Dr. Scot Morse
Department of Computer Science
Western Oregon University

January 16, 2015

LAB 2 — Graphs Part I: Movies and Movie Stars

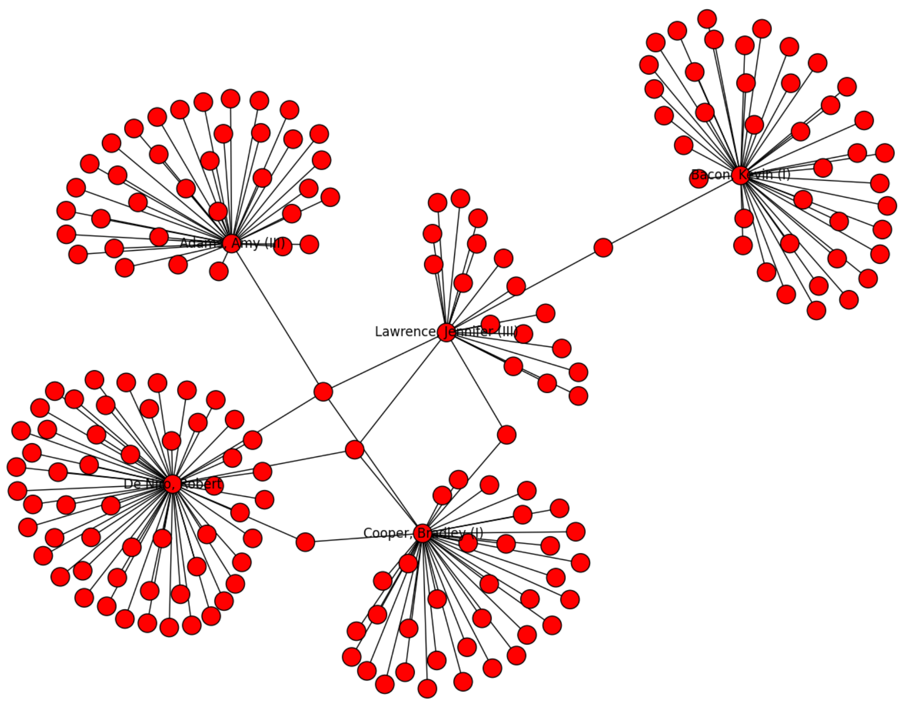


Figure 1: Subgraph for: Robert De Niro, Amy Adams, Jennifer Lawrence, Bradley Cooper and of course Kevin Bacon. Trivia without looking it up on IMDB, 'Jennifer Lawrence has a Kevin Bacon number of 1. Which movie did she co-star in with Kevin Bacon?' Figure uses `movie_2000.txt`.

This is the first lab about graphs. The main purpose here is to get you some seat time with their basic operations and simple algorithms. To do this we need two things: a graph data structure and some data.

The graph data structure can come from the book, with modifications/additions by you.

As for the data, we will set ourselves up to do something similar to the 'Degrees of Separation' Kevin Bacon problem. This is where we analyze the relationships between movies and the actors and actresses who starred in them. We'll save most of that for the next lab; for this one we just want to get started.

Begin by downloading the sample data on the class web page. There is one giant file that includes all movies and performers from the Internet Movie Database (IMDb), and two abbreviated ones that include movies since the year 2000.

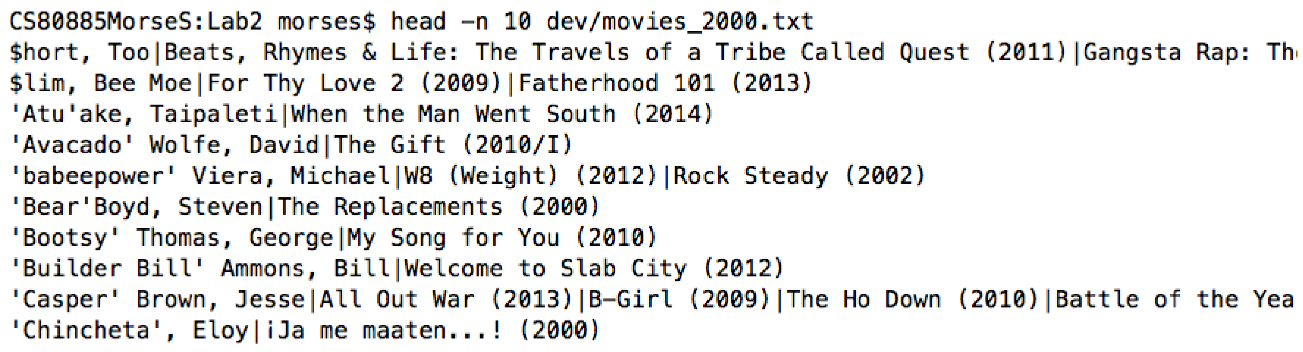
Complete the Java program (`Main.java`) to read a given file (making it easy to change files *on the command line* – specification to follow), extract movie and performer data and build an undirected graph.

Both performers and movies should be vertices. Edges correspond to the relationship where an actor or actress was in a movie. You should, therefore, not have any edges between movies or between performers.[1]

The movie files have this format:

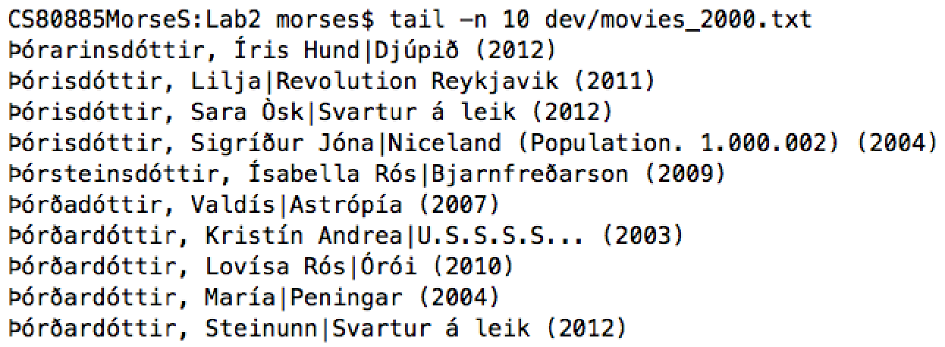
```
Performer name|movie|movie|movie
```

where each performer gets their own line. For the file containing movies since 2000, here are the first 10 lines (long lines cut off):



```
CS80885MorseS:Lab2 morses$ head -n 10 dev/movies_2000.txt
$hort, Too|Beats, Rhymes & Life: The Travels of a Tribe Called Quest (2011)|Gangsta Rap: Th
$lim, Bee Moe|For Thy Love 2 (2009)|Fatherhood 101 (2013)
'Atu'ake, Taipaleti|When the Man Went South (2014)
'Avacado' Wolfe, David|The Gift (2010/I)
'babeepower' Viera, Michael|W8 (Weight) (2012)|Rock Steady (2002)
'Bear'Boyd, Steven|The Replacements (2000)
'Bootsy' Thomas, George|My Song for You (2010)
'Builder Bill' Ammons, Bill|Welcome to Slab City (2012)
'Casper' Brown, Jesse|All Out War (2013)|B-Girl (2009)|The Ho Down (2010)|Battle of the Yea
'Chincheta', Eloy|¡Ja me maaten...! (2000)
```

and the last 10 lines:



```
CS80885MorseS:Lab2 morses$ tail -n 10 dev/movies_2000.txt
Þórarinsdóttir, Íris Hund|Djúpið (2012)
Þórisdóttir, Lilja|Revolution Reykjavik (2011)
Þórisdóttir, Sara Òsk|Svartur á leik (2012)
Þórisdóttir, Sigríður Jóna|Niceland (Population. 1.000.002) (2004)
Þórsteinsdóttir, Ísabella Rós|Bjarnfreðarson (2009)
Þórðadóttir, Valdís|Astrópía (2007)
Þórðardóttir, Kristín Andrea|U.S.S.S.S... (2003)
Þórðardóttir, Lovísa Rós|Órói (2010)
Þórðardóttir, María|Peningar (2004)
Þórðardóttir, Steinunn|Svartur á leik (2012)
```

[1] That would be nonsensical – how could a movie act in a movie? I suppose two actors could be married, but let's not go there.

Here are some things to notice about the data and format:

- Names are in *Last, First* format
- The delimiter separating actors from movies and between movies is the | character.
- Each movie (should) have a year in parentheses. Occasionally it will look like (????) or (2010/I) or (2006/II).
- Performer names aren't always what you would expect, as the first 10 shows you.
- The files are encoded in UTF-8. You **must** read it with this encoding. Some systems (Linux and some Mac) use it as the default, but don't trust that. Windows definitely does not use this encoding. And all bets are off if your OS is set up in Chinese or Arabic. Make sure you specify the `Charset` as `UTF-8` when reading the text file. If you don't then you *will have problems.*
- There is only one performer per line and every line has one performer.
- I believe every performer has at least one movie, but don't depend on it.
- Movies, obviously, appear more than once.
- There are many movies and movie stars. Perhaps it would be good to think about efficiency? For both time *and* space.

Problem 1.

Download Sedgewick's `Graph`, `Bag`, `Stack`, `SymbolGraph` and `ST` classes. If you want to compile these as is or run his examples then you'll need the `stdlib.jar` file. (You can use this jar file one of three ways: 1) if using Eclipse, add it as a library just like you did with JUnit; 2) unjar it (`jar -xvf stdlib.jar`) next to your code; or 3) put it next to your code and add it to your classpath when compiling and running (`javac -classpath .:stdlib.jar *.java` and `java -classpath .:stdlib.jar Graph input.txt`; change the : to a ; if you're on a Windows machine.) Make sure you can compile and run simple examples. For this lab we won't be implementing these classes ourselves, but will need to modify them. We won't need the `stdlib.jar` code so at some point delete or comment out the existing `main` and other dependencies so we can compile and run with just the code above.

Use these classes as your starting point to implement the lab requirements that follow. It is possible to implement the methods below without using a graph, but you may not do that. All methods below must use the graph and its public methods to do their work (yes you may add public methods, but try not to make it tightly coupled to your application code – a graph should be a general purpose data structure). This lab is about graphs for one, and besides you'll need graphs for the lab that follows.

Problem 2. *100 points*

Here is the interface we want to support, in `Main.java`. See that file for full javadocs. Make it happen!

```
/**
 * Get the total number of vertices in the graph.
 */
public static int getNumberOfVertices() {}

/**
 * Get the total number of edges in the graph.
 */
public static int getNumberOfEdges() {}

/**
 * Get a list of all the movies that the given actress or actor has acted in.
 */
public static List<String> getMovies( String performer ) {}

/**
 * Get a list of the top performers, according to how many movies they have acted in.
 */
public static List<String> getTopPerformers( int n ) {}

/**
 * Get a list of the top movies, according to how many performers they have.
 */
public static List<String> getTopMovies( int n ) {}
```

NOTE: It would be wise to think ahead to the next lab where we will pick up where we leave off here and implement depth-first and/or bread-first search/traversal, or shortest path as well as other graph algorithms.

Turn it in

We will turn this lab in on paper and electronically via the same turn-in web page as for Lab 1. Here you'll have multiple source code files to submit. When you're ready, select only those source code files you want to submit and zip them. Name your zip file `Lab2.zip`. Here are things to be mindful of.

- **Don't** put your code in a folder before you zip it.
- **Don't** put the movie text file in your zip.
- **Don't** zip your Eclipse folder.
- **Don't** put any .class files in your zip.
- **Don't** use rar, 7z, cab, dmg, tgz, bzip2, xar or any other type of compressed archive.
- **Do** double check that the files you handed in are everything that is needed to run your program. Try unzipping it elsewhere and see if everything is there. Compile it to make sure.
- **Do** use the command line to compile and run your code, because that is what I will do.

Submitted by Dr. Scot Morse on January 16, 2015.

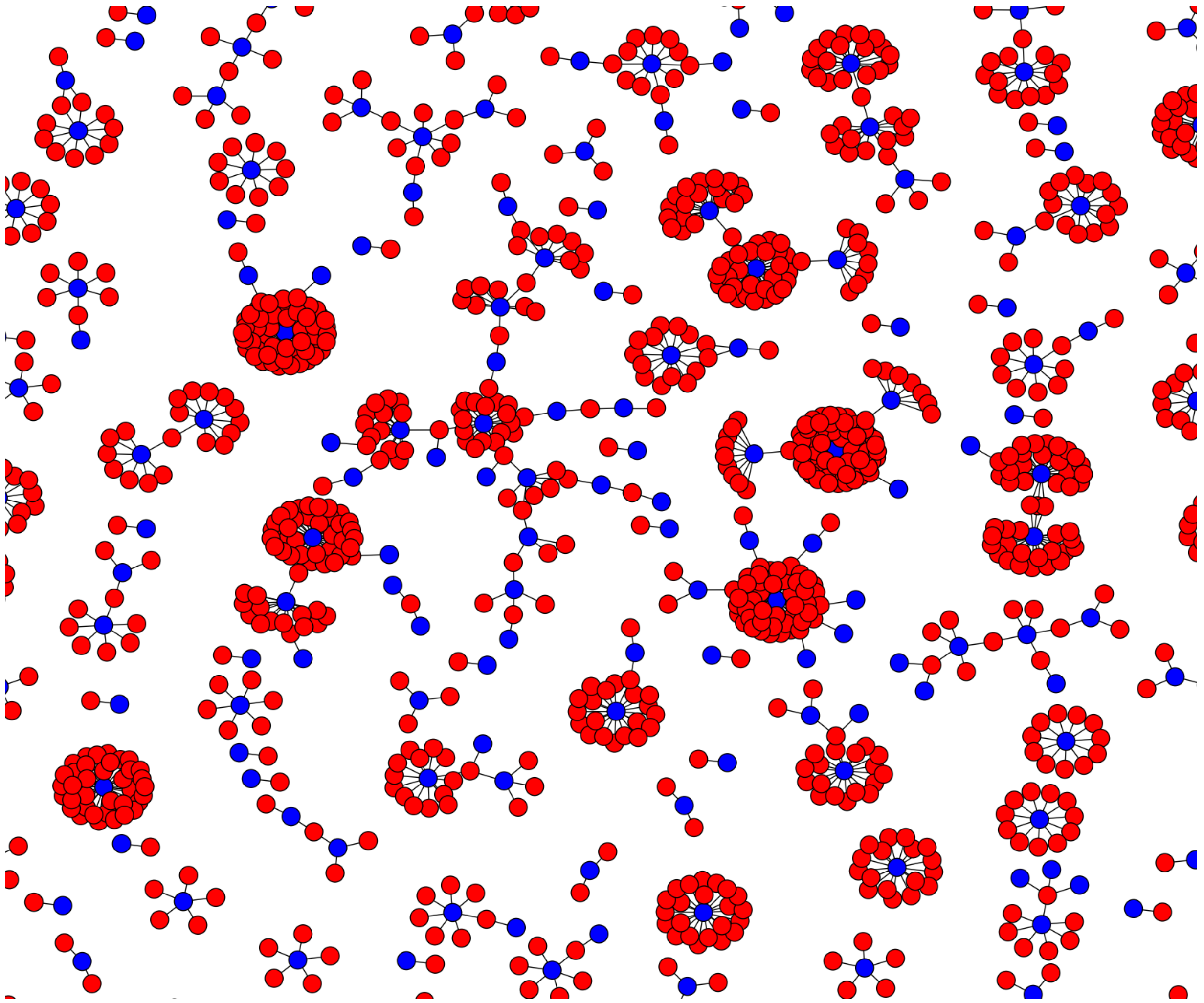


Figure 2: Snapshot of the graph as it is being built. Blue vertices are performers; red are movies. Since movies typically have many actors/actresses you can tell that the graph has a lot more data remaining to be loaded.