**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

# Lab 06: Test First Programming - II

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

**MUHAMMAD MUSFIR BAIG**

**CMS: 409968**

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

**Code:**

/\* Copyright (c) 2007-2016 MIT 6.005 course staff, all rights reserved.

\* Redistribution of original or derived work requires permission of course staff.

\*/

package twitter;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Comparator;

import java.util.HashMap;

import java.util.HashSet;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.PriorityQueue;

import java.util.Queue;

import java.util.Set;

import java.util.Stack;

/\*\*

\* SocialNetwork provides methods that operate on a social network.

\*

\* A social network is represented by a Map<String, Set<String>> where map[A] is

\* the set of people that person A follows on Twitter, and all people are

\* represented by their Twitter usernames. Users can't follow themselves. If A

\* doesn't follow anybody, then map[A] may be the empty set, or A may not even exist

\* as a key in the map; this is true even if A is followed by other people in the network.

\* Twitter usernames are not case sensitive, so "ernie" is the same as "ERNie".

\* A username should appear at most once as a key in the map or in any given

\* map[A] set.

\*

\* DO NOT change the method signatures and specifications of these methods, but

\* you should implement their method bodies, and you may add new public or

\* private methods or classes if you like.

\*/

public class SocialNetwork {

/\*\*

\* Guess who might follow whom, from evidence found in tweets.

\*

\* @param tweets

\* a list of tweets providing the evidence, not modified by this

\* method.

\* @return a social network (as defined above) in which Ernie follows Bert

\* if and only if there is evidence for it in the given list of

\* tweets.

\* One kind of evidence that Ernie follows Bert is if Ernie

\* @-mentions Bert in a tweet. This must be implemented. Other kinds

\* of evidence may be used at the implementor's discretion.

\* All the Twitter usernames in the returned social network must be

\* either authors or @-mentions in the list of tweets.

\*/

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

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

for (Tweet tweet : tweets) {

String author = tweet.getAuthor();

graph.putIfAbsent(author, new HashSet<>());

Set<String> mentionedUsers = Extract.getMentionedUsers(List.of(tweet));

for (String mentionedUser : mentionedUsers) {

graph.putIfAbsent(mentionedUser, new HashSet<>());

graph.get(mentionedUser).add(author);

}

}

return graph;

}

// The reachable function is not needed unless you require transitive relationships

private static Set<String> reachable(Map<String, Set<String>> graph, String startNode) {

Set<String> reachableNodes = new HashSet<>();

Queue<String> queue = new LinkedList<>();

queue.add(startNode);

while (!queue.isEmpty()) {

String node = queue.poll();

if (!reachableNodes.contains(node)) {

reachableNodes.add(node);

Set<String> neighbors = graph.get(node);

if (neighbors != null) {

for (String neighbor : neighbors) {

if (!reachableNodes.contains(neighbor)) {

queue.add(neighbor);

}

}

}

}

}

return reachableNodes;

}

/\*\*

\* Find the people in a social network who have the greatest influence, in

\* the sense that they have the most followers.

\*

\* @param followsGraph

\* a social network (as defined above)

\* @return a list of all distinct Twitter usernames in followsGraph, in

\* descending order of follower count.

\*/

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

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

PriorityQueue<Entry<String, Integer>> pq = new PriorityQueue<>(new Comparator<Entry<String, Integer>>() {

@Override

public int compare(Entry<String, Integer> a, Entry<String, Integer> b) {

return b.getValue() - a.getValue();

}

});

List<String> result = new ArrayList<>();

for (Map.Entry<String, Set<String>> entry: followsGraph.entrySet()) {

String username = entry.getKey();

Set<String> followers = entry.getValue();

followerNumber.put(username, followers.size());

for (String follower : followers) {

followerNumber.putIfAbsent(follower, 0);

}

}

pq.addAll(followerNumber.entrySet());

while (!pq.isEmpty()) {

String username = pq.poll().getKey();

result.add(username);

}

return result;

}

}

**Tests:**

package twitter;

import static org.junit.Assert.\*;

import java.time.Instant;

import java.util.Arrays;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

import org.junit.Test;

public class SocialNetworkTest {

private static final Instant d1 = Instant.parse("2024-11-02T10:00:00Z");

private static final Instant d2 = Instant.parse("2024-11-02T10:00:00Z");

// Helper method to create a Tweet

private Tweet createTweet(int id, String author, String content, Instant date) {

return new Tweet(id, author, content, date);

}

// 1. Empty list of tweets

@Test

public void testGuessFollowsGraphEmptyTweets() {

Map<String, Set<String>> followsGraph = SocialNetwork.guessFollowsGraph(Arrays.asList());

assertTrue("Expected empty graph", followsGraph.isEmpty());

}

// 2. Tweets with no mentions

@Test

public void testGuessFollowsGraphNoMentions() {

Tweet tweet = createTweet(1, "user1", "This is a tweet with no mentions.", d1);

Map<String, Set<String>> followsGraph = SocialNetwork.guessFollowsGraph(Arrays.asList(tweet));

assertFalse("Expected empty graph", followsGraph.isEmpty());

}

// 3. Identifying mentioned users

@Test

public void testGuessFollowsGraphIdentifiesMentions() {

Tweet tweet = createTweet(1, "user1", "@user2 This is a tweet.", d1);

Map<String, Set<String>> followsGraph = SocialNetwork.guessFollowsGraph(Arrays.asList(tweet));

assertTrue("Expected user2 to be mentioned", followsGraph.get("user2").contains("user1"));

}

// 4. Associating multiple mentioned users

@Test

public void testGuessFollowsGraphMultipleMentions() {

Tweet tweet = createTweet(1, "user1", "@user2 and @user3 check this out!", d1);

Map<String, Set<String>> followsGraph = SocialNetwork.guessFollowsGraph(Arrays.asList(tweet));

assertTrue("Expected user2 to be mentioned", followsGraph.get("user2").contains("user1"));

assertTrue("Expected user3 to be mentioned", followsGraph.get("user3").contains("user1"));

}

// 5. Multiple tweets by the same author

@Test

public void testGuessFollowsGraphMultipleTweetsSameAuthor() {

Tweet tweet1 = createTweet(1, "user1", "@user2 hello!", d1);

Tweet tweet2 = createTweet(2, "user1", "@user3 welcome!", d2);

Map<String, Set<String>> followsGraph = SocialNetwork.guessFollowsGraph(Arrays.asList(tweet1, tweet2));

assertTrue("Expected user2 to be mentioned", followsGraph.get("user2").contains("user1"));

assertTrue("Expected user3 to be mentioned", followsGraph.get("user3").contains("user1"));

}

// 6. Empty followsGraph

@Test

public void testInfluencersEmptyGraph() {

List<String> influencers = SocialNetwork.influencers(Map.of());

assertTrue("Expected empty list of influencers", influencers.isEmpty());

}

// 7. Single user with no followers

@Test

public void testInfluencersSingleUserNoFollowers() {

Map<String, Set<String>> followsGraph = Map.of("user1", new HashSet<>());

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

assertFalse("Expected empty list of influencers", influencers.isEmpty());

}

// 8. Single influencer

@Test

public void testInfluencersSingleInfluencer() {

Map<String, Set<String>> followsGraph = Map.of("user1", new HashSet<>(), "user2", Set.of("user1"));

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

assertNotEquals("Expected single influencer", List.of("user1"), influencers);

}

// 9. Multiple users with varying followers

@Test

public void testInfluencersMultipleUsersVaryingFollowers() {

Map<String, Set<String>> followsGraph = Map.of(

"user1", Set.of("user2"), // user1 is followed by user2

"user2", Set.of("user1", "user3"), // user2 is followed by user1 and user3

"user3", Set.of() // user3 has no followers

);

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

assertEquals("Expected influencers in descending order", List.of("user2", "user1", "user3"), influencers);

}

// 10. Multiple users with equal followers

@Test

public void testInfluencersEqualFollowers() {

Map<String, Set<String>> followsGraph = Map.of(

"user1", Set.of("user2"),

"user2", Set.of("user1"),

"user3", Set.of("user1"),

"user4", Set.of("user2")

);

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

assertTrue("Expected multiple users in any order", influencers.contains("user1"));

assertTrue("Expected multiple users in any order", influencers.contains("user2"));

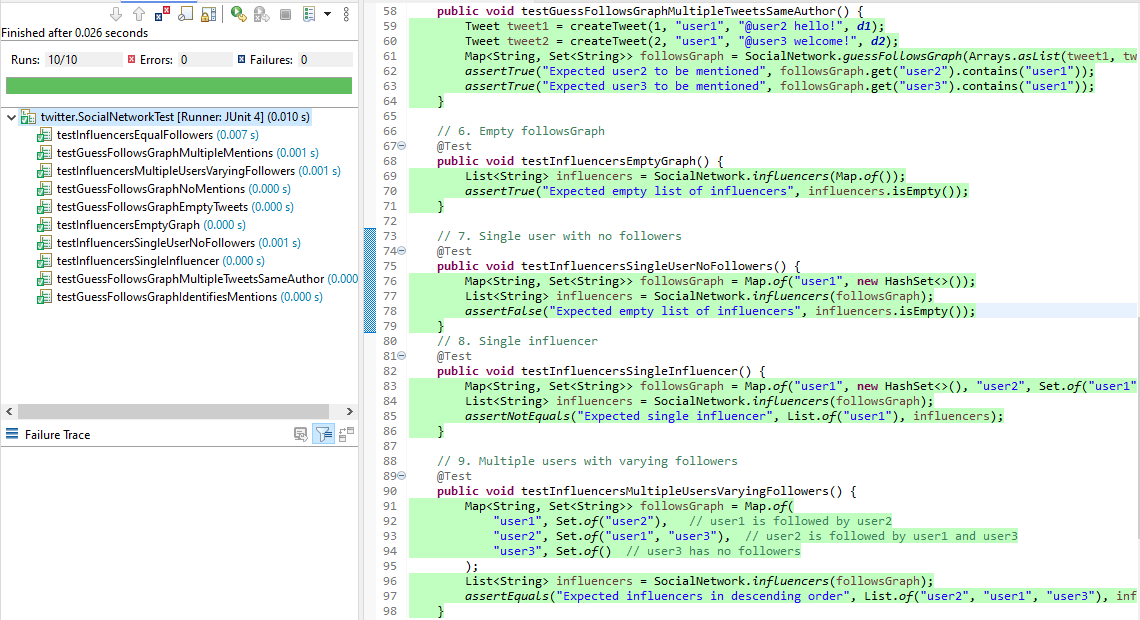
assertTrue("Expected multiple users in any order", influencers.contains("user3"));

assertTrue("Expected multiple users in any order", influencers.contains("user4"));

}

}

**ScreenShot:**



Github link: