# CSE201 Assignment
# Programming HW1: On finding less than six degrees of separation



## 1 Problem description

This is a fun assignment. Once you solve it, you might waste countless hours improving your pop culture knowledge. You have been forewarned.

Read this document carefully. It probably answers the questions you have in mind.

**Main objective:** You are going to design a program that solves the "Six Degrees of Kevin Bacon" game. In terms of CS, you will parse a dataset into a graph and perform shortest path computations. But I think it's more interesting to talk about Six Degrees of separation.

Given any two actors, the aim is to find a "path" between them, consisting to actors with whom they have coacted. For example, given Brad Pitt and Rachel McAdams, one can find a "path" between them:

Brad Pitt -(Fury: Blood Brothers)- David Ayer -(The Making of 'End of Watch')- Jake Gyllenhaal -(Southpaw: Inside the Ring)- Rachel McAdams

Meaning, Brad Pitt appeared with David Ayer in "Fury: Blood Brothers", who appeared with Jake Gyllenhaal in "The Making of 'End of Watch'", who appeared with Rachel McAdams in "Southpaw: Inside the Ring".

I will give you a snapshot of movie credit data (scraped from IMBD), and your program needs to find the **shortest** such paths. Efficiency in important, since your code will have to parse more than 100,000 movies (with a similar number of actors) to find these paths.

**Basic setup:** Download the file `hw1.zip` and unzip it. You will get movie credit files, which are the raw data files you need to process. You will get some helper Python code to build graphs. And, since I'm a nice guy, you will get some test input/output files.

- There are two movie credits files, `all-imdb-cleaned.txt` and `smaller-imdb-cleaned.txt`. Each line of these files is of the form <MOVIE> <ACTOR1> <ACTOR2> .... To make parsing easy, the movie and actor names have underscores in them to represent space. Thus, to parse a line, you simply tokenize by

space. The first token is the movie, all remaining tokens are actors. For example, here is a line you may recognize.

Terminator_2:_Judgment_Day Arnold_Schwarzenegger Linda_Hamilton Edward_Furlong Robert_Patrick

**There may be different movies with the same name.** Each line is a different movie, even if the movies have the same name. Be wary of this issues, since it can lead to errors.

The file `all-imdb-cleaned.txt` has 827K movies, scraped from a few years ago. The smaller file `smaller-imdb-cleaned.txt` has 131K movies, a random subset of the previous file. This file is provided so that more experimentation can be done.

- The file `graph-tools.py` is a simple graph library that I wrote. It has basic functions to create and represent a graph as an adjacency list. You do not have to use it, but I think it will make your life easier. You do not need to read over internal implementations to understand the functions.

- The file `more-input.txt` (and the corresponding output file) are example outputs for the shortest path problem on the `smaller-imdb-cleaned.txt` file. While your output could discover different paths, it must have the same length as these example outputs.

**The algorithmics:** Before getting into how your solution should be structured, let us discuss the algorithmics of the basic problem. Given a movie credits file, you need to construct an undirected co-actor graph $G$. This is the graph formed by connecting any two actors who have acted together in a movie.

Given two vertices $s$ and $t$, you have to find a shortest path between them. Let us discuss two algorithms to do so.

*Vanilla BFS:* This is the procedure we discussed in class, with a slight modification. Start a BFS from $s$. As soon as $t$ is visited, terminate the procedure.

*Bidirectional BFS:* In practice, Bi-BFS is a common heuristic used to reduce the search time. Start two "simultaneous" BFSes, one from $s$ and the other from $t$. So there are two queues, one of each BFS. At each step, we pick *one* of the queues, dequeue it, and process as usual. Then, we pick the *other* queue, dequeue it, and process. So we keep switching between the queues. As soon as we discover a vertex $v$ that is reached by both BFSes, we terminate. We stitch together the shortest paths from $v$ to $s$ and from $v$ to $t$, to get a path from $s$ to $t$. Turns out, this is a shortest path from $s$ to $t$ (proof will be part of the homework!).

You will need to implement both these algorithms. To understand the efficiency of these algorithms, you will also need to put in a "counter" that counts the number of edges traversed. Specifically, each method has a for loop going over the neighbors of a dequeued vertex. Increment a counter for each step of the loop. The value of the counter at termination is the number of edges seen by the procedure.

Important!! To improve efficiency, implement the `visited` data structure as a set data structure. Implement the `pred` data structure as a dictionary, and do not perform an initialization step involving a for loop over all vertices. *Do not create arrays whose size is the number of vertices.* This will make your code faster (think about why).

## 2 What you need to do

**Implementation details:** Write all the code as a Python notebook, with comments on what is in each block. Someone using your code should be able to run all the code, and obtain all the output for the analysis questions (below). Here is what should be implemented.

- First, write code that constructs the co-actor graph for a given movie credit file.
- Write code that does a full BFS from a source $s$. You may also want to write code that does a BFS of whole graph.
- Implement the BFS and Bi-BFS procedures that find paths from a source to a destination.
- Write code to answer the analysis questions below.

Manage error handling and corner cases however you want. Structure code however you want, but make sure you add comments so a user can understand your code. *Your code should not crash!*

**What you need to find:** This is a list of all the questions you need to answer. Mark a separate code block for each of these. The answer could be the output of your block, some text, and/or a plot.

1. For each credits file, get the shortest path of all the pairs in the example input files. The corner cases to worry about are (i) an actor is not present in the graph and (ii) $s$ and $t$ are not connected. For each of these cases, output the appropriate message.
2. Add in some interesting shortest paths you discovered, in either credits file.
3. How do the shortest paths typically differ between the same vertices in both the credits? Obviously, one of them should lead to shorter paths. Pick some random pairs and show the difference.
4. Pick some famous starting actors/vertices (let's definitely do Kevin Bacon). What are the sizes of the level sets from Kevin Bacon? These are the sets of vertices at distance 1, distance 2, distance 3, etc. It is nicest to give this answer as a plot.

5. What are the sizes of the connected components in each of the graphs?

6. (This is really important) What is the difference in the number of edges seen (the counter we discussed earlier) between vanilla BFS and Bi-BFS? Do a comparison for the pairs in the example input files. Generate some of your own examples.

7. Taking the previous point further, do a comparison of the "wall-clock" time, which is the actual time taken for BFS or Bi-BFS. Use the `timeit` class of Python to get these running times.

# 3 How did I get the data

I got the raw data from `https://datasets.imdbws.com/` and `https://www.imdb.com/interfaces/`. To get it manageable size, I (and a student I had hired over the summer) only kept non-adult titles. I also trimmed it to US releases.