CSC 212 Programming Assignment # 4
Recommending friends on social networks
Due date: 10/12/2019
Mark: 8+3. Plagiarism:-11.

Guidelines:	This is an **individual** assignment. The assignment must be submitted to **Web-CAT**

1 Introduction

One of the main functionalities offered by online social platforms such as Facebook and Twitter is the recommendation of new friends. This is achieved by utilizing various information about the users, but the main factor used for recommending a new friend to a user is how well these two users are connected. A social network such as Facebook can be represented as undirected graph such as the one shown in Figure 1. We can use the information contained in the graph to select the top candidate friends for a given user. There are many ways to do this, but we will focus on two methods:

1. **Popular users**: In this method, we recommend the most popular users in the graph, that is nodes with the highest degrees (number of neighbors).

   Example 1. *If we want to recommend 4 new friends for user 3 using the popular users method, we recommend:*

   (a) User 8, which has degree 7.

   (b) User 12, which has degree 5.

   (c) User 4, which has degree 3.

   (d) User 6, which has degree 3 (we break ties according to user ID).

2. **Common neighbors**: In this method, we recommend users who have the most common friends with the user.

   Example 2. *If we want to recommend 4 new friends for user 3 using the common neighbors method, we recommend:*

   (a) User 4, which has 2 common neighbors with 3, nodes 1 and 5.

   (b) User 6, which has 2 common neighbors with 3, nodes 2 and 5.

   (c) User 12, which has 1 common neighbor with 3, node 1.

   (d) User 8, which has 0 common neighbors with 3 (we break ties according to user ID).

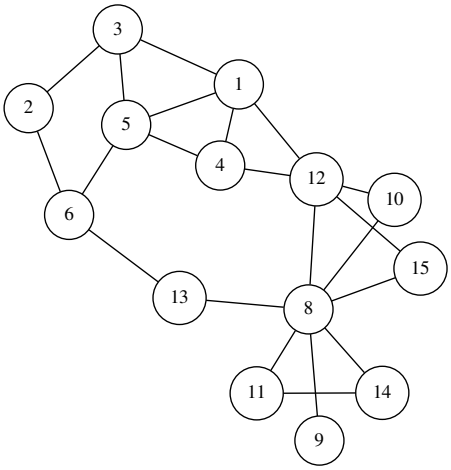

Figure 1: Example of a graph representing a social network.

In this assignment, your are going to create data structures to represent graphs and use them to implement these friend recommendation methods.

2 The data structures

In this section, we present the data structures necessary for this assignment.

2.1 Implementing a top k priority queue

To recommend top k users, we use a priority queue that keeps only the top k elements and serves them in decreasing order of priority. For this, write the class `PQKImp` that implements the interface `PQK` below.

```
public interface PQK<P extends Comparable<P>, T> {

  // Return the length of the queue
  int length();

  // Enqueue a new element. The queue keeps the k elements with the
     highest priority. In case of a tie apply FIFO.
  void enqueue(P pr, T e);

  // Serve the element with the highest priority. In case of a tie apply
     FIFO.
  Pair<P, T> serve();
}
```

The class `PQKImp` takes the parameter k as parameter in the constructor:

```
public class PQKImp<P extends Comparable<P>, T> implements PQK<P, T> {
  public PQKImp(int k) {
    ...
  }
  ...
}
```

2.2 Implementing a map

In this step, you write a BST implementation of a map. The `Map` interface is as follows:

```
public interface Map<K extends Comparable<K>, T> {

  // Return the size of the map.
  int size();

  // Return true if the map is full.
  boolean full();

  // Remove all elements from the map.
  void clear();

  // Update the data of the key k if it exists and return true. If k does
      not exist, the method returns false.
  boolean update(K k, T e);

  // Search for element with key k and returns a pair containing true and
      its data if it exists. If k does not exist, the method returns
     false and null.
  Pair<Boolean, T> retrieve(K k);

  // Insert a new element if does not exist and return true. If k already
      exists, return false.
  boolean insert(K k, T e);

  // Remove the element with key k if it exists and return true. If the
     element does not exist return false.
  boolean remove(K k);

  // Return the list of keys in increasing order.
  List<K> getKeys();
}
```

Notice that this interface does not have a current element. Write the class `BSTMap` that implements the interface `Map`:

```
public class BSTMap<K extends Comparable<K>, T> implements Map<K, T> {
  public BSTNode<K, T> root; // Do not change this
  ...
  public BSTMap() {
    ...
  }
}
```

The interface `List` is defined as follows:

```
public interface List<T> {
  boolean empty();

  boolean full();

  void findFirst();

  void findNext();

  boolean last();

  T retrieve();

  void update(T e);

  void insert(T e);
```

```
  void remove();

  // Return the number of elements in the list.
  int size();

  // Searches for e in the list. Current must not change.
  boolean exists(T e);
}
```

3 Representing the social network

To represent the friendship graph, we use the following interface:

```
public interface Graph<K extends Comparable<K>> {

  // Add a node to the graph if it does not exist and return true. If the
      node already exists, return false.
  boolean addNode(K i);

  // Check if i is a node
  boolean isNode(K i);

  // Add an edge to the graph if it does not exist and return true. If i
     or j do not exist or the edge (i, j) already exists, return false.
  boolean addEdge(K i, K j);

  // Check if (i, j) is an edge.
  boolean isEdge(K i, K j);

  // Return the set of neighbors of node i. If i does not exist, the
     method returns null.
  List<K> neighb(K i);

  // Return the degree (the number of neighbors) of node i. If i does not
      exist, the method returns -1.
  int deg(K i);

  // Return a list containing the nodes in increasing order.
  List<K> getNodes();
}
```

We will use adjacency list representation, but instead of an array of lists, we use a map of lists. Each list in the map represents the neighbors of a node. Write the class `MGraph` that implements the interface `Graph` using this representation:

```
public class MGraph<K extends Comparable<K>> implements Graph<K> {
  public Map<K, List<K>> adj; // Do not change this
  public MGraph() {
    ...
  }
  ...
}
```

4 The friends recommender

Write the class `Recommender` that implements the two friends recommendation methods discussed above:

```
import java.io.File;
import java.util.Scanner;

public class Recommender {

  // Return the top k recommended friends for user i using the popular
     nodes method. If i does not exist, return null. In case of a tie,
     users with the lowest id are selected.
  public static <K extends Comparable<K>> PQK<Double, K> recommendPop(
     Graph<K> g, K i, int k) {
    return null;
  }

  // Return the top k recommended friends for user i using common
     neighbors method. If i does not exist, return null. In case of a tie
     , users with the lowest id are selected.
  public static <K extends Comparable<K>> PQK<Double, K> recommendCN(
     Graph<K> g, K i, int k) {
    return null;
  }
}
```

5 Deliverable and rules

You must deliver:

1. Source code submission to Web-CAT. You have to upload the following classes (and any other classes you need)in a zipped file:

   - `PQKImp.java`
   - `BSTMap.java`
   - `MGraph.java`
   - `Recommender.java`

   Notice that you should **not upload** the interfaces and classes given to you.

The submission **deadline** is: **10/12/2019**.

You have to read and follow the following rules:

1. The specification given in the assignment (**class and interface names, and method signatures**) must not be modified. Any change to the specification results in compilation errors and consequently the mark zero.

2. All data structures used in this assignment **must be implemented** by the student. The use of Java collections or any other data structures library is strictly forbidden.

3. This assignment is an individual assignment. Sharing code with other students will result in harsh penalties.

4. Posting the code of the assignment or a link to it on public servers, social platforms or any communication media including but not limited to Facebook, Twitter or WhatsApp will result in disciplinary measures against any involved parties.

5. The submitted software will be evaluated automatically using Web-Cat.

6. All submitted code will be automatically checked for similarity, and if plagiarism is confirmed penalties will apply.

7. You may be selected for discussing your code with an examiner at the discretion of the teaching team. If the examiner concludes plagiarism has taken place, penalties will apply.