CoE 164

Computing Platforms

Software Exercise 02

Academic Period: 2nd Semester AY 2021-2022
Workload: 6 hours
Synopsis: Website Priority Queueing
Submission Platform: UVLe Submission Bin

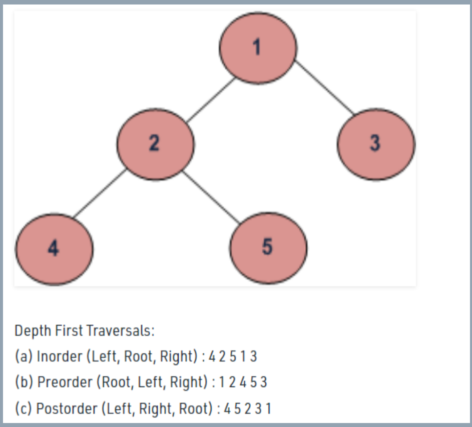
Introduction

As a fresh graduate, you opted to work at a startup company which provides IT solutions to small and large-scale businesses. One project of this company is to handle requests coming from multiple client websites which will be processed by a back-end mainframe system. The mainframe system runs faster compared to normal computers; however, it can only process a limited number of requests simultaneously. Since the company you're working for could not afford to buy licenses for queuing platforms such as IBM ActiveMQ, you were asked to create a robust and efficient queuing system that can temporarily store the requests to be picked up by the mainframe once it has free processing capacity. You investigated possible solutions from your previous collegiate computer engineering courses and remembered something about binary trees and heaps.

The queuing system should be able to handle the following commands: 1) adding of jobs, 2) batch execution of jobs, 3) printing of the jobs and their details from the queue, and 4) printing of already executed jobs. When the system is started, it can optionally have an initial amount of jobs in the queue. Additionally, it works under a priority system - a job of higher priority should appear at the front over a lower one, which in turn will be executed first. Finally, it can only have at most 30 jobs in it, so it will refuse any additions to it if it is already full.

Each job in the system is to be entered in a single line. It has a priority number, name, and creation date, like `10 run_invoice_reports 2022-04-26`, all separated by a single space. When comparing two jobs, the one that has a *greater priority number* has the higher priority. If they have the same priority numbers, the one with the *earliest creation date* has the higher priority. If they also have the same creation date, then the one whose job name *appears earlier alphabetically* has the higher priority.

When jobs are added, the queue will handle the priority ordering depending on the rules set above. The system can only add up to 5 jobs at the same time. If the queue is already full, the system will refuse to add any additional jobs and will record the amount of jobs that were "dropped" in this manner.


1
2
3
4
5
Depth First Traversals:
(a) Inorder (Left, Root, Right) : 4 2 5 1 3
(b) Preorder (Root, Left, Right) : 1 2 4 5 3
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

When a batch execution is requested, the queue will pop the frontmost jobs in it starting from the one with the highest priority. These jobs shall be saved in a history log ordered by execution date. If the number of jobs to be executed is greater than the queue size, the system will pop all the available jobs in the queue.

The system can print the contents of the queue as if it is a binary tree. It should be able to print the contents in pre-, in-, and post-order traversals. A short graphic for reference to these traversals is shown on this page.

The system can also print a history of already executed jobs in descending execution date. In other words, the one with the highest priority in a batch execution appears last in this list.

Now your queuing system is complete and ready for client demo!

Input

The input to the program starts with a number J_{init} indicating the number of jobs to be initially loaded into the queuing system. Each of the next J_{init} lines contain a job description of the format $prio\ jobDesc\ createDate$. After these lines, there will be a number C indicating the number of commands that will be sent to the queuing system. Each of the next C blocks of lines contain a command of the format $cmd\ input$. The supported commands are as follows.

- add_jobs J_{add}
  - Add J_{add} number of jobs into the queue. The next J_{add} lines contain a job description of the format $prio\ jobDesc\ createDate$.
- exec_jobs J_{exec}
  - Execute J_{exec} number of jobs from the queue from highest priority level
- print_queued_jobs Q_{jobs}
  - Print all of the jobs in the queue using mode Q_{jobs}. $Q_{jobs} \in \{pre,\ in,\ post\}$
- print_exec_hist
  - Print all of the executed jobs from the most recent execution.

Output

The output should consist of $C + 1$ blocks of lines. The first line should contain a string "Loaded J_{init} job(s) into queue". The next C blocks will contain the output of each corresponding command. For each command c_i, the output in the corresponding c_i block is as follows:

- add_jobs J_{add}
  - "[ADD] J_{add} job(s) and refused Q_{drp} job(s)"
- exec_jobs J_{exec}
  - "[EXEC] J_{exec} job(s)"
- print_queued_jobs Q_{jobs}
  - First line contains "[PRINT] Q_{jobs} job(s) in queue (Morder)".
  - Next Q_{jobs} lines contain each job in the queue of the format $prio\ jobDesc\ createDate$
- print_exec_hist
  - First line contains "[HIST] H executed job(s)"
  - Next H lines contain each executed job of the format $jobDesc\ createDate$

Example

Input

```
1
3
1 run_invoice_reports 2022-04-26
3 examine_sales_note 2022-04-30
4 download_journal_comment 2022-04-30
7
add_jobs 2
1 run_invoice_reports 2022-04-28
2 check_financial_report 2022-04-27
print_queued_jobs pre
print_queued_jobs in
print_queued_jobs post
exec_jobs 3
print_queued_jobs post
print_exec_hist
```

Output

```
Loaded 3 job(s) into queue
[ADD] 2 job(s) and refused 0 job(s)
[PRINT] 5 job(s) in queue (preorder)
4 download_journal_comment 2022-04-30
```

```
3 examine_sales_note 2022-04-30
1 run_invoice_reports 2022-04-28
2 check_financial_report 2022-04-27
1 run_invoice_reports 2022-04-26
[PRINT] 5 job(s) in queue (inorder)
1 run_invoice_reports 2022-04-28
3 examine_sales_note 2022-04-30
2 check_financial_report 2022-04-27
4 download_journal_comment 2022-04-30
1 run_invoice_reports 2022-04-26
[PRINT] 5 job(s) in queue (postorder)
1 run_invoice_reports 2022-04-28
2 check_financial_report 2022-04-27
3 examine_sales_note 2022-04-30
1 run_invoice_reports 2022-04-26
4 download_journal_comment 2022-04-30
[EXEC] 3 job(s)
[PRINT] 2 job(s) in queue (postorder)
1 run_invoice_reports 2022-04-28
1 run_invoice_reports 2022-04-26
[HIST] 3 executed job(s)
check_financial_report 2022-04-27
examine_sales_note 2022-04-30
download_journal_comment 2022-04-30
```

Additional Description/Requirements

Your program can only support the following limits:

$0 \leq J_{init} \leq 100$
$1 \leq prio \leq 100$
$createDate$ is in ISO 8601 format (i.e. YYYY-MM-DD)
$jobDesc$ only contains characters from the ASCII set and does not contain any control characters or whitespace
$1 \leq J_{add} \leq 5$
$1 \leq J_{exec} \leq 25$
$M \in$ {pre, in, post}
$cmd \in$ {add_jobs, exec_jobs, print_queued_jobs, print_exec_hist}

You can assume that all of the inputs are well-formed and within the above constraints, and you are not required to handle any errors arising from them. After all, you are pressed for deadlines!

Since your company cannot afford purchasing enterprise-grade queueing platforms, much less have a patent liability, your program should **not** use any library that implements some sort of priority queueing system. Additionally, for the sake of internal use and for it to function offline, your program should also **not** use any libraries that need to be downloaded off the internet (e.g. libraries that have to be downloaded from pip (for Python) or npm (for Javascript) are prohibited).

Please put a readme.txt for instructions on how to install the necessary platforms to run the code and the running guide itself. **Non-runnable codes will be graded as-is.**

Grading Rubric

10% Program initialization - able to accept an initial list of jobs
10% Add jobs - able insert jobs into queue
40% Batch execute jobs - able to batch execute jobs and record those to the history list
10% Print in-order - correct printing of queued jobs in-order
10% Print pre-order - correct printing of queued jobs pre-order
10% Print post-order - correct printing of queued jobs post-order
10% Print history - correct printing of executed job history