COP 5536 Summer 2016

Programming Project

Due Date: July 18th, 2016, 11:59 P.M.

Submission Website: **Canvas**

1. General

1. Problem description

You are to implement an event counter using AVL 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. Initially, your program must build AVL tree from a sorted list of *n* events (i.e., *n* pairs (*ID*, *count*) in ascending order of *ID*) in O(*n*) time. Your counter should support the following operations in the specified time complexity. You are not allowed to use TreeMap in Java and std::map in C++.

Command	Description	Time Complexity
Increase(ID,m)	Increase the *count* of the event *ID* by *m*. If *ID* is not present, insert it. Print the *count* of *ID* after the addition.	O (log n)
Reduce(ID, m)	Decrease the *count* of *ID* by *m*. If *ID*'s *count* becomes less than or equal to *0*, remove *ID* from the counter. Print the *count* of *ID* after the deletion, or 0 if *ID* is removed or not present.	O (log n)
Next(ID)	Print ID and count of the event with lowest ID that is greater than ID. Print "0 0" if there is no next ID.	O (log n)
Count(ID)	Print the count *of ID*. If not present print 0.	O (log n)
Previous(ID)	Print ID and count of the event with greatest ID that is less than ID. Print "0 0" if there is no previous ID.	O (log n)
InRange (ID_1 ,ID_2)	Print the total count for IDs between ID_1 and ID_2 inclusively. Note $ID_1 \leq ID_2$.	O(log n + s), where s is the number of ID in this range.

2. Input/output Requirements

You may implement this assignment in Java or C++. Your program must be compliable and runnable on the Thunder CISE server using gcc/g++ or standard JDK. You may access the server using Telnet or SSH client on thunder.cise.ufl.edu. You must write a makefile document which creates an executable. The names of your executable must be bbst.
Your program has to support redirected input from a file "file-name" which contains the initial sorted list. The command line for this mode is as follows for C++ and Java respectively:

```
$bbst file-name
$java bbst file-name
```

Input format

n
ID_1 $count_1$
ID_2 $count_2$
...
ID_n $count_n$

Assume that $ID_i < ID_{i+1}$ where ID_i and $count_i$ are positive integers and the total count fits in 4-byte integer limits.

Interactive part:

Read the commands from the standard input stream and print the output to the standard output stream. Use the command specifications described in part 1 with all lower cases. The command and the arguments are separated by a space, not parenthesis or commas (i.e. “inrange 3 5” instead of “InRange(3, 5)”). At the end of each command, there will be an EOL character. For each command, print the specified output in the table. Use one space if more than one numbers are printed. Print an EOL character at the end of each command. To exit from the program, use “quit” command.

3. Submission

The following contents are required for submission:

1.Makefile: Your make file must be directly under the zip folder. No nested directories.

2. Source Program: Provide comments.

3. REPORT: The report must be in PDF format. State what compiler you use.
Present function prototypes showing the structure of your programs. Include the structure of your program. List of function prototypes is not enough.

To submit, please compress all your files together using a zip utility and submit to the Canvas system. You should look for Assignment Project for the submission. Your submission should be named LastName_FirstName.zip.

Please make sure the name you provided is the same as the same that appears on the Sakai system. Please do not submit directly to a TA. All email submission will be ignored without further notification. Please note that the due day is a hard deadline. No late submission will be allowed. Any submission after the deadline will not be accepted.

4. Grading Policy

Grading will be based on the correctness and efficiency of algorithms. Below are some details of the grading policy.
Correct and efficient implementation and execution: 60%.
Makefile and readability(comments): 10%
Report: 30%
Note: If you do not follow the input/output or submission requirements above, 25% of your score will be deduced. In addition, we may ask you to demonstrate your projects.
If you have any question, please contact Kiana Alikhademi at kalikhademi@cise.ufl.edu.