ADVANCE DATA STRUCTURES

COP 5536 Programming Project
Spring 2016

Submitted By: Naman Rajpal
UF-ID: 41353307
Email-ID: namanrajpal16@ufl.edu

Project Section Overview:

Compiler used
I have used Java to program the project. I used **Java compiler 'javac.exe'** to compile the programs that I made. Compiler uses **JRE7** (Java runtime environment 7) provided in JDK1.7.0_51 to execute my programs.

Software Used
Windows 7 – Operating System.
Eclipse Juno – IDE used to write the program.
JDK7 – Java Development Kit comprising of compiler and java libraries.
Putty – Remote Linux Terminal software used to execute project on thunder.
WinSCP – Remote Linux File Transfer software used to copy my project on thunder.
GnuWin32 – Software used to run 'make' on a windows platform.

Structure of the Program.

Project consists of two .Java files comprising of three classes:
bbst.java consisting of:
    RBTree.class (class to implement tree)
    bbst.class (Mainclass)
RBNode.java consists of:
    RBNode.class (class to implement red black node)

Structure of RBNode.java

Field Summary

Modifier and Type	Field and Description

private int	**color** (1 for Black, 0 for Red)
private int	**count**
private long	**ID**
private RBNode	**left** (left child of the node)
private static int	**LEFT** (set as -1)
private RBNode	**right** (right child of the node)
private static int	**RIGHT** (set as 1)

Constructor Summary

Constructor and Description
RBNode()
RBNode(int color)
RBNode(long id, int count)
RBNode(long id, int count, int color, RBNode left, RBNode right)

Method Summary

Modifier and Type	**Method and Description**
void	**changeCount(int change)** This function changes the ID of the Node it is called from. Change can be negative or positive.
int	**getColor()** returns color of the node.
int	**getCount()** returns count of the node.
long	**getID()** returns ID of the node.
RBNode	**getLeft()** returns left child.

RBNode	getRight() returns right child.
private RBNode	link(int direction) This function returns RBNode. If incoming int is -1, it will return left otherwise it will return right.
void	setColor(int color)
void	setCount(int count)
void	setID(long iD)
void	setLeft(RBNode left)
(package private) RBNode	setLink(int direction, RBNode n) This function sets the incoming RBNode as left or right child depending upong the value of incoming int. It returns right child in default ir dir is wrong!
void	setRight(RBNode right)

Structure of bbst.java

Structure of class RBTree

Fields/Variables of the Class

Modifier and Type	Field and Description
(package private) static int	**BLACK** Set as 1
private **RBNode**	**head** This is the head file where all the nodes will be inserted as children. All the functions will be called on this object to traverse down.
private static int	**LEFT** Set as -1

(package private) static int	**RED** Set as 0
private static int	**RIGHT** Set as 1
private int	Used in Count function to Gather sum of multiple ids between multiple function calls

Constructor and Description
RBtree() It initializes the tree. It sets the field ‘head’ as null

Method Summary and Description

Modifier and Type	Method and Description
void	**Count**(long id) Count Function prints the count of the node with the incoming ‘id’ parameter. It uses search function which returns a valid existing node.
void	**Increase**(long id, int count) This function was required to be implemented according to the project description. It uses a function named TreeIncreaseSearch(long,int) to search and increase the count of the node with parameter ‘id’. It inserts in the tree if no node is found.
private **RBNode**	**inOrderPredeccessor**(**RBNode** node) It searches for the inorder predecessor for the incoming parameter ‘node’ return the predecessor if found else returns null.
private **RBNode**	**inOrderSuccessor**(**RBNode** node) It searches for the inorder successor for the incoming parameter ‘node’ return the successor if found else returns null.
void	**InRange**(long id1, long id2) This function was required to be implemented for the project requirement. It is called when ‘inrange’ is found on standard input. It calls range function(described below) to does its task.

private **RBNode**	**insert**(**RBNode** root, long id, int count) This is the insertion function. Recursion has been used here. A new node is inserted keeping in mind the BST property and after insertion balancing using single and double rotation functions is done. So, it basically recurs all the way down to place where node has to be inserted and recurs back(return) all the way up to balance the tree up to root.
private **RBNode**	**max**(**RBNode** node) This function finds the maximum element present in the tree whose root is ‘node’(parameter). This function is used in inOrderSuccessor and Next functions.
private **RBNode**	**min**(**RBNode** node) This function finds the minimum element present in the tree whose root is ‘node’(parameter). This function is used in inOrderPredeccessor and Previous functions.
void	**Next**(long id) Next function prints the next greater node present in the tree. Incoming parameter ‘id’ may or may not be present in the tree. If its present in the tree then it simply finds the inorder successor of the node with the given ‘id’ and prints returned id and count. If node is not present in the tree, nextnullsearch(long id) function is called to search for appropriate greater id. It returns ‘0 0’ if id input is greater than the maximum id present in the tree.
RBNode	**nextnullsearch**(long id) This function is used by Next function when input id to Next is not present in the tree. It traverses tree based on input ‘id’ comparing it with reached node’s id and successor to find the appropriate result.
private **RBNode**	**parentsearch**(**RBNode** root, long id) This function return the parent of node with incoming ‘id’ parameter in the ‘root’ subtree.

void	**Previous**(long id) Previous function prints the previous node just less than the ‘id’ present in the tree. Incoming parameter ‘id’ may or may not be present in the tree. If it’s present in the tree then it simply finds the inorder predecessor of the node with the given ‘id’ and prints returned id and count. If node is not present in the tree, previousnullsearch(long id) function is called to search for appropriate node with lesser id. It returns ‘0 0’ if id input is less than the minimum id present in the tree.
RBNode	**previousnullsearch**(long id) This function is used by Previous function when input id to Previous is not present in the tree. It traverses tree based on input ‘id’ comparing it with reached node’s id and predecessor to find the appropriate result.
private void	**range**(**RBNode** node, long id1, long id2, int count) This function is use by Inrange function. It traverses the tree and adds up the counts of id’s which are in the range id1<id<id2.
RBNode	**Rdouble**(**RBNode** root, int lr) Function to implement double rotation. Returns root rotation it twice. It determines the direction using lr. lr is 0 for left and 1 for right. It uses Rsingle rotation function to rotate twice.
void	**Reduce**(long id, int count) This function is executed hen ‘reduce’ command is given on the console. It search for the node with given ‘id’, if it’s found, it reduces its count by the value ‘count’ and if the count becomes less than zero then it uses remove function to remove the node.
(package private) boolean	**remove**(long data) This function is used to remove a node from the tree with id as given ‘data‘. It implements top down

	balancing deletion.
RBNode	**Rsingle**(**RBNode** root, int lr) Function to implement single rotation.lr input is used to determine direction of the rotation.
private **RBNode**	**search**(**RBNode** root, long id) searches the node with ‘id’ in the given ‘root’ subtree. Returns found node else null.
void	**TreeIncreaseSearch**(long id, int count) s
void	**Treeinsert**(long id, int count)

4. Flow of the structure of the program:

The structure of the program is that: Project consists of two .Java files, namely bbst.java and RBNode.java.
For the bbst.java file:
There is a RBTree class.In that class itself we have various functions defined in it.
To insert a node in the tree another function is defined in the class that is the treeinsert function. In that function itself another function namely “ insert” function is called.
To remove all the anomalies in the constructed new tree, like the Rsingle and Rdouble functions.(R and RR).Where R denotes single rotation and RR denotes double rotation.
The **Rsingle** and the **Rdouble** functions are basically used to perform rotations in the tree if ever there are some sort of anomalies occurred in it.
To perform any deletions of the node another function is developed that is the remove function. This function is a iterative function that uses top down balancing approach, which pushes a new red node down to prevent black violation, Reduce function calls the remove function.
In the reduce function we are calling the other two functions namely **search** and **remove**. If the value of count is 0 then the reduce function is called.
Next is the increase function which is used to increase the count of the node with a particular ID or if that Node is not present inserts that values as a new node into the tree. Thus for that case calls the insert function.
The **Reduce** function which is used to reduce the count of the node with a particular ID or if that Node is not present prints a value. If the value of count becomes less than 0 then delete function is called.
max and **min** are the two functions deployed to find the maximum and the minimum node. Also the search function is used to find the node and return the values accordingly.
Count function is used to return the ID of a node with the same count. And for that to run uses search function.
inorderPredecessor and **inorderSuccessor** functions are used to find the just previous and the just next nodes in given tree, these two functions further make call to the max and find min functions.
The next, previous and inrange functions further call inorder successor, in order predecessor and the In Range functions. In the main class all these functions like the insert and all the other functions like

increase, reduce , count, inrange, next previous are called. And the input is taken from a file both to insert the data as well as to run the commands.

Bbst.class:
This function consists of the **main** function which first take sinput from a file reading args[0] and constructs the tree using Treeinsert function, then it waits for the commands from standard input to run the different commands which invoke the associated function on the tree object defined at the beginning.

For the RBNode.java.:
In this RBNode class we have two main functions:
The Setlink function sets the left and the right child.
The Link function returns left and the right child.
Other functions are all getters and setters.

5. Summary:

Overall the Red Black tree data Structures handles the insert and the delete operations efficiently for all the input and commands data sets we had.RBTree search trees have a height that is always O(log N). One consequence of this is that the insert, and delete on a tree like this can be done in O(log N) worst-case time. In constrast, binary search trees have a worst-case height of O(N) and insert, and delete are O(N) in the worst-case.