Bacon Number

Submission to website: Monday, February 20, 10pm

Checkoff by LA/TA: Thursday, February 23, 10pm

This lab assumes you have Python 3.5 or later installed on your machine. Please use the Chrome web browser.

The lab has 14 tests and 3 coding points. In this lab the coding points are determined by the time it takes to complete the tests on the server:

- all tests complete in less than 1 second each: 3 points
- all tests complete in less than 2 seconds each: 2 points
- all tests complete in less than 3 seconds each: 1 point

Some notes on code performance:

- Using the `in` operator to test for membership in a `list` is slow when the list is long. If the order of the list elements isn't important consider using a `set` or `dict` since the time to test membership using those data types is independent of their size.
- `l.pop(0)` is a very expensive operation if the list `l` is long. Ask yourself if you really need to remove the element from the list, of if you can simply use an index to visit each element of the list in turn.
- the movie data passed as a `list` to the functions below may not be the most efficient representation of that information. Think about what operations you need to perform on the data and then consider if there's another data structure that would make those operations much easier to code and/or faster to execute.

The server allocates 30 seconds to complete all the tests.

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 6.009 staff, have prepared an ambitious database of approximately 37,000 actors and 10,000 films so that you may look up your favorites. 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 fromIMDB via the www.themoviedb.org API. We present this data set to you as a list of records (3-element lists), each of the form `[actor_id_1, actor_id_2, film_id]`, which tells us that `actor_id_2` acted with `actor_id_1` in a film denoted by `film_id`. Since "acts with" is a symmetric relationship, if the record `[a1,a2,f]` is in the list, then `[a2,a1,f]` is also in the list. You may be able to use this fact to simplify your code. The list of records in the data set is exactly all pairs of actors acting in any film in our data set.

The tests use both a small and large version of the database. Tests 12-14 use the large version and a slow implementation for part 3 might cause the tests to not complete within the 30-second time limit. See Part 3 below for how to work around this.

`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`) 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 reload your code if you make changes.

Part 1: Better together (`did_x_and_y_act_together`) [tests 1-3]

`did_x_and_y_act_together(data, actor_id_1, actor_id_2)`

Given the database (`data` , a list of records 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 Parts 2 and 3.

Part 2: To infinity and beyond! (`get_actors_with_bacon_number`) [tests 4-7]

`get_actors_with_bacon_number(data, n)`

Given the database (`data` , a list of pairs of actors who have acted together in a film), reeturn a list of actor IDs of all actors with a *Bacon number* of `n` . **The returned 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 all actors with a *Bacon number* of `i` and returns all actors with a *Bacon number* of `i+1` (but make sure that this function doesn't assign multiple Bacon numbers to the same actor!).

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

Part 3: Finding Nemo (`get_bacon_path`) [tests 8-14]

Given the database (`data` , a list of records 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 should optimize your code to handle the large database. In particular,***avoid repeatedly iterating through all of*** `data` ***.*** Test 12-14 use the large database, so if your code runs too slowly on these tests, the server will not complete execution of `test.py` within the allotted time limit and you'll receive a score of 0. To get credit for passing the tests with the small database, you can skip tests that use the large database by including the following statement as the first statement in `get_bacon_path` :

```
if len(data) > 500000: raise ValueError('large data')
```

Please note that the paths are not necessarily unique, so 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. 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` at web.mit.edu/6.009 and get your lab checked off by a friendly staff member.

Good luck! Start early.