­

**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13-B**

# Lab 06: Test First Programming - II (Social Network Analysis)

**CLO-03:** Design and develop solutions based on Software Construction principles.  
**CLO-04:** Use modern tools such as Eclipse, NetBeans etc. for software construction.

**Date: 21st Oct 2024**

**Instructor: Dr. Mehvish Rashid  
Lab Engineer: Mr. Aftab Farooq**

**Submitted by:  
 Name: Ali Amar**

**CMS id: 426014**

**Introduction:**

Students will have hands-on experience of test-first programming. Given a set of specifications, you will write unit tests that check for compliance with the specifications, and then implement code that meets the specifications.

**Material:**

https://ocw.mit.edu/ans7870/6/6.005/s16/psets/ps1/

Lectures on LMS regarding designing Specifications

## Lab Tasks

Solve problem 3 of problem set 1 listed on the link. The goal of problem set is to build a toolbox of methods that can extract information from a set of tweets downloaded from Twitter.

### Test-First Programming:

1. Study the specification of the method carefully.
2. Write JUnit tests for the method according to the spec.
3. Implement the method according to the spec.
4. Revise your implementation and improve your test cases until your implementation passes all your tests.

### Task-1:

**Inferring a social network**

1. In this problem, you will test and implement the methods in *SocialNetwork.java .* The *guessFollowsGraph()* method creates a social network over the people who are mentioned in a list of tweets. The social network is an approximation to who is following whom on Twitter, based only on the evidence found in the tweets.  
     
   **Hint :** ***guessFollowsGraph()****:* This method analyzes a list of tweets and builds a social network where users are linked to those they "follow," inferred from the users they mention in their tweets. It creates a graph (Map) where keys are users and values are sets of mentioned users.

### Task-02:

The *influencers()* method returns a list of people sorted by their influence (total number of followers).

**Hint :** ***influencers()****:* This method returns a sorted list of users by their influence, measured by how many people follow them.

**Task-03**  
  
 a. Devise, document, and implement test cases for *guessFollowsGraph()* and *influencers()*, and put them in SocialNetworkTest.java . Be careful that your test cases for *guessFollowsGraph()* respect its underdetermined postcondition.

1. Implement *guessFollowsGraph()* and *influencers()* , and make sure your tests pass. For now, implement only the minimum required behaviour for *guessFollowsGraph()* , which infers that Ernie follows Bert if Ernie @-mentions Bert.

#### **Test Cases should be:**

1. **Empty List of Tweets**: Ensures that an empty list results in an empty graph.
2. **Tweets Without Mentions**: Verifies that tweets with no mentions do not add entries to the graph.
3. **Single Mention**: Tests whether a user who mentions someone is correctly added to the graph.
4. **Multiple Mentions**: Checks if multiple mentioned users are linked to the tweet author.
5. **Multiple Tweets from One User**: Ensures that repeated mentions from the same user are captured.
6. **Empty Graph for influencers ():** Verifies that no users yield an empty influencer list.
7. **Single User Without Followers**: Tests that a user without followers yields no influencers.
8. **Single Influencer**: Verifies correct identification of the only influencer.
9. **Multiple Influencers**: Tests for correct influencer ordering.
10. **Tied Influence**: Ensures equal influencers are handled correctly.

If you want to see your code work on a live sample of tweets, run Main.java . It will print the top 10 most-followed people according to the social network you generated. You can search for them on Twitter to see if their actual number of followers has a similar order.

### Solution

### SocialNetwrok.java:

**package** twitter;

**import** java.util.\*;

**public** **class** SocialNetwork {

**public** **static** Map<String, Set<String>> guessFollowsGraph(List<Tweet> tweets) {

Map<String, Set<String>> followsGraph = **new** HashMap<>();

**for** (Tweet tweet : tweets) {

String author = tweet.getAuthor().toLowerCase();

Set<String> mentionedUsers = Extract.*getMentionedUsers*(Collections.*singletonList*(tweet));

**if** (!followsGraph.containsKey(author)) {

followsGraph.put(author, **new** HashSet<>());

}

// Add the mentioned users to the author's set of people they follow

followsGraph.get(author).addAll(mentionedUsers);

}

**return** followsGraph;

}

**public** **static** List<String> influencers(Map<String, Set<String>> followsGraph) {

Map<String, Integer> followerCount = **new** HashMap<>();

// Count followers

**for** (Set<String> follows : followsGraph.values()) {

**for** (String followed : follows) {

followerCount.put(followed, followerCount.getOrDefault(followed, 0) + 1);

}

}

// Sort users by their number of followers

List<String> influencers = **new** ArrayList<>(followerCount.keySet());

influencers.sort((user1, user2) -> followerCount.get(user2) - followerCount.get(user1));

**return** influencers;

}

}

### SocialNetworkTest.java:

**package** twitter;

**import** **static** org.junit.Assert.\*;

**import** java.time.Instant;

**import** java.util.\*;

**import** org.junit.Test;

**public** **class** SocialNetworkTest {

@Test(expected = AssertionError.**class**)

**public** **void** testAssertionsEnabled() {

**assert** **false**;

}

@Test

**public** **void** testGuessFollowsGraphEmpty() {

Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(**new** ArrayList<>());

*assertTrue*("expected empty graph", followsGraph.isEmpty());

}

@Test

**public** **void** testGuessFollowsGraphNoMentions() {

Tweet tweet = **new** Tweet(1, "user1", "Hello world", Instant.*now*());

List<Tweet> tweets = Collections.*singletonList*(tweet);

Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);

*assertTrue*("expected user1 in graph", followsGraph.containsKey("user1"));

*assertTrue*("expected empty set for user1", followsGraph.get("user1").isEmpty());

}

@Test

**public** **void** testGuessFollowsGraphSingleMention() {

Tweet tweet = **new** Tweet(1, "user1", "Hello @user2", Instant.*now*());

List<Tweet> tweets = Collections.*singletonList*(tweet);

Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);

*assertTrue*("expected user1 in graph", followsGraph.containsKey("user1"));

*assertTrue*("expected user2 in set", followsGraph.get("user1").contains("user2"));

}

@Test

**public** **void** testGuessFollowsGraphMultipleMentions() {

Tweet tweet = **new** Tweet(1, "user1", "Hello @user2 @user3", Instant.*now*());

List<Tweet> tweets = Collections.*singletonList*(tweet);

Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);

*assertTrue*("expected user1 in graph", followsGraph.containsKey("user1"));

*assertTrue*("expected user2 in set", followsGraph.get("user1").contains("user2"));

*assertTrue*("expected user3 in set", followsGraph.get("user1").contains("user3"));

}

@Test

**public** **void** testGuessFollowsGraphMultipleTweets() {

Tweet tweet1 = **new** Tweet(1, "user1", "Hello @user2", Instant.*now*());

Tweet tweet2 = **new** Tweet(2, "user1", "Hi @user3", Instant.*now*());

List<Tweet> tweets = Arrays.*asList*(tweet1, tweet2);

Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);

*assertTrue*("expected user1 in graph", followsGraph.containsKey("user1"));

*assertTrue*("expected user2 in set", followsGraph.get("user1").contains("user2"));

*assertTrue*("expected user3 in set", followsGraph.get("user1").contains("user3"));

}

@Test

**public** **void** testInfluencersEmpty() {

Map<String, Set<String>> followsGraph = **new** HashMap<>();

List<String> influencers = SocialNetwork.*influencers*(followsGraph);

*assertTrue*("expected empty list", influencers.isEmpty());

}

@Test

**public** **void** testInfluencersSingleUserNoFollowers() {

Map<String, Set<String>> followsGraph = **new** HashMap<>();

followsGraph.put("user1", **new** HashSet<>());

List<String> influencers = SocialNetwork.*influencers*(followsGraph);

*assertTrue*("expected empty list", influencers.isEmpty());

}

@Test

**public** **void** testInfluencersSingleInfluencer() {

Map<String, Set<String>> followsGraph = **new** HashMap<>();

followsGraph.put("user1", **new** HashSet<>(Arrays.*asList*("user2")));

List<String> influencers = SocialNetwork.*influencers*(followsGraph);

*assertEquals*("expected single influencer", Collections.*singletonList*("user2"), influencers);

}

@Test

**public** **void** testInfluencersMultipleInfluencers() {

Map<String, Set<String>> followsGraph = **new** HashMap<>();

followsGraph.put("user1", **new** HashSet<>(Arrays.*asList*("user2", "user3")));

followsGraph.put("user4", **new** HashSet<>(Arrays.*asList*("user2")));

List<String> influencers = SocialNetwork.*influencers*(followsGraph);

*assertEquals*("expected influencers sorted by followers", Arrays.*asList*("user2", "user3"), influencers);

}

}

### Results:

### 

### 

### Link to GitHub:

### Deliverables:

In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to [aftab.farooq@seecs.edu.pk.](mailto:aftab.farooq@seecs.edu.pk.)