CS 10 PS-4

Social network analysis (the "Bacon" game)

This problem set tackles the important social network problem of finding an actor's "Bacon number". Starting with an actor, see if they have been in a movie with someone who has been in a movie with someone who has been in a movie ... who has been in a movie with Kevin Bacon. They're usually at most 6 steps away. There are plenty of other 6-degrees-of-separation phenomena in social networks. In a geekier version, the center of the universe is Paul Erdos, a profilic author and coauthor, and people are characterized by their Erdos numbers. The highest known finite Erdos number is 13. Remarkably, there are a number of people who have both small Erdos numbers and small Bacon numbers (number = steps away). Dan Kleitman has total Erdos-Bacon number of 3 (Erdos 1, Bacon 2), but the Bacon number is due to a role as an extra. Danica McKellar has an Erdos-Bacon number of 6, and is both a professional actress (*The Wonder Years* and *West Wing*) and wrote a published math paper as well as supplemental math texts designed for teenage girls (*Math Doesn't Suck*, *Kiss My Math*, and *Hot X: Algebra Exposed*).

Background

In this problem set you will write code for social network analysis, and you will use it in variations on the Kevin Bacon game.

In the Kevin Bacon game, the vertices are actors and the edge relationship is "appeared together in a movie". The goal is to find the shortest path between two actors. Traditionally the goal is to find the shortest path to Kevin Bacon, but we'll allow anybody to be the center of the acting universe. We'll also do some analyses to help find better Bacons. Since "degree" means the number of edges to/from a vertex, I'll refer to the number of steps away as the "separation" rather than the common "degrees of separation".

Here are some examples from the sample solution (slightly reformatted for space, and to omit answers that you're supposed to compute):

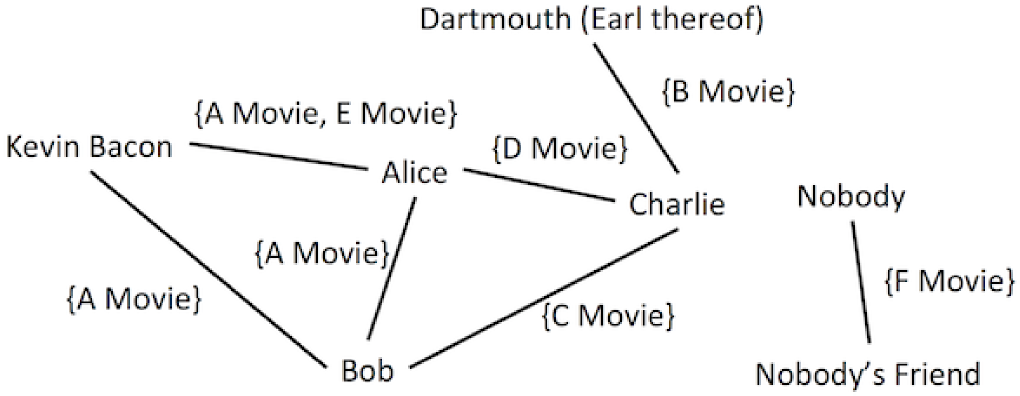
```
Commands:
c <#>: list top (positive number) or bottom (negative) <#> centers of the universe, sorted by average separation
d <low> <high>: list actors sorted by degree, with degree between low and high
i: list actors with infinite separation from the current center
p <name>: find path from <name> to current center of the universe
s <low> <high>: list actors sorted by non-infinite separation from the current center, with separation between low and high
u <name>: make <name> the center of the universe
q: quit game

Kevin Bacon is now the center of the acting universe, connected to [#]/9235 actors with average separation [#]

Kevin Bacon game >
p Diane Keaton
Diane Keaton's number is 2
Diane Keaton appeared in [Something's Gotta Give (2003)] with Jack Nicholson
Jack Nicholson appeared in [Few Good Men, A (1992)] with Kevin Bacon

Kevin Bacon game >
p LeVar Burton
LeVar Burton's number is 3
LeVar Burton appeared in [Star Trek: Insurrection (1998), Star Trek: Nemesis (2002),
  Star Trek: First Contact (1996), Star Trek: Generations (1994)] with Patrick Stewart
Patrick Stewart appeared in [Conspiracy Theory (1997)] with Julia Roberts
Julia Roberts appeared in [Flatliners (1990)] with Kevin Bacon

Kevin Bacon game >
p Buster Keaton
Buster Keaton's number is 5
Buster Keaton appeared in [Limelight (1952)] with Claire Bloom
Claire Bloom appeared in [Crimes and Misdemeanors (1989)] with Martin Landau
Martin Landau appeared in [Tucker: The Man and His Dream (1988)] with Joan Allen
Joan Allen appeared in [Searching for Bobby Fischer (1993)] with Joe Mantegna
Joe Mantegna appeared in [Queens Logic (1991)] with Kevin Bacon

Kevin Bacon game >
u John Longden
John Longden is now the center of the acting universe, connected to [#]/9235 actors with average separation [#]

John Longden game >
p Kevin Bacon
Kevin Bacon's number is 8
Kevin Bacon appeared in [She's Having a Baby (1988)] with Elizabeth McGovern
Elizabeth McGovern appeared in [Wings of Courage (1995)] with Val Kilmer
Val Kilmer appeared in [Top Secret! (1984)] with Peter Cushing
Peter Cushing appeared in [Tales from the Crypt (1972)] with Richard Greene
Richard Greene appeared in [Little Princess, The (1939)] with Ian Hunter
Ian Hunter appeared in [Ring, The (1927)] with Carl Brisson
Carl Brisson appeared in [Manxman, The (1929)] with Anny Ondra
Anny Ondra appeared in [Blackmail (1929)] with John Longden

John Longden game >
p Buster Keaton
Buster Keaton's number is 9
Buster Keaton appeared in [Limelight (1952)] with Nigel Bruce
Nigel Bruce appeared in [Suspicion (1941)] with Cary Grant
Cary Grant appeared in [Affair to Remember, An (1957)] with Deborah Kerr
Deborah Kerr appeared in [End of the Affair, The (1955)] with Peter Cushing
Peter Cushing appeared in [Tales from the Crypt (1972)] with Richard Greene
Richard Greene appeared in [Little Princess, The (1939)] with Ian Hunter
Ian Hunter appeared in [Ring, The (1927)] with Carl Brisson
```

```
Carl Brisson appeared in [Manxman, The (1929)] with Anny Ondra
Anny Ondra appeared in [Blackmail (1929)] with John Longden
```

So based on the data set we supply for this problem, Diane Keaton's Bacon Number is two, LeVar Burton's is three, and Buster Keaton's is five. Note that actors can costar in multiple movies, as LeVar Burton has with Patrick Stewart. And we can switch the center of the universe, here to somebody far removed from both Kevin Bacon and Buster Keaton.

The easiest way to play the Kevin Bacon game is to do a breadth-first search (BFS), as covered in lecture. This builds a tree of shortest paths from every actor who can reach Kevin Bacon back to Kevin Bacon. More generally, given a root, BFS builds a shortest-path tree from every vertex that can reach the root back to the root. It is a tree where every vertex points to its parent, and the parent is the next vertex in a shortest path to the root. For the purposes of this assignment, we will store the tree as a directed graph. Once the tree is constructed, we can find the vertex for an actor of interest, and follow edges back to the root, tracking movies (edge labels) and actors (vertices) along the way.

Here's an example with a small graph.



In this small graph, Alice and Bob have Kevin Bacon numbers of 1, Charlie's Kevin Bacon number is 2, and Dartmouth's Kevin Bacon number is 3. Nobody and Nobody's Friend have *infinite* Kevin Bacon numbers, since there is no path between them and Kevin Bacon. A query for Charlie could produce the following output:

```
Charlie appeared in [C movie] with Bob
Bob appeared in [A movie] with Kevin Bacon
```

In fact, there are two paths with three edges between Charlie and Kevin Bacon. Instead of going through Bob, the path could have gone through Alice. That's OK, because we don't need all the paths with fewest edges to Kevin Bacon; we just need one of them.

Inputs

Datasets are provided in bacon.zip, both a simple set of test files for the graph illustrated above, and a large set of actor-movie data (thanks to Brad Miller at Luther College). The three main files, actors.txt, movies.txt, and movie-actors.txt are large — 9,235 actors, 7,067 movies, and 21,370 movie-actor pairs, resulting in 32,337 edges. So while you are developing your program use the smaller versions actorsTest.txt, moviesTest.txt, movie-actorsTest.txt.

The files are all formatted the same way. Each line has two quantities separated by a "|". In the actors file the quantities are actorID and actorName. In the movies file they are movieID and movieName. In the movies-actors file they are movieID and actorID, indicating that the actor associated with actorID appeared in the movie associated with movieID.

Use the file contents to build a graph whose vertices are the actor names (not IDs), and whose edges are labeled with *sets* of the movie names (again, not IDs) in which actors appeared together. So an edge is created when two actors costar, its label is a set of strings, and each movie in which they costar is added to the edge label. You can assume that no movie appears twice in the movies file and that no actor appears twice in the actors file.

You may find it useful to create maps for mapping IDs to actor names and IDs to movie names. You can also use a map to figure out which actors appeared in each movie, and can use that information to add the appropriate edges to the graph. This may take a little thought, but try it by hand on the small data set given above.

A piece of advice about processing the lines of the input files. For each line of each file, you will need to find the string to the left of the pipe symbol and the string to the right of the pipe symbol. I suggest using the familiar split method that we've used before for String objects. One catch here is that the pipe character is a special character that you will need to escape. In such a case, a line like this should work:

```
line.split("\\|");
```

where line is a String from a file that has data separated by pipes.

The Game Interface

You may devise your own interface for playing the game, and need not follow the structure of the sample solution above. But you do need to support the following functionality:

- change the center of the acting universe to a valid actor
- find the shortest path to an actor from the current center of the universe
- find the number of actors who have a path (connected by some number of steps) to the current center
- find the average path length over all actors who are connected by some path to the current center.

In addition, your program should help you find other possible Bacons, according to two different criteria:

- degree (number of costars). This is the same as in the short assignment.
- average separation (path length) when serving as center of the universe. This simply requires looping over all the actors one by one. Consider each one as the center of the universe, find the average path length using the method above, and store it away for sorting. So just a few lines of code. It could take some time, but under a minute on my machine with the sample solution. Since not all actors are in the Bacon universe (have a path to him), you can limit the result to those that are.

Java notes re the command-line interface. The info retrieval lecture demonstrated the use of `Scanner` to get a line of input. Also note that `Integer.parseInt` will extract an `int` from a `String` (throwing an exception if the format is invalid).

Exercises

For this assignment, you may work either alone, or with one other partner. Same discussion as always about partners.

1. Make a graph library class with BFS-related methods; you can just augment the one from class and the short assignment if you want. These should all be generic with respect to vertex and edge types (i.e., not specialized for Bacon graphs). In class I demonstrated a BFS using a Map to track the previous vertex, but in this assignment you should create a "path tree" using a Graph during your BFS, where the root of the path tree is the center of the universe and all actors that have been in a movie with the center of the universe are children of the root. Other actors that have been a movie with a child of the root are grandchildren of the root and so on. Unlike a standard Tree, edges in your path tree should point from a child to its parent (that's why we use a Graph instead of a Tree in this assignment). This way if you are given a vertex, you can follow edges back to the root (the center of the universe). Create the following methods:

   ```
   public static <V,E> Graph<V,E> bfs(Graph<V,E> g, V source);
   public static <V,E> List<V> getPath(Graph<V,E> tree, V v);
   public static <V,E> Set<V> missingVertices(Graph<V,E> graph, Graph<V,E> subgraph);
   public static <V,E> double averageSeparation(Graph<V,E> tree, V root);
   ```

   a. BFS to find shortest path tree for a current center of the universe. Return a path tree as a Graph.

   b. Given a shortest path tree and a vertex, construct a path from the vertex back to the center of the universe.

   c. Given a graph and a subgraph (here shortest path tree), determine which vertices are in the graph but not the subgraph (here, not reached by BFS).

   d. Find the average distance-from-root in a shortest path tree, *without enumerating all the paths*. Warning: this one takes some thinking to do correctly and efficiently; don't put it off for the last minute. Hint: the average is the sum divided by the number, and computing the sum is now a tree recursion problem (in which you need to pass information from parent to child, too).

   Test these functions on the simple graph illustrated above. Hard-code the addition of the vertices and edges.

2. Implement the Bacon game, with a command-line interface supporting the analyses described in the previous section.

   a. The graph class and the library use generic types `V` and `E`. `V` is the type of the vertex; you'll want it to be `String` because it's an actor's name. `E` is the type of the edge label; you'll want it to be `Set<String>`, to hold the movies in which the actors costar.

   b. Test your program on the movieTest.txt, actorTest.txt, and movie-actorTest.txt files. Make sure to demonstrate that you program works for boundary conditions.

   c. When you are sure that your program works on the test data, change it to use the movie.txt, actor.txt, and movie-actor.txt files. Demonstrate that your program works for these as well.

   d. Who would you recommend as good and bad Bacon, in terms of degree and in terms of average separation. Justify, in comparison to others including Bacon himself.

Some extra credit suggestions (other possibilities welcome too):

- Perform additional network analysis (e.g., betweenness centrality, "page" rank, clique / dense subgraph detection).
- This movie data is a bit out of date now. Find a newer or more comprehensive dataset (perhaps by way of querying some web service). Or do a six-degree-of-separation game with a different social network.
- Get a different / bigger dataset (or a subset of a bigger dataset) and see how your program scales. E.g., see Stanford Large Network Dataset Collection and links therein. Perhaps try doing random walks instead of BFS if things start to break down.
- Reformat the data into Gephi format. Then set up a Bacon game Gephi module, that plays the game on a graph loaded into the software.

Submission Instructions

Turn in your code and test results. The code should implement BFS and associated analyses (on generic graphs), read the files and produce the graph, and play the game. The tests should show the graph constructed from the hand-coded example and from the test files, and example runs playing the game.

Grading rubric

Total of 100 points

Correctness (70 points)

10	Breadth first search algorithm
5	Construct shortest path tree during BFS
10	Construct path from vertex to root
5	Identify disconnected vertices
10	Compute average path lengths of connected vertices
10	Read data files and create costar graph
5	Search for best center of universe by path length
15	Interactive interface

Structure (10 points)

4	Good decomposition into objects and methods
3	Proper use of instance and local variables
3	Proper use of parameters

Style (10 points)

3	Comments for classes and methods

4 Good names for methods, variables, parameters
3 Layout (blank lines, indentation, no line wraps, etc.)

Testing (10 points)

5 Include test runs and analyses specified above.
5 Demonstrate that the code works for boundary cases