**The (Kevin) Bacon Number Game**

In the Kevin Bacon game, you are given an actor and try to find the shortest sequence of actors between the given actor and Kevin Bacon, where you list actors consecutively if they appeared in a movie together. For example, the silent film star Renée Adorée was in *The Blackbird* (1926) with Doris Lloyd, who was in *Keep 'Em Flying* (1941) with Carol Bruce, who was in *Planes, Trains, and Automobiles* (1987) with Kevin Bacon. Since that is the shortest sequence of actors between Renée Adorée and Kevin Bacon, we say that Renée Adorée's **Kevin Bacon number** is 3.

In this assignment, you will build an undirected graph whose vertices are movie actors and edges indicate that the actors appeared in the same film. You will then use a breadth-first search (BFS) to answer queries about paths in the BFS tree from actors to Kevin Bacon—their "Kevin Bacon numbers."

The input to the Kevin Bacon game comes from three files, which are (unfortunately) outdated and not at all complete. The [Oracle of Bacon](http://oracleofbacon.org/) site maintains a much more up-to-date database internally; they claim to scour the [Internet Movie Database](http://www.imdb.com/) (IMDb) every couple of weeks. The example above for Renée Adorée comes from the Oracle of Bacon site. However, their files contain 2.9 million actors and 1.9 million movies and TV shows (which is a bit excessive for this assignment!). You are supplied with the following files:

* The "actors.txt" file contains one line per actor. Each line consists of an integer ID, a pipe symbol (|), and an actor's name, with no spaces around the pipe. The file contains 9235 lines.
  + For testing, you are supplied a much smaller version of this file in "actorsTest.txt", consisting of only seven lines as follows:

1|Kevin Bacon

3|Tom Hanks

841|Meg Ryan

3823|Parker Posey

3590|Lisa Kudrow

2976|W.C. Fields

1534|George Burns

* The "movies.txt" file contains one line per movie. Each line consists of an integer ID, a pipe symbol, and the name of a movie. The file contains 7067 lines.
  + The smaller version for testing is "moviesTest.txt":

150|Apollo 13 (1995)

539|Sleepless in Seattle (1993)

6885|In the Cut (2003)

2424|You've Got Mail (1998)

1160|Six of a Kind (1934)

1875|Clockwatchers (1997)

* The "movie-actors.txt" file contains one line for each actor's appearance in a movie. Each line consists of a movie ID, a pipe symbol, and an actor ID, indicating that the actor appeared in the movie.
  + The smaller version for testing is "movie-actorsTest.txt":

150|1

150|3

6885|1

6885|841

539|3

539|841

2424|3

2424|3823

2424|841

1875|3823

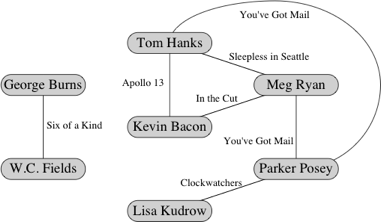
1875|3590

1160|2976

1160|1534

For example, the first line indicates that actor 1 (Kevin Bacon) appeared in movie 150 (Apollo 13 (1995)).

Here is the undirected graph that the test files give:



Vertices are labeled with actor names, and edges are labeled with movie names (the years were omitted). In this small graph, Tom Hanks and Meg Ryan have Kevin Bacon numbers of 1, Parker Posey's Kevin Bacon number is 2, and Lisa Kudrow's Kevin Bacon number is 3. W.C. Fields and George Burns have undefined Kevin Bacon numbers, since there is no path between them and Kevin Bacon. A query for Lisa Kudrow could produce the following output:

Enter the name of an actor: Lisa Kudrow

Lisa Kudrow's Kevin Bacon number is 3.

In fact, there are two paths with three edges between Lisa Kudrow and Kevin Bacon. Instead of going through Tom Hanks, the path could have gone through Meg Ryan. That's OK, because we don't need *all* the paths with fewest edges to Kevin Bacon; we just need one of them. Recall that a BFS of a graph with unweighted edges will by default produce the shortest path (the least number of edges).

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.

The String method indexOf returns the index of a character within a string. Once you know where the pipe symbol is within a line, you can use the method substring to extract the strings on either side of the pipe symbol. As usual, you can read about these methods in the [Java documentation](http://docs.oracle.com/javase/8/docs/api/).

Be careful if using the String.split method: this method uses a regular expression (regex) delimiter; the pipe symbol is a protected regex character, it will need to be escaped (split("\\|")) if you want it to split on a pipe literal.

***Forming the Bacon graph.*** From the three input files, you will need to form an undirected graph, which we call the **Bacon graph** (though there's nothing special about Kevin Bacon's place in the graph). It should be like the one above for the test files.

If you look carefully at the test files, you'll see something interesting. In the figure, the edge between Tom Hanks (ID 1) and Meg Ryan (ID 841) is labeled with the movie *Sleepless in Seattle* (ID 539). However, it could also have been labeled with the movie *You've Got Mail* (ID 2424). Put another way, we could have had two edges between Tom Hanks and Meg Ryan, since they made both of those rom-coms together.

When you form the graph, define an insertEdge method, which throws an IllegalArgumentException if the graph already contains an edge between the two vertices given in the call. *You should catch this exception and ignore it.*  The extra edge won't be added to the graph, but you don't want the exception to crash your program. You *could* just have the method silently ignore duplicates, but throwing/catching the exception is more explicit and forces the graph client to be cognizant of what's happening.

You are encouraged to choose the data structures for representing the graph on your own. You may make some missteps along the way, but you will probably end up with a better understanding of graph algorithms. If you'd like a couple hints before starting, check the **FAQ**.

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. I suggest that you make, for each movie, some sort of iterable set of actors in the movie. For example, you could make an ArrayList of actors for each movie. As you process the "movie-actors.txt" file, add the actor in each line to the movie in that line. When you're done reading the file, you have, for each movie, a collection of all the actors in that movie. Once you read the entire file, you're ready to add edges to the Bacon graph.

For each movie, go through the collection of actors in that movie. For each actor x in the collection, go through all the other actors that appear *after* him or her and add an edge between x and the other actor. If a movie has k actors, then you'll add k − 1 edges between the first actor and all those who follow, k − 2 edges between the second actor and all those who follow, and so on, until the next-to-last actor, who will have an edge to only the last actor in the list. The total number of edges added will be:

(k − 1) + (k − 2) + (k − 3) + ... +  2 + 1 + 0

…which equals k(k−1)/2. (You don't have to use an ArrayList. It's just a suggestion.) What is the Big-O of building the graph? Ideally you wouldn't be doing this more than once (per root).

***Running a breadth-first search.*** Once the Bacon graph is formed, you'll need to run a BFS from a root vertex (the Kevin Bacon vertex). You will need a queue for your BFS. Any sort of linked list will do.

You need to record the distance from the root for each vertex. That is, you need to record Kevin Bacon numbers. You can store distances as you perform the BFS; you might find it convenient to store distances in a Map whose keys are either actor names or Vertex objects. You could also construct another graph as a directed tree, which you could traverse to count the distances.

Optionally, you can print out edges as you find them, or you can append them to a list and then print out the edges in the list. It's up to you. (If you make a list, then you can find the length of the path by just asking for the length of the list.)

If there is no path from the actor to the root, your program should say so. And if the actor is not even in the "actors.txt" file, your program should say that, too. Example runs:

Enter the name of an actor: Art Clokey

Art Clokey's Kevin Bacon number is undefined.

Enter the name of an actor: Moe Howard

Moe Howard is not in our database.

Enter the name of an actor: Parker Posey

Parker Posey has a Bacon number of 2.

**(Advanced) Other ideas**

* ***Printing the "path".*** Devise a way to store and output the "path" of movies that connect two actors (rather than just the Bacon number). Example:

Enter the name of an actor: Parker Posey

Parker Posey's Kevin Bacon number is 2.

Parker Posey appeared in You've Got Mail (1998) with Meg Ryan.

Meg Ryan appeared in In the Cut (2003) with Kevin Bacon.

* ***Other roots.*** Why be limited to the Kevin Bacon game? Why not be able to play the Tom Hanks game, for example? The sample client below has an option where if the first character typed into the console is a question mark, the rest of the line is taken as the name of an actor who becomes the root:

Enter the name of an actor: Tom Hanks

Now playing the Tom Hanks game!

Enter the name of an actor: John Belushi

John Belushi's Tom Hanks number is 3.

John Belushi appeared in Neighbors (1981) with Dan Aykroyd.

Dan Aykroyd appeared in Great Outdoors, The (1988) with John Candy.

John Candy appeared in Splash (1984) with Tom Hanks.

Of course, when you switch to a new root, you will have to compute a whole new BFS tree.

* ***All actors with a specific number.*** The sample below also provides an option where, if the first character typed into the console is a digit, then the entire line is converted to an integer, and all actors whose distance from the root is given by that integer are listed. Continuing the previous example:

Enter the name of an actor: 7

Actors with a Tom Hanks number of 7:

Gordon Harker

Randle Ayrton

Jobyna Ralston

Jameson Thomas

Richard Arlen

Clara Bow

Malcolm Keen

Gibb McLaughlin

Anny Ondra

This option was implemented by making a Map that maps actor names to their distance from the root, and filling in entries the BFS tree is built.

* ***Spelling suggestions.*** It's easy to misspell an actor's name or to just use the wrong form of the name. Perhaps you type in *Timothy Allen* instead of *Tim Allen* or *Kate Blanchett* instead of *Cate Blanchett*. Devise some sort of scheme for finding "near matches" and reporting near matches to the name entered, so that the user can re-enter the correct form of the name. For full-featured auto-complete, see the "Autocomplete" project also, in the bonus labs folder.
* ***Statistics.*** What is the average Kevin Bacon number among actors with finite Kevin Bacon numbers? The median? What pair of actors has the longest path between them? (The length of a longest path is called the **diameter** of the graph.) Which actor has the shortest/longest finite path? (In other words, from which vertex do you minimize the maximum finite path length?)

*Adapted from the* ***Kevin Bacon Game*** *assignment*

*http://www.cs.dartmouth.edu/~scot/cs10/lab/lab5/lab5.html*