Project 4: SMM Priority Queue

Preliminary specifications

Project start: Thursday, Nov. 19, 2015
Early submission: Monday, Nov. 30, 2015, 11:57pm.
Projected score will be available **within** 2 **business** day.
Project due date: **Monday, December 7, 2015, 11:57pm**

Input: Dated items with variable priorities
Output: Protocol of the results of the transactions processed, and
a report describing your data structure for maintaining min and max.
The report needs to disclose all internet resources utilized.

Please read the description very carefully. You will lose valuable time and effort if you miss some of the requirements.

We provide an optional project skeleton; you will choose your own data structures

You must use stdin and stdout, also you must create a file called Project4.java with a main method in it.

The recommended data structure is the symmetric min-max heap. Alternatively, you can run two coordinated heaps, one a min, the other a max heap. Coordinating those two heaps, however, may take a **significantly** greater effort than working with a min-max heap.

Description

You run a streaming music service; subscribers listen to the songs you play. When a subscriber connects, via the web, you display the currently 3 most popular songs s. Popularity p_s is measurd in terms of the number of times N_s it has played, the length of time the song is part of your service, and the number L_s of *likes*, the number of times users have pushed the like-button on your web page. The exact formula for popularity is

$$p_s = N_s + 2L_s$$

As the administrator of the web site, you have to periodically buy and add new songs to your collection. If some songs are no longer listened to by your subscribers, and are low in popularity, you want to remove them from the collection to improve the quality of the service.

You read a stream of data, comprising listening requests, likes, requests to show the top 3, acquisitions, purges. You output a protocol that chronicles the state of your repository. All transactions are dated by a time stamp, an integer. The initial data in your repository has date 1. Transactions are by increasing date.

Songs age. Every now and then your marketing manager tells you that a specific song is listened to less frequently and issues an adjustment transaction that lowers the popularity of a given song. An example is given in the next section.

Input commands and actions

d S ΔN ΔL	At date d, song S has been listened to additional ΔN times and liked ΔL times. You update N_s and L_s adding the respective quantities. Note that ΔN and ΔL can be negative, indicating a drop in popularity. In that case you subtract, but N_s and L_s must not become negative. Update the popularity p_s according to the formula $p_s = N_s + 2L_s$ where N_s and L_s are new values.
d T3	At date d, report the 3 top songs with the highest popularity. Order songs of equal popularity by acquisition date, with older songs listed before newer ones.
d B	At date d, buy a new song with initial N=20 and L=20. The song gets the title Sd, where d is the date
d X n	At date d, delete n songs with the lowest priority. If two songs S_i and S_k have equal popularity but only one is to be deleted, you delete the older one. That is, with d1 and d2 the respective acquisition dates, S_i is deleted if d1<d2, and S_k is deleted when d2<d1.
d end	Print the highest and lowest popularity, then stop the program.

Example: The following transactions occur, with consequences periodically explained.

1 B	The song you buy is called S1. N and L are initialized to 20 each, giving S1 popularity 60. The acquisition date of the song is d=1
3 S1 4 7	S1 now has N=24 and L=27; d remains fixed, equal to 1
4 B	You buy S4, N=20, L=20, d=4
5 B	You buy S5, N=20, L=20, d=5
6 S5 8 12	S5 now has N=28, L=32, d=5
10 B	You buy S10, N=20, L=20, d=10
20 X 1	The lowest priority song is to be deleted. You have these songs and their priorities: S1 (78), S4 (60), S5 (92), S10 (60). S4 and S10 have equal and lowest priority, but S4 is older, so it is deleted. The remaining songs in your inventory are S1, S5 and S10.
51 S1 8 8	S1 now has N=32 and L=35
60 S1 -4 0	S1 now has N=28 and L=35
63 S5 2 3	S5 now has N=30, L=35
112 B	You buy song S112 with N=20 and L=20.
115 T3	You have the following songs and their popularity: S1 (98), S5 (100), S10 (60), S112 (60). You report these songs and their popularity: S5(100), S1 (98), S10 (60). Note that the items are ordered by decreasing popularity. Ties in popularity are broken by the date as described before. The output is on 3 lines in the format described below.
120 end	Min-max popularities are 100 and 60. End of the program.

Output and format

end: When the program is to stop, you print the two popularities as "min <n>, max <m>" where <n> and <m> are the lowest and highest popularities, both integers; e.g., "`min 120, max 355`". You should have no leading or trailing blanks, and a single blank between each field.

B: Buying: Print "S<d>: N=<n>, L=<l>, pop=<p>" where <d> is the acquisition date, <n> the times played, <l> the likes, and <p> the resulting popularity. E.g., "`S132: N=20, L=20, pop=60`" You should have no leading or trailing blanks, and a single blank between each field.

S: update: After updating N and L print the same information as B does, in the same format. Output the number of comparisons needed to adjust the data structure.

X: Delete: print "S<d> deleted, pop <p>", S<d> the song name, <p> the popularity; e.g. "`S15 deleted, pop=24`"

T: report top 3: Print the top 3, each in a single line, as "<r>: S<d>, pop=<p>", where <r> is 1 2 or 3, S<d> the song name, and <p> the popularity. E.g., "`1: S213, pop=660`"

Task 1 – Implement the reader

Each input transaction is on a single line. The parts of the transaction are separated by blanks. There could be leading and/or trailing blanks. Transactions have unique dates and arrive by increasing date in the input.

Task 2 – Store the song data

A song Sd has the date d, has been listened to N times and is liked L times. You need this information to determine the song's popularity and when to delete it. You will also need indexing structures that give you the required performance, and the associations between song record and indexing construct(s) should probably be bi-directional.

Task 3 – Build the infrastructure to carry out the transactions at the required efficiency

In this task, you build the supporting data structures and algorithms. Operations you need and required performance are as follows, with n the number of songs in the inventory:

addSong(…) all required changes and updates should be in O(log(n))

deleteSong(…) O(log(n))

updateSong(…) changes N and L of an individual song; O(log(n))

popular (…) all three popular songs in O(1)

minMax(…) O(1) – only reporting the maximum and minimum values.

Notes

Input errors can happen. Regardless of the type, you issue an error message and terminate the run. Any error message has a required prefix and optional additional text of your choosing.

Check for syntactic mistakes such as illegal characters or ill formed transactions. If there is one, issue the message “Error input syntax line <n>”. <n> is the line number (in the input not the date). You may add more information following <n>.

Check for semantic errors, such as duplicate dates, dates that do not follow the increasing order, play and like numbers for non-existent songs, etc. For all those events you issue an error message that starts out “Error semantics date <n>”

The maximum size of the repository is 500 songs, the minimum size is 3. When the maximum is exceeded you issue “Error max size exceeded date <n>”. When the number of songs is below the minimum you issue “Error min size violated”. For the first 5 transactions fewer than three songs are ok. Thereafter, the minimum size is enforced.

Data Structures

You need the performance of a heap. Sadly, heaps either track maxima or minima, but not both. But wait: you could run two heaps. Or you hit google scholar.

If you run a min heap and a max heap in parallel, each item appears once in each heap, but in different locations. To coordinate, you would link the two occurrences so that each can find the other. This means that every time an entry is moved or swapped, the affected items must have their cross-reference pointers adjusted. Not impossible, but full of opportunities to make mistakes.

There are variations of min-max heaps. Data structures that track both min and max. The description of one such is inked from the course web page cs.purdue.edu/homes/cmh/251. Scroll to the end and you will find it.

When updating the popularity of a song, you have to find it first. Since songs come and go, you may want to use a balanced search tree to find the song and its N and L as well as its position in the heap queue.

You may use code from the internet provided you give full reference to its origin. Note well that it is your responsibility to ensure that the code works. If you download and submit buggy code don’t be surprised when you lose points.

Example Files

Check “testcases” directory.

Grading:

Tests	Points
Program compiles and run	10; 5 if warnings occur
Coding standard	10
Passing all test cases	80

Details

Program Compiled and Run: 10 pts

1. We can compile your program and run it successfully, according to our requirements. (If your code cannot compiled by our script you lose 10 points).
2. We care about warnings. You may lose points if warnings are raised even though your code compiles and runs.

Coding Standard: 10pts

1. Your code should be well structured.
2. Rule of the thumb: TA can understand any method in less than 10 seconds.
3. Suggestions:
   a). Add Comments. Usually you will have the same amount of comments as code.
   b). Friendly Variable Names.
   c). Lots of small methods rather than several large methods.
   d). Indentation. You will lose all 10 points if your code is not well indented.

Passing All Test Cases: 80pts

All test cases are equally weighted for this project. You either get points for passing a test case or not.
No partial credits.
You are responsible for the robustness of the program. Passing only the provided vanilla test cases may result in very low scores.

Early Submission:

If you submit your project before “early submission” deadline, you will receive a projected score before the real deadline. Your program will be tested with exact same test cases for the final judgment. But you are allowed to resubmit if you want to improve the score.

After the projected score shows up in Blackboard check Piazza. We will post students’ common mistakes to help you improve your project.

We will NOT release test cases before the real deadline.

Submission Instructions:

Your code must be compiled on our data machine(data.cs.purdue.edu) by “javac *.java” command.

You must create a directory “project4”. You must Put all your .java files under “project4” directly. If you are using Eclipse or other IDE, make sure you put .java files under project4 directly. Failing to do so will result in 10 points off.

Your file structure should looks like the following:

```
project4:
  |- all java files(No extra folder like "src")
```

Note:

1. you should not literally copy the following commands.
2. Replace yourLogin with your login ID like “zhan1015”.

To resubmit, you just need to retype the following commands and type “yes” after the third command.

```
ssh yourLogin@data.cs.purdue.edu
cd directoryContainsProject4
turnin -v -c cs251 -p project4 project4
```

After the third command is executed, the system will give you some feedback about which files and folders have been submitted. If you resubmit a project, the previously submitted files will be overwritten.