Bacon Number

Submission to website: Monday, February 29, 10pm

Checkoff by LA/TA: Tuesday, March 1, 10pm

This lab assumes you have Python 2.7 installed on your machine. Please use the Chrome web browser.

Introduction

Have you heard of *Six Degrees of Separation*? This simple theory states that at most 6 people separate you from any other person in the world. (Facebook actually showed that the number among their users is significantly lower, at about 4.6. You can read more about it in a research paper.)

Hollywood has its own version: Kevin Bacon is the center of the universe (not really, but let's let him feel good about himself). Every actor who has acted with Kevin Bacon in a movie is assigned a "Bacon number" of 1, every actor who acted with someone who acted with Kevin Bacon is given a "Bacon number" of 2, and so on. (What Bacon number does Kevin Bacon have? Think about it for a second.)

Note that if George Clooney acts in a movie with Julia Roberts, who has acted with Kevin Bacon in a different film, George has a Bacon number of 2 through this relationship. If George himself has also acted in a movie with Kevin, however, then his Bacon number is 1, and the connection through Julia is irrelevant. We define the notion of a "Bacon number" to be the *smallest* numer of films separating a given actor (or actress) from Kevin Bacon.

In this lab, we will explore the notion of the Bacon number. We, your friendly 6.S04 staff, have prepared an ambitious database of approximately 37,000 actors and 10,000 films so that you may look up your favorites. Given that this class does not focus primarily on performance optimization, however, we test your code against a smaller data set. Did Julia Roberts and Kevin Bacon act in the same movie? And what does Robert De Niro have to do with Frozen? Let's find out!

The film database

We've mined a large database of actors and films from IMDB via the www.themoviedb.org API. We present this data set to you as a list of tuples (3-element lists), each of the form `[actor_id_1, actor_id_2, film_id]`. Each tuple denotes the fact that actors given by `actor_id_1` and `actor_id_2` played in a film denoted by a `film_id`. The list of tuples in the data set is exactly all pairs of actors starring in any film in our data set.

The required component of this lab works with a subset of this data set, presented via `small.json`, which is passed to your functions via `data`, as described below. If you wish to do fancier things with this data set, consult the Bonus section.

`lab.py`

This file is yours to edit in order to complete this lab. You are not expected to read or write any other code provided as part of this lab. In `lab.py`, you will find a skeleton for your solution: please correctly implement the methods `did_x_and_y_act_together`, `get_actors_with_bacon_number` and `get_bacon_path` according to this document in order to earn full credit.

Your code will be loaded into a small server (`server.py`, must the same as the one from the Panda lab) and will serve up a visualization website. To use the visualization, run `server.py` and use your web browser navigate to localhost:8000. You will need to restart the `server.py` in order to re-load your code if you make changes.

Part 1: Better together (`did_x_and_y_act_together`)

`did_x_and_y_act_together(data, actor_id_1, actor_id_2)`

Given the database (`data`, a list of tuples of actors who have acted together in a film, as well as a film id: `[actor_id_1, actor_id_2, film_id]`), produce `True` if the actors denoted by `actor_id_1` and `actor_id_2` acted in a film together, and `False` otherwise. For example, Kevin Bacon (`id=4724`) and Steve Park (`id=4025`) did *not* act in a film together, meaning `did_x_and_y_act_together(data=..., 4724, 4025)` should return `False`.

To check your code with the web UI, run `server.py` and try the following pairs (which all should return `True`): Kevin Bacon and Matt Dillon, Bernard Freyd and Elisabeth Depardieu, Samuel L. Jackson and Yvonne Zima.

Please note that this function is a warmup and was given to help you to get familiar with the DBs. You don't have to use this piece of code in part 2 and 3.

Part 2: To infinity and beyond! (`get_actors_with_bacon_number`)

`get_actors_with_bacon_number(data, n)`

Given the database (`data` , a list of pairs of actors who have acted together in a film), produce a list of actor IDs of all actors with a *bacon number* of `n` . **The list should be sorted in ascending order**. We define the *bacon number* to be the **smallest** number of films separating a given actor from Kevin Bacon, whose actor ID is `4724` .

Some questions to get you started: What is the set of actors with a *bacon number* of 1, as given by `data` ? Given the set of actors with a *bacon number* of 1, think of how you can find the set of actors with a *bacon number* of 2. Given the definition of the *bacon number* observe that an actor with a *bacon number* of `i+1` will be connected to Kevin Bacon via acting in a movie with some actor with a *bacon number* of `i` , who in turn is connected through some actor with a *bacon number* of `i-1` , and so on. `get_actors_with_bacon_number(data, n)` is best written by decomposing it into functions: we recommend you write a helper function that takes a list of all actors with a *bacon number* of `i` and generates the list of all actors with a *bacon number* of `i+1` .

Note that the test cases in `test.py` run against small and large databases of actors and films, and that your implementation need to be efficient enough to handle the large database in a timely manner.

Part 3: Finding nemo (`get_bacon_path`)

Given the database (`data` , a list of tuples of actors who have acted together in a film, as before), produce a list of actor IDs (any such shortest list, if there are several) detailing a “bacon path” from Kevin Bacon to the actor denoted by `actor_id` . For example, if we run this method with Julia Roberts’s ID (`actor_id=1204`), one valid path is `[4724, 3087, 1204]` , showing that Kevin Bacon (`4724`) has acted with Robert Duvall (`3087`), who in turn acted with Julia Roberts (`1204`). (Can you guess the films? You can find out using our data set!). If no path exists, return `None` . Any shortest list that connects Bacon to the actor denoted by `actor_id` is valid.

Furthermore, you may wish to optimize your code to handle the large database. Avoid repeatedly iterating through all of `data` . The staff solution completes all tests in a few seconds.

Please note, because the paths are not necessarily unique, the tester does not hard-code the correct paths, and only verifies the *length* of the path you find (also that it is indeed a path that exists in the database).

If you wish to do other things with this ambitious data set, you may wish to examine `imdb_util.py` , which we used to produce `small.json` and `large.json` . You may use it to extract additional information from the film database.

Auto-grader (unit tester)

As in the previous lab, we provide you with a `test.py` script to help you verify the correctness of your code. The script will call the required methods in `lab.py` and verify their output. Again, we encourage you to write your own test cases to further verify the correctness of your code, and to help you diagnose any problems. We will only use the provided test cases to auto-grade your work. You may also wish to investigate the Python debugger (PDB) to help you find bugs efficiently.

In-Browser UI

Once you're done, you can visualize your code! Run `server.py` and open your browser to localhost:8000. You will be able to see actors as circular nodes (hover above the node to see the actor's name) and the movies as edges linking nodes together.

Above the graph, we define three different tabs, one for each component of the lab (the last of which is optional). Each tab sets up the visualization appropriate for each aspect of the lab.

Does your lab work? Do all tests in `test.py` pass? You're done! Submit your `lab.py` on funprog, and get your lab checked off by a friendly staff member.

Good luck! Start early.