

insertNode(): From the root of the tree, compare the ID with the node, if ID is smaller, go leftchild of the node else go to right of the node until the end of the tree. Then create a new node with the ID, count color and set the relation of parent, leftchild and rightchild. Then perform setNode().

setNode(): According to the position and color of the nodes, rebalance the tree to be a new red-black tree. It can be divided into several cases. If uncle of the inserted node is red, change the parent to be black grandparent to be red and continue setNode(grandparent). If uncle is black, take LL/RR/RL/LR rotations.

deleteNode(): It can be divided into three cases: node with 2 degrees, 1 degree, 0 degree. For 2 degrees, replace the node with the smallest successor of the right sub-tree using Right_Minimum(node) and change the case to 1 degree case. For 0 degree case, add a black dummy node, then change the case to delete node with 1 degree.

setDelete(): For 1 degree case, there is the deficient node. If the node node is red, make it black, done. If the node is a black root, done. If sibling of the node is black, change colors with 0 red child of sibling or LL/RL/RR/LRrotation with 1 red child of sibling or LR/RLrotation with 2 red child of sibling. If exchange the color of parent and sibling, and then it can be change to the previous cases.

How to run

The command line for this program:
$java bbst file-name

Testfile format:
n
ID1 count1
ID2 count2
...
IDn countn

After inputting the data of the test file, program will read the commands from the standard input stream and print the output to the standard output stream. Use the command specifications described increase(theID, m), reduce(theID, m), count(theID), inrange(ID1, ID2), next(theID), previous(theID) with all lower cases.