Advanced Data Structures

Spring 2016 Assignment #1

RB Tree Implementation Report

Author: Alekhya Gupta UFID: 3953-8073

1. Compilation

To compile and run:

The project has been written in java and can be compiled using standard jre1.8.0_73 / jdk1.8.0_60

Unzip the folder submitted. The folder contains the project files (.java files), the makefile and the input files used for testing (.txt files) as well.

UNIX:

First do a make clean

This will remove any previously compiled class files

To compile in the unix environment, run the make command: make

This will compile all the classes mentioned in the makefile

To run, use the command java bbst [arguments]

For redirection of input or output use:

java bbst [arguments]<commandsfile>outputfile

2. Program structure

This program contains 3 classes: Node.java, RedBlackTree.java, bbst.java. Here bbst.java contains the main method. Node.java contains definition of red black tree nodes and RedBlackTree.java contains the implementation of the red black tree functions.

This program is an implementation of 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.

All of the operations are performed on instances of Node class. These instances contain attributes like ID, count, color (red or black), left, right and parent pointers. Sentinel nodes with dummy values like -1,-1 for ID, count and color black are defined.

Red black tree is created recursively from sorted array, by finding the middle element of list and making it the root of the tree to be constructed, then recursively the same is done with its left and right halves. These halves in turn are the left and right subtrees of the middle element. Initially nodes at all levels expect the last (non-sentinel) level are colored black. The last level (non-sentinel) nodes are colored red. This is because, while recursively creating the red black tree only the last level could be incomplete. By coloring this level red I ensure that the black height will be preserved for all nodes and red black tree properties will be guaranteed. The last (sentinel) level containing sentinel nodes is colored black.

Usual red black tree functions like insert, delete and search are implemented.
For insert initially a node is inserted just like it would be in any binary search tree. After that, insertFixer function is called to restore and guarantee the red black tree properties through color swapping and rotations.
Similarly, delete searches for node matching with inputted ID and if found deletes it from red black tree without guaranteeing balance (like BST) or returns without any operation. After that, deleteFixer function is called to restore and guarantee the red black tree properties through color swapping and rotations.
Search is implemented just like search function in binary search tree.

After implementing the usual red black tree functions, specific commands like increase, reduce, count, next, previous and inrange are implemented.
In the increase function, the count of the node corresponding to the inputted ID is increased by m. If the node isn't present, then it is inserted with m as its count. Finally, the count of the ID after addition is displayed.
In the reduce function, the count of the node corresponding to the inputted ID is decreased by m. If the count becomes less than or equal to 0 then the node is deleted from the tree and 0 is printed. If the node has a positive count, its count is displayed. If the node isn't present in the tree then 0 is displayed.
In the count function, the count of the ID is displayed if it is present in the red black tree, else 0 is displayed.
In the inrange function, total count of the IDs in between ID1 and ID2 inclusive is displayed.

In the next function, the count of the event with the lowest ID that is greater than the input ID is displayed. If such an ID isn't possible (for example, if the input ID corresponds to the right most node in the tree) then "0 0" should be displayed.
In the previous function, the count of the event with the greatest ID that is lesser than the input ID is displayed. If such an ID isn't possible (for example, if the input ID corresponds to the left most node in the tree) then "0 0" should be displayed.
Inrange, next and previous functions are implemented recursively.

In the main function of the bbst class, after initialization of red black tree using program-argument input from test file, commands are inputted and the corresponding output is displayed. Program terminates once the quit command is entered.

Summary of program structure:

1. Input is read line by line from input file (which is sent as a program argument), this input consists of IDs and counts. The input is given sorted in increasing order of IDs.
2. This input is fed to the initializeRBT method which with the help of the initializeRBTHelper method creates red black tree by finding the middle element of list and making it the root of the tree to be constructed, then recursively the same is done with its left and right halves. These halves in turn are the left and right subtrees of the middle element. Nodes at all levels expect the last non-sentinel one are colored black. These last non-sentinel level nodes are colored red.
3. Then commands (increase, reduce, count, inrange, next, previous) are inputted from the standard input stream and output is printed on the standard output stream.
4. The program terminates when quit command is entered.

3. Function Prototypes

I used 3 classes "bbst.java", "Node.java" and "RedBlackTree.java" to implement the programming assignment.
Here's how the functions are defined in each class and a short description of how they work.

Node.java

The class Node mainly describes the structure of a node that is used in a Red Black tree.

The attributes are:

ID: This variable is of the primitive type int and it represents the ID number stored in the node.

count: This variable is of the primitive type int and it represents the count corresponding to the ID number stored in the node.

left, right and parent: These variables are of the type Node and store the left, right and parent pointers of the node.

color: This variable is of the type char and stores whether the node is a red node or a black node using characters 'b' or 'r'.

Node(int ID, int count)

Return type: Node

Parameters: int ID, int count

This class also defines a constructor which initializes ID and count to the input parameters, left, right and parent to null and color to black.

RedBlackTree.java

The class RedBlackTree is where all the operations of a Red Black tree have been defined. The operations of this class are performed on instances of class Node.

The functions defined in this class are:

void initializeRBT(int arr[][],int start,int end,int n)

Return type: void

Parameters : int arr[][],int start,int end,int n

(array sorted on IDs from which input is fed, start index and end indices of the array, number of elements in the array)

This function calls the helper function and creates the tree. The root's parent is set to null (sentinel) and its color is set to black.

Node initializeRBThelper(int arr[][], int start, int end, int level, int height)

Return type: Node

(the root of the tree is returned)

Parameters : int arr[][], int start, int end, int level, int height

(array sorted on IDs from which input is fed, current start, end and level, height of the tree based on the number of nodes in it)

This function creates the red black tree by finding the middle element of list and making it the root of the tree to be constructed, then recursively the same is done with its left and right halves. These halves in turn are the left and right subtrees of the middle element. Initially nodes at all levels expect the last (non-sentinel) level are colored black. The last level (non-sentinel) nodes are colored red. This is because while recursively creating the red black tree only the last level could be incomplete. By coloring this level red I ensure that black height will be preserved for all nodes and red black tree properties will be guaranteed. The last (sentinel) level containing sentinel nodes is colored black. The height of the corresponding tree would be log to the base 2 of the nodes in it. This is because red black tree is a balanced binary tree.

void insert(int ID, int count)

Return type: void

Parameters : int ID, int count

(ID and count of the node to be inserted)

This function inserts key/value pair at appropriate position by comparing existing RB-nodes without guarantee of balance (like BST).

void insertFixer(Node z)

Return type: void

Parameters : Node z

(Node for which balance of existing RB-tree is to be corrected)

This function corrects RB-tree color as well as height balance by perfoming color swaps and rotations.

void delete(Node z)

Return type: void

Parameters : Node z

(Node to be deleted)

This function deletes the node from RB-tree without guaranteeing balance (like BST) or returns without any operation.

void deleteFixer(Node z)

Return type: void

Parameters : Node z

(Node for which balance of existing RB-tree is to be corrected)

This function corrects RB-tree color as well as height balance by performing color swaps and rotations.

void leftRotate(Node x)

Return type: void

Parameters : Node x

(Node around which rotation is to be performed)

This function performs left rotation around pivot to correct black height property and ensure height balance.

void rightRotate(Node x)

Return type: void

Parameters : Node x

(Node around which rotation is to be performed)

This function performs right rotation around pivot to correct black height property and ensure height balance.

void transplant(Node u, Node v)

Return type: void

Parameters : Node u, Node v

This function replaces one subtree as a child of its parent with another subtree. In case of u and v, node u's parent becomes node v's parent and u's parent ends up having v as its appropriate child.

Node[] successorPredecessor(Node root, Node[] presuc, int key)

Return type: array of Nodes

(returns array of nodes where the first value corresponds to previous and the second value corresponds to next)

Parameters : Node root, Node[] presuc, int key

(root of the tree, array of next of previous nodes, key for which next or previous is computed)

This function is a helper method to recursively compute the smallest value just greater than or the largest value just less than the key ID.

void next(int ID)

Return type: void

Parameters : int ID

(ID for which next's count and ID have to be displayed)

This function uses successorPredecessor function to print the ID and the count of the event with the lowest ID that is greater than the inputted ID or print "0 0", if there is no next ID.

void previous(int ID)

Return type: void

Parameters : int ID

(ID for which previous's count and ID have to be displayed)

This function uses successorPredecessor function to print the ID and the count of the event with the greatest ID that is lesser than the inputted ID or print "0 0", if there is no previous ID.

Node search(Node k, int x)

Return type: Node

(Node corresponding to the input key or sentinel if "key not found")

Parameters : Node k, int x

(Node starting from which comparison with x starts, till a match is found or sentinel is reached)

This function searches for the node corresponding to the inputted key by comparing key's value with that of the node and goes left if the key is lesser or right if the key is greater than the node's ID.

void increase(int ID, int m)

Return type: void

Parameters : int ID, int m

(ID for which count value has to be increased by m)

This function increases the count of the event ID by m. If the ID is not present, it inserts it by calling the insert function. It prints the count of the ID after addition.

void reduce(int ID, int m)

Return type: void

Parameters : int ID, int m

(ID for which count value has to be decreased by m)

This function decreases the count of the event ID by m. If the ID's count becomes less than or equal to 0, it removes the ID by calling search followed by delete function. It prints the count of the ID after deletion, or 0 if the ID is removed or is not present.

void count(int ID)

Return type: void

Parameters : int ID

(ID for which count value has to be displayed)

This function displays the count of the event ID or 0 if the ID is not present.

int inrangeHelper(Node root, int ID1, int ID2)

Return type: int

Parameters : Node root, int ID1, int ID2

This function is a helper function to recursively compute count of the nodes whose IDs lie between ID1 and ID2 inclusive.

void inRange(int ID1, int ID2)

Return type: void

Parameters : int ID1, int ID2

This function calls the helper function to recursively compute count of the nodes whose IDs lie between ID1 and ID2 inclusive.

5. Outputs

The following are the output screenshots after running bbst.java on thunder.cise.ufl.edu. The first one is with the input file test_100.txt. The output is first displayed on the screen and then redirected to ouput.txt as well. Diff command gives no difference between ouput.txt and out_100.txt.

The same process is repeated with test_1000000.txt and test_10000000.txt.

```
Alekhya@LAPTOP-KN8L9FEG /cygdrive/c/Users/Alekhya/Documents/Gupta_Alekhya
$ java bbst test_10000000.txt<commands.txt
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
```

thunder.cise.ufl.edu - PuTTY

```
login as: alekhya
Authorized Access only

_____________________________________________________________
UNAUTHORIZED ACCESS TO THIS DEVICE IS PROHIBITED.
You must have explicit permission to access this
device. All activities performed on this device
are logged. Any violations of access policy can
result in disciplinary action.

_____________________________________________________________
alekhya@thunder.cise.ufl.edu's password:
Last login: Tue Mar 22 16:47:08 2016 from ip70-171-34-152.ga.at.cox.net
thunderx:1% cd Gupta_Alekhya
thunderx:2% make clean
rm -rf *.class
thunderx:3% make
javac bbst.java RedBlackTree.java Node.java
thunderx:4% java bbst test_100.txt<commands.txt
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
thunderx:5% java bbst test_100.txt < commands.txt>output.txt
thunderx:6% diff -b out_100.txt output.txt
thunderx:7% java bbst test_1000000.txt < commands.txt > output.txt
output.txt: File exists.
thunderx:8% java bbst test_1000000.txt < commands.txt
104
54
```

thunder.cise.ufl.edu - PuTTY

```
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
thunderx:5% java bbst test_100.txt < commands.txt>output.txt
thunderx:6% diff -b out_100.txt output.txt
thunderx:7% java bbst test_1000000.txt < commands.txt > output.txt
output.txt: File exists.
thunderx:8% java bbst test_1000000.txt < commands.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
thunderx:9% java bbst test_1000000.txt < commands.txt>output1.txt
thunderx:10% diff -b out_1000000.txt output1.txt
thunderx:11%
```