Project 2 - Instagram Feed
CmpE 250, Data Structures and Algorithms, Fall 2025

SAs: Hasancan Keleş, Umut Şendağ
TAs: İrem Urhan, Kutay Altıntaş
Instructor: Atay Özgövde

Due: 00/00/2023, 23:55 Strict

1 Introduction

You found a mysterious device in your basement and it took you back in time to early 2010. After learning about your time travel, Kevin Systrom, developer of Instagram, hired you to create a Feed Manager for his new idea Instagram. You are expected to develop a Java program which keeps track of users, their posts and likes, and users' feeds and creates a log about these operations.

2 Application's Capabilities

Kevin wants the application to be able to create users and posts, enable the users to interact with the posts and other users, suggest new posts based on the interaction that post receive.

2.1 How Should It Work

- The application should be able to create users with an ID.
- A user should be able to follow another user or unfollow an already followed user.
- All users should be able to create posts with their ID and content inside it.
- A user should easily be able to see another user's all posts and like or unlike them.
- Generate a feed for a user according to post likes and the user's seen posts.
- Sort a user's posts with respect to its likes.

2.2 Some Warnings

- There cannot be multiple users with same IDs.
- There cannot be multiple posts with same IDs.
- User IDs, post IDs and post contents only include alphanumeric characters.
- The feed for a user should be arranged in a way that the post with most likes should show up first. If two or more posts have the same likes, the post with the biggest lexicographical ID should show up first. More information is provided in the subsection below.
- A user's feed should only consist of that user's followed users' posts.
- Kevin wants the application to work swiftly so finding maximum liked posts should be handled fast.
- When we want to sort a user's posts according to their likes, if multiple posts have the same number of likes the post with the biggest lexicographical ID should be first.

2.2.1 Process of Generating Feed

1. Generation of feed will also supply you a number which tells you how many posts should be generated for that user.
2. While generating feed you should choose the most liked posts among all posts. However the feed of a user shouldn't include posts from itself and posts from unfollowed users. If the user already saw a post before the feed generation it shouldn't also show up in the new feed.
3. Finding the post with maximum likes should be handled fast so use corresponding methods.
4. When the feed is generated it is not shown to the user, so for example, if a user's feed is generated two times in a row with same number, feeds would be the same.
5. You should handle the cases where the given user ID is not present.

3 I/O Files

The input will consist of a file of instructions that should be executed. You should create a log in which you record the events that are happened during execution. The format of the log will be explained in detail later. You can assume the instructions in the input will be syntactically correct, so you don't have to check whether the input instruction is in correct format or not.

Example:
create_user user1
follow_user user2 user3
see_post user4 post1
see_all_posts_from_user user5 user6 1 1 0 0 1
generate_feed user7 5

For more information, you can refer to the additional input-output files. There is an additional .txt file named time.txt which contains execution time for each case. Further clarification about grading and valid time intervals will be announced on short notice.

4 Instructions

4.1 User Creation

A new user should be created with its userId. The input format is as:

Structure: create_user userId
Example: create_user user1

If a user with the given Id is already created you shouldn't create a new one or update the existing user. You should log:

Structure: Some error occurred in create_user.

When this operation successfully happens you should log a line in the following format.

Structure: Created user with Id <UserId>.
Example: Created user with Id user1.

4.2 Following/Unfollowing a User

A user with the Id <userId1> should be able to follow or unfollow another user with the Id <userId2>. The input format is:

Structure: follow/unfollow_user <userId1> <userId2>

Example: follow_user user1 user2
Example: unfollow_user user1 user2

If any of the users doesn't exist you should log:

Structure: Some error occurred in follow_user.
Structure: Some error occurred in unfollow_user.

If <userId1> and <userId2> are the same log do nothing and log this:

Structure: Some error occurred in follow_user.
Structure: Some error occurred in unfollow_user.

If the user with Id <userId1> is already following or unfollowing the user with Id <userId2> you shouldn't change anything and log:

Structure: Some error occurred in follow_user.
Structure: Some error occurred in unfollow_user.

If the operation is done succesfully:

Structure: <userId1> followed/unfollowed <userId2>.
Example: user1 followed user2.
Example: user1 unfollowed user2.

4.3 Creating a Post

A user with the Id <userId> should be able to create a post with <postId> and <content>. The format is:

Structure: create_post <userId> <postId> <content>
Example: create_post user1 post1 Hello

If there is no user with Id userId you should log:

Structure: Some error occurred in create_post.

Then if another post with Id <postId> is already present. You shouldn't change anything and log:

Structure: Some error occurred in create_post.

If operation is done successfully you should log:

Structure: <userId> created a post with Id <postId>.
Example: user1 created a post with Id post1.

4.4 Seeing a Post

A user with <userId> sees the post with Id <postId>. The format is

Structure: see_post <userId> <postId>
Example: see_post user1 post1

If the user or the post doesn't exist:

Structure: Some error occurred in see_post.

After this operation the post shouldn't show up in this user's feed. Upon completion, log:

Structure: <userId> saw <postId>.
Example: user1 saw post1.

4.5 Seeing All Posts of A User

A user with Id <viewerId> can see a user with Id <viewedId>'s all posts. The format is:

Structure: see_all_posts_from_user <viewerId> <viewedId>
Example: see_all_posts_from_user user1 user2

If any of the users are not present you should log:

Structure: Some error occurred in see_all_posts_from_user.

Upon successful completion, you should log:

Structure: <viewerId> saw all posts of <viewedId>.
Example: user1 saw all posts of user2.

4.6 Pressing the Like Button

When the like button is pressed, if the post with <postId> is not already liked by the user with Id <userId> it should be liked; otherwise, it shouldn't be liked. Liking a post also counts as seeing a post. The format is:

Structure: toggle_like <userId> <postId>
Example: toggle_like user1 post1

If the user or the post doesn't exist you should log:

Structure: Some error occurred in toggle_like.

Upon successful completion you should log:

Structure: <userId> liked/unliked <postId>.
Example: user1 liked post1.
Example: user1 unliked post1.

4.7 Generating Feed

We should be able to generate a <num> post long feed for a user with Id <userId>. The posts that the user created and the posts that the user already seen shouldn't show up. Most liked post appears first. The format is:

Structure: generate_feed <userId> <num>
Example: generate_feed user1 5

If the user doesn't exist log:

Structure: Some error occurred in generate_feed.

You should log the feed in this format:

Structure: Feed for <userId>:
Post Id: <postId>, Author: <authorId>, Likes: <likeCount>
Example: Feed for user1:
Post ID: post1, Author: user2, Likes: 0
Post ID: post2, Author: user2, Likes: 0

If there are not enough posts in the feed to satisfy <num>, you should log the ones in

the feed and then log this:

Structure: No more posts available for <userId>.
Example: No more posts available for user1.

An example feed which <num> is not satisfied.

Feed for user1:
Post ID: post1, Author: user2, Likes: 0
Post ID: post3, Author: user2, Likes: 0
No more posts available for user1.

4.8 Scrolling Through Feed

When this instruction is given user with Id <userId> should scroll through <num> number of posts in its feed. Note that num is always a positive integer. Next to the <num> there are <num> number of 1s or 0s. If you encounter a 1 it means the user clicked the like button for that post, didn't if 0. Format and clarification below:

Structure: scroll_through_feed <userId> <num> <like1> <like2> .... <likenum>
Example: scroll_through_feed user1 3 0 0 1

If the user doesn't exist log:

Structure: Some error occurred in scroll_through_feed.

In the example instruction 3 means the user scrolled through 3 posts. 0 0 1 means the user clicked the like button in only third post. Scrolling through a post means also that the post is seen by the user. It is guaranteed that there will be <num> number of 1s or 0s after <num>.

When the instruction is called, you should first log this:

Structure: <userId> is scrolling through feed.
Example: user1 is scrolling through feed.

Then for all posts in feed, if the user didn't press the like button you should log:

Structure: <userId> saw <postId> while scrolling.
Example: user1 saw post1 while scrolling.

If user pressed the like button you should log:

Structure: <userId> saw <postId> while scrolling and clicked the like button.

Example: user1 saw post1 while scrolling and clicked the like button.

If there are not enough posts to satisfy num, yo should log this once:

Structure: No more posts in feed.

Example log for example, assume there are only two posts in user1's feed:

user1 is scrolling through feed.
user1 saw post1 while scrolling.
user1 saw post2 while scrolling and clicked the like button.
No more posts in feed.

4.9 Sorting Posts

When this is called you should sort a user's all posts, and log all the posts according to their likes from bigger to smaller. If two posts likes are equal lexicographically bigger one comes first.

Structure: sort_posts <userId>
Example: sort_posts user1

If the user doesn't exist log:

Structure: Some error occurred in sort_posts.

If there are no posts from that user you should only log this:

Structure: No posts from <userId>.

First you should log the following:

Structure: Sorting <userId>'s posts:

Then proceed as follows:

Structure: <post1>, Likes: <post1LikeCount>

<post2>, Likes: <post2LikeCount>
...
Example: post1, Likes: 1
post2, Likes 0

5 Submission

You will be submitting a zip file containing your code via Moodle. Put your .java files in the zip file. Keep the name of the main function "Main.java". Name the zip file in this format.

Structure: <studentnumber>.zip

Your code must be able to run correctly with these commands:

javac *.java
java Main <input_file> <output_file>

6 Grading

6.1 Types and Grading

There are 4 types of inputs, each having their specific operations and values in terms of grading.

6.1.1 Type 1: Operations about Users

Implement the methods to create users and make them follow/unfollow each other.

6.1.2 Type 2: Operations About Posts

Implement the methods to create posts and also by extending type 1 by making a user see or like another person's posts.

6.1.3 Type 3: Operations About Feed Generation

Extend type 1 and type 2 by creating a feed for users according to their posts and the likes that the posts got.

6.1.4 Type 4: Operations About Sorting

Implement the method to sort a user's posts by their likes.

6.2 Grade Distribution

You can see the grade distribution for each input type in Table 1:

Type	Small Case (%)	Large Case (%)
1	6	4
2	6	4
3	36	24
4	12	8

Table 1: Grade Distribution for Small and Large Cases

6.3 Constraint Specifications

Constraint specifications are given below in Table 2. Note that the 0s after the user numbers are not shown:

Input Type	Type 1 (S)	Type 1 (L)	Type 2 (S)	Type 2 (L)	Type 3 (S)	Type 3 (L)	Type 4 (S)	Type 4 (L)
num_create_user	600	500000	200	150000	100	2500	5	15
num_follow_user	300	300000	0	0	20	10000	0	0
num_unfollow_user	300	300000	0	0	20	10000	0	0
num_create_post	0	0	350	350000	150	200000	2500	400000
num_see_post	0	0	4000	550000	2000	250000	0	0
num_see_all_posts_from_user	0	0	200	200000	500	50000	0	0
num_toggle_like	0	0	150	150000	200	200000	1500	250000
num_generate_feed	0	0	0	0	50	10000	0	0
num_scroll_through_feed	0	0	0	0	1000	100000	0	0
num_sort_posts	0	0	0	0	0	0	5	15

Table 2: Maximum Values of Different Types (Small and Large Versions)

- **num_create_user:** The maximum number of create_user commands. It is also the max number of users. Keep in mind that user Id's are at most 8 characters
- **num_follow_user:** The maximum number of follow_user commmands.
- **num_unfollow_user:** The maximum number of unfollow_user commands.
- **num_create_post:** The maximum number of create_post commands. It is also the max number of posts. Keep in mind that postId's are at most 14 characters
- **num_see_post:** The maximum number of see_post commands.
- **num_see_all_posts_from_user:** The maximum number of see_all_posts_from_user commands.

- **num_toggle_like:** The maximum number of toggle_like commands.
- **num_generate_feed:** The maximum number of generate_feed commands.
- **num_scroll_through_feed:** The maximum number of scroll_through_feed commands
- **num_sort_posts:** The maximum number of sort_posts commands.

6.4 Time Constraints

Each test case has a time constraint according to this table:

Type	Small Case (seconds)	Large Case (seconds)
1	2	10
2	2	10
3	2	45
4	2	20

Table 3: Time Constraints for Small and Large Cases

This constraints are decided according to our codes execution time. These durations were measured on an 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz with 32GB of RAM. Keep this in mind while testing your code.

7 Warnings

- All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code. Any sign of cheating will be penalized by at least -100 points at first attempt and disciplinary action in case of recurrence.
- Make sure you document your code with necessary inline comments and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long. This is very important for partial grading.
- Make sure that the white-spaces in your output is correct. You can disregard the ones at the end of the line.
- Please use the discussion forum at moodle for your questions, and check if it is already answered before asking.