COP 5536 Programming Project Report

*Event Counter using Red Black Tree*

-------------------------------------------------------------------------------------------------------------------------------

Project implementation done in JAVA.

java version "1.8.0\_65"

Java(TM) SE Runtime Environment (build 1.8.0\_65-b17)

Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)

**Compiler**: javac "1.8.0\_65"

**Program Structure:**

Project structure consists of 4 files –

* Node.java
* Color.java
* RBEventTree.java
* Bbst.java

Detailed Structure of each file as follows -

**Node.java**

Class for the nodes in a Red-black Tree.

Defines following Properties and Constructor of a Node –

|  |  |
| --- | --- |
| int theID | *ID of the event* |
| int count | *number of active events with theID* |
| Node parent | *Parent Node* |
| Node left | *Left child Node* |
| Node right | *Right child Node* |
| Color color | *Color of the Node* |

**Color.java**

Enum Color with values RED and BLACK

**RBEventTree.java**

Class for the Red black tree event counter which is an actual implementation of Red-black tree which supports various operations specified in the project description.

Function Prototypes for the event counter -



**bbst.java**

* This file contains Main() function of the project which expects one input argument which is input ‘file-name’.

**Program Execution –**

* Project submission contains above 4 java files and *makefile* which creates an executable with name *bbst*.
* Command line for program –
  + $ make - the will compile the project
  + $ java bbst file-name
    - file-name – input file to construct initial event counter RB tree located in the same directory
* When we enter above commands through command line, main function will accept file-name as command line argument. And will construct initial event counter RB tree by calling ‘*initializer’* function.
  + *void Initializer(int[] idArray, int[] countArray)*

This function takes two arguments- sorted array of events and sorted array of their counts and calls helper method *‘buildRBTree’* to construct the initial tree recursively.

* + Running time for this O(n).
* After initial tree building, program reads the commands from the standard input stream or also commands can be entered through file by using unix redirection sign (<)
  + $ java bbst file-name < commands.txt
* Program structure for different event counter operations that can be performed –

**Increase –**

* + - *void increase(int theID, int m) –*

This function increases event count by *m* and prints it,if *theID* exsits. Else it will add new event *theID* by method ‘*add’*.

* + - *void add(int theID, int m) –*

This will insert a new event node *theID* and count *m*.

After inserting new node, method *balanceNewEventInsert* will be called for adjusting the RB tree.

* + - *void balanceNewEventInsert(Node node)*

After inserting a node to the tree, this function maintains Red-black tree property by rotating and recoloring*.*

* + - Helper methods used by *balanceNewEventInsert –*

*void leftRotate(Node node)*

*void rightRotate(Node node)*

These two methods are used to fix the unbalanced tree by rotating it left or right. This two functions take constant time.

* + - Time Complexity – O(log n)

**Reduce –**

* + - *void reduce(int theID, int m)* –

This function reduces event count by *m* and prints it,if *theID* exsits. If count reduces to 0 or less than 0, it will delete event *theID* by helper method ‘*delete’*.

* + - *void delete(Node node)*

This method will delete *node* from the tree. After that helper method *balanceEventDelete* will be called for adjusting the RB tree after deleting the Node.

* + - *void balanceEventDelete (Node node)*

After deleting a node from the tree, this function maintains Red-black tree property by rotating and recoloring*.*

* + - Other helper methods used by *delete –*

*void RB\_transplant(Node u, Node v)*

*Node getMin(Node node) –* This method returns a node with the minimum ID in the tree rooted at *'node'*

* + - Helper methods used by *balanceEventDelete* *–*

*void leftRotate(Node node)*

*void rightRotate(Node node)*

These two methods are used to fix the unbalanced tree by rotating it left or right. This two functions take constant time.

* + - Time Complexity – O(log n)

**Count –**

* + - *void count(int theID) –*

Prints count of the event theID if exists else prints “0”.

* + - It uses helper method *search* to search event theID in the RB tree.
    - *Node search(int theID)*

Returns node with ID theID

* + - Time Complexity – O(log n)

**Inrange –**

* + - *void inrange(int ID1, int ID2) –*

Print the total count of ids between ID1 and ID2

* + - It uses helper method *inrangecalculator* to find the required count.
    - *void inrangecalculator(int ID1, int ID2)*

Recursively calculates total count of ids between ID1 and ID2

* + - Time Complexity – O(log n+s) where s is the number of

IDs in the range.

**Next –**

* + - *void next(int theID) –*

Print the lowest ID event and its count which is greater than theID, if exists else prints “0 0”

* + - It uses helper method *findNext* which returns the required node.
    - *Node findNext(int theID)*

Returns lowest ID event node and its count which is greater than theID.

* + - Time Complexity – O(log n)

**Previous –**

* + - *void* prev *(int theID) –*

Print the greatest ID event and its count which is less than *theID* if exists else prints “0 0”

* + - It uses helper method *findPrev* which returns the required node.
    - *Node findPrev(int theID)*

Returns greatest ID event node and its count which is less than *theID.*

* + - Time Complexity – O(log n)

**Note:** For running huge input file (order of ~ 1GB), we need to increase JVM heap size to 8GB. By using following command –

$ java –Xmx8g bbst test\_100000000.txt < commands.txt >myout\_100000000.txt

**References –**

* Introduction to Algorithms, 3rd Edition

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.