About Lab 10

In this lab you will read in files that contain many lines of the form

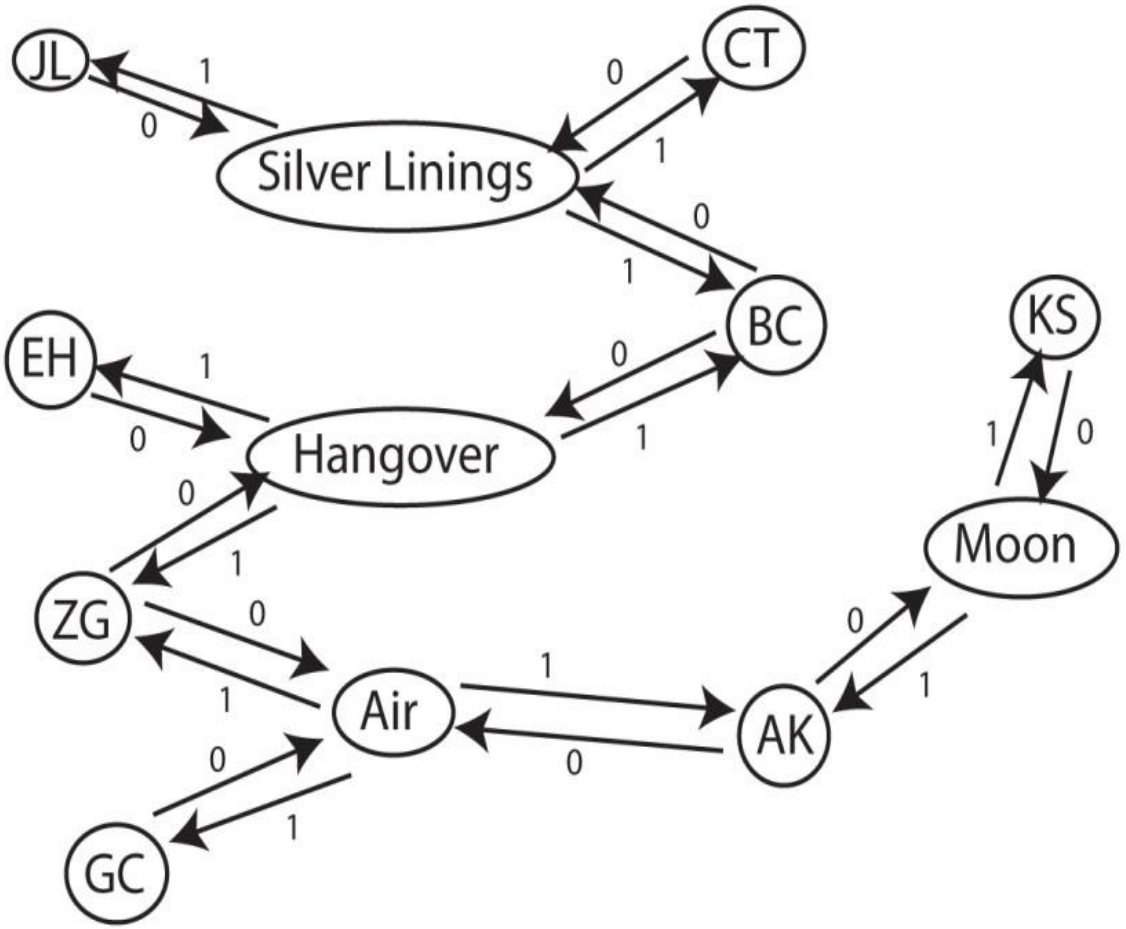
Actor | Movie

As you read the file you form a graph in which both actors and movies are nodes. There is an edge of weight 1 from each movie to each actor in that movie. There is an edge of weight 0 from each actor to each movie that actor is in.

For example, this might be the contents of such a file:

Jennifer Lawrence | Silver Linings Playbook
Bradley Cooper | Silver Linings Playbook
Chris Tucker | Silver Linings Playbook
Bradley Cooper | The Hangover
Ed Helms | The Hangover
Zach Galifianakis | The Hangover
George Clooney | Up in the Air
Anna Kendrick | Up in the Air
Zach Galifianakis | Up in the Air
Anna Kendrick | New Moon
Kristen Stewart |New Moon

This file generates the following graph (using abbreviations for the titles and initials for the actors to save space on this page):



You can see in the graph that there is a path of length 1 from Jennifer Lawrence to Bradley Cooper, a path of length 2 from Jennifer Lawrence to Zach Galifianakis, and one of length 3 from Jennifer Lawrence to George Clooney. The length of the path is the "Bacon distance" separating those actors. Furthermore, if you printed all of the nodes on the path you would have the sequence of actors and the movies that link them together.

The bad news is that this graph has many cycles so we can't use our super-efficient acyclic shortest path algorithm, but the weights are non-negative so we can use the efficient Dijkstra algorithm for finding shortest paths.

On April 19 we talked about implementing graphs and shortest path algorithms. You might want to look back at that.

The key ideas were

a) A Vertex class that contains the name of the node a list of its outgoing edges, the cost of the shortest path from the source node to this vertex, and the previous vertex on the shortest path to it. The latter two fields you assign while the algorithm is running. A Vertex also has a boolean variable *done*, initialized to false.

(cont'd)

b) An Edge class that has the destination Vertex of the edge and its weight.
c) A Graph class that contains a HashMap<String, Vertex> object representing the graph. You will probably have a ShortestPath( ) and a PrintPath( ) method for this class.

Building the graph should be easy. For an edge

A | B

look in the HashMap for the graph to get the Vertex corresponding to string A; call this vertexA. If there is no such vertex, make one. Do the same for B, getting vertexB. Then add to the outgoing edge list for vertexA a new Edge to vertexB of weight 0, and add to the edge list for vertexB a new Edge to vertexA of weight 1. You are going to need a structure (a Set perhaps?) that holds the names of all of the actors, so add A to this structure if it isn't already there.

For the ShortestPath method have a PriorityQueue<Entry>, where an Entry has a Vertex, its previous vertex, and its distance from the source. Start it off with (SourceVertex, null, 0). Each time you poll the PriorityQueue to get an Entry, check to see if its Vertex X is done. If so, ignore the Entry. If it isn't done, make it done, and assign to it the Entry's distance and previous Vertex. Then walk along X's outgoing edge list and add an Entry for each non-done node, making X its previous and making the distance to X plus the cost of the edge its cost. Continue this until the PriorityQueue becomes empty.

If you have all of this implemented correctly, you should find the particular Kevin Bacon methods easy to implement. "Recentering" is just a matter of calling the ShortestPath method with a new source. After you do this you can find the average distance to each actor by running through your actor set (or list or whatever), bringing up the Vertex for this via your graph HashMap, and adding its distance onto your running total if this distance is less than its initial value of INFINITY (for which I use Integer.MAX_VALUE). Divide this by the number of actors whose distance was less than INFINITY and you have the average. Other methods are similar.

Note that the lab expects you to implement a small input loop that allows the user to specify commands and responds to those commands:

command : find Robert Redford

prints the shortest path from the actor to the current center

command: recenter Robert Redford

runs the ShortestPath algorithm using this actor as the source

command: avgdist

finds the average distance from the current center to each reachable actor,

and so forth.

Here is some data about the various input files

Name	Size	# Lines	# Nodes	# Actors
small	79 KB	1,817	1,747	161
top250	560 KB	14,339	12,716	12,466
pre1950	38 MB	966.338	234,022	127,552
post1950	284 MB	6,848,516	2,838,261	2,215,490
no-tv-v	225 MB	5,793,218	2,483,127	1,952,232
full	323 MB	7,814,854	3,043,915	2,314,654