Lab 2: Bacon Number

You are not logged in.

If you are a current student, please Log In for full access to the web site.
Note that this link will take you to an external site (`https://shimmer.csail.mit.edu`) to authenticate, and then you will be redirected back to this page.

Table of Contents

- 1) Preparation
- 2) Introduction
  - 2.1) The Film Database
  - 2.2) The Names Database
  - 2.3) Using the UI
  - 2.4) `lab.py` and `test.py`
- 3) Acting Together
- 4) Bacon Number
- 5) Paths
  - 5.1) Bacon Paths
    - 5.1.1) Speed
  - 5.2) Arbitrary Paths
- 6) Movie Paths
- 7) Code Submission
- 8) Checkoff
  - 8.1) Grade

1) Preparation

This lab assumes you have Python 3.7 or later installed on your machine.

The following file contains code and other resources as a starting point for this lab: `lab2.zip`

Most of your changes should be made to `lab.py`, which you will submit at the end of this lab. Importantly, you should not add any imports to the file.

This lab is worth a total of 4 points. Your score for the lab is based on:

- correctly answering the questions throughout this page (1 point)
- passing the `test.py` tests and the server-only tests efficiently (2 points, see below), and
- a brief "checkoff" conversation with a staff member to share your code, including discussion of the bullets below and a review of your code's clarity/style (1 point).

Your points for the test cases are based on how quickly your code runs on the server together with correctness. Your lab code will be expected to run in a certain amount of time (which varies depending on the test); this execution time is based on the amount of time it takes to run that test on the server (*not* your own machine).

The questions on this page, and the code submission at the end, are due at 4pm on Friday, September 20.

2) 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* number of films separating a given actor (or actress) from Kevin Bacon.

In this lab, we will explore the notion of the Bacon number. We 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!

2.1) 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 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`.

Keep in mind that "acts with" is a symmetric relationship. If `[a1, a2, f]` is in the database, it is true both that `a1` acted with `a2` *and* that `a2` acted with `a1`, even if `[a2, a1, f]` is not explicitly represented in the database.

However, these relationships do not necessarily exhibit the transitive property. That is, if `[a1, a2, f]` and `[a2, a3, f]` are in the database, it is *not necessarily true* that `a1` and `a3` have acted together (unless `[a1, a3, f]` or `[a3, a1, f]` is in the database).

We store these data as JSON files. The server tests will use `small.json` and `large.json`, but we have also included a `tiny.json` that you will use to write your own tests.

2.2) The Names Database

The methods in `lab.py` expect you to use integer actor IDs, but the tests we give you on this page will have actor names as inputs and outputs.

To help with this mapping, we include a file, `resources/names.json`, which has a JSON representation of the mapping between actor IDs and names. You can use the `load` method of Python's `json` module to get the data out of the file and into Python. For an example of this, check out how we load databases in the `setUp` function of `test.py`.

Answer the following questions *using Python*.

Which of the following best describes the Python object that results from loading `resources/names.json`?

-- ▼

This question is due on Friday September 20, 2019 at 04:00:00 PM.

What is Katarina Cas's ID number?

This question is due on Friday September 20, 2019 at 04:00:00 PM.

Which actor has the ID 83462?

This question is due on Friday September 20, 2019 at 04:00:00 PM.

2.3) Using the UI

We have also provided a visualization website which loads your code into a small server (`server.py`) and visualizes your results. To use the visualization, run `server.py` and use your web browser navigate to localhost:8000. You will need to restart `server.py` in order to reload your code if you make changes.

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 its aspect of the lab.

2.4) `lab.py` and `test.py`

These files are yours to edit in order to complete this lab. You should implement the main functionality of the lab in `lab.py`, and you should implement additional test cases (as described throughout the assignment) in `test.py`.

In `lab.py`, you will find a skeleton for the functions we expect you to write.

3) Acting Together

To get used to the structure of the databases, complete the definition of `did_x_and_y_act_together` in `lab.py`. This function should take three arguments in order:

- The database to be used (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]`),
- Two IDs representing actors

This function should return `True` if the two given actors ever acted together in a film 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(..., 4724, 4025)` should return `False`.

Inside `test.py`, we have included a `TestTiny` class which has a `setUp` method but no tests. Add at least one test testing `did_x_and_y_act_together` on the tiny database (you can open `tiny.json` in a text editor to see the data it contains).

When you are done implementing this method and it passes the associated tests, use your code to answer the following questions according to the data in the `resources/small.json` database. (Hint: You will need to load `names.json` to get the mapping from actors to IDs, then find the actor corresponding to the ID; consider writing a helper method that does this automatically.)

According to the `small.json` database, have Patrick Malahide and Dan Warry-Smith acted together?

-- ▼

This question is due on Friday September 20, 2019 at 04:00:00 PM.

According to the `small.json` database, have Carole Bouquet and Jennifer Taylor acted together?

-- ▼

This question is due on Friday September 20, 2019 at 04:00:00 PM.

Please note that `did_x_and_y_act_together` is a warm-up and was given to help you get familiar with the structure of the databases. You don't have to use the function in subsequent sections. Also note, though, that `test.py` does test this function.

4) Bacon Number

Complete the definition of `get_actors_with_bacon_number` in `lab.py`. This function should take two arguments in order:

- The database to be used (the same structure as before)
- The desired Bacon number

This function should return a Python set containing the ID numbers of all the actors with that Bacon number. Note that we'll define the *Bacon number* to be the **smallest** number of films separating a given actor from Kevin Bacon, whose actor ID is `4724`.

Look at the data in `tiny.json` (we suggest you do this *without* using Python i.e. open it up in a text editor) and answer these questions:

What are the **ID numbers** of the actors who have a Bacon number of 0 in `tiny.json`? Enter your answer below as a Python `set` of integers:

This question is due on Friday September 20, 2019 at 04:00:00 PM.

What are the **ID numbers** of the actors who have a Bacon number of 1 in `tiny.json`? Enter your answer below as a Python `set` of integers:

This question is due on Friday September 20, 2019 at 04:00:00 PM.

What are the **ID numbers** of the actors who have a Bacon number of 2 in `tiny.json`? Enter your answer below as a Python `set` of integers:

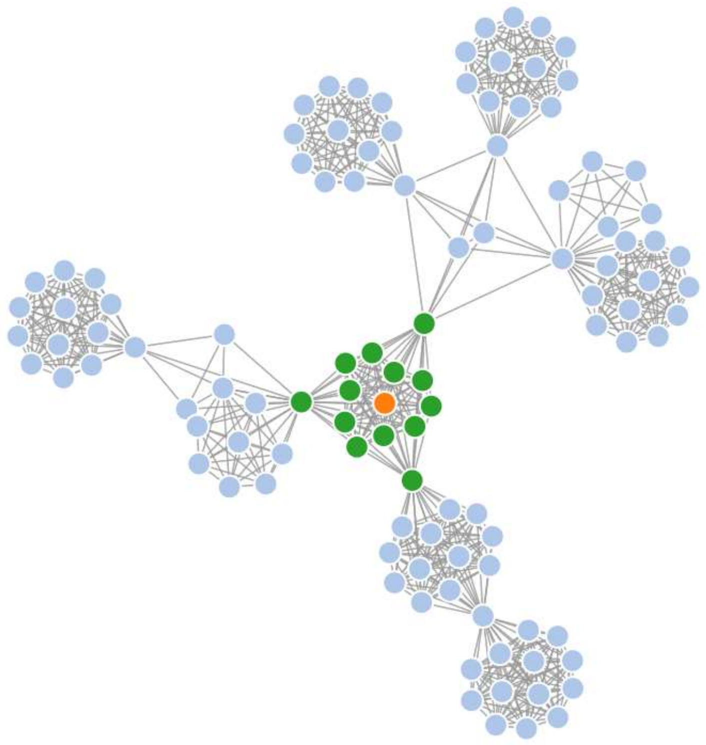
This question is due on Friday September 20, 2019 at 04:00:00 PM.

What are the **ID numbers** of the actors who have a Bacon number of 3 in `tiny.json`? Enter your answer below as a Python `set` of integers:

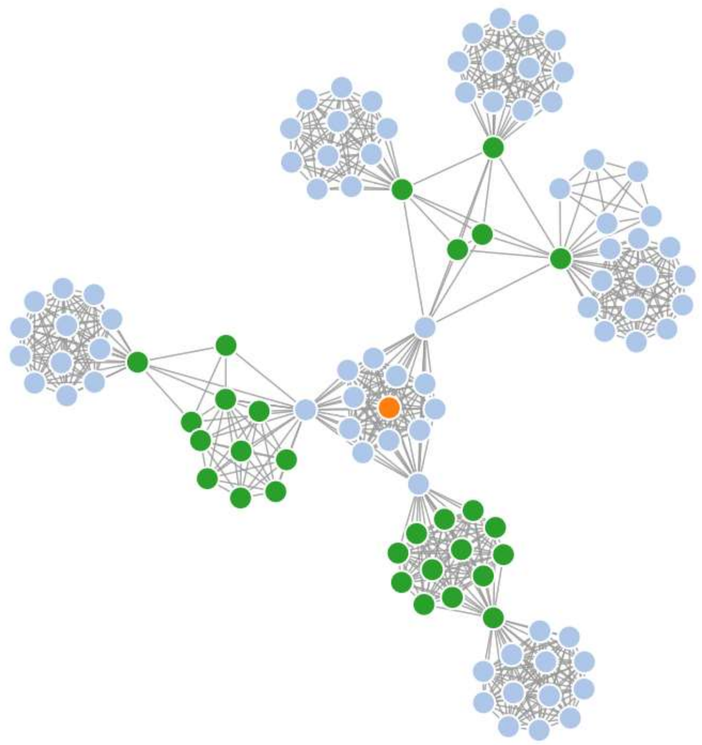
This question is due on Friday September 20, 2019 at 04:00:00 PM.

Add the four above questions as tests in the `TestTiny` class. Use your `get_actors_with_bacon_number` function to compute the sets of actors with Bacon numbers 0, 1, 2, and 3 and compare them against the results you just found above. So that they are recognized as individual tests, **define each as a separate method in the class, starting with `test_`** (e.g. `test_bacon_number_0`).

Now you're ready to write your Bacon number code! Here are some things to think about when writing your implementation. Consider the set of actors with a *Bacon number* of 1. Here is a visual representation of the data from the `small.json` database. (You can use our provided server to generate pictures like these.)



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:



Once you get a sense for how to get the *Bacon number* 2 actors from the *Bacon number* 1 actors, try to generalize to getting the *Bacon number* `i+1` actors from the *Bacon number* `i` actors.

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. Your code should also handle the case of arbitrary Bacon numbers (not just $n \leq 6$) since some databases may be structured to assign some actors quite large Bacon numbers.

When you're done writing this method and it passes all of your tests, answer the following question that uses the `large.json` database:

In the `large.json` database, what is the set of actors with Bacon number 6? Enter your answer below as a Python `set` of strings representing **actor names**:

This question is due on Friday September 20, 2019 at 04:00:00 PM.

5) Paths

Now we'll turn our attention to finding the *chain* of actors that connects Kevin Bacon to someone else.

5.1) Bacon Paths

Complete the definition of `get_bacon_path` in `lab.py`. The function should take two arguments in order:

- The database to be used (the same structure as before),
- An ID representing an actor

Your function should 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`. If no path exists, return `None`.

Please note that the paths are not necessarily unique, so any shortest list that connects Bacon to the actor denoted by `actor_id` is valid. The tester does not hard-code the correct paths and only verifies the *length* of the path you find (as well as that it is indeed a path that exists in the database).

For example, if we run this method on `large.json` 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`).

Take look at the data in `tiny.json`. What's the shortest path that connects actor `4724` to actor `1640`? Enter your answer below as a Python list of **ID numbers**:

This question is due on Friday September 20, 2019 at 04:00:00 PM.

Add a test for the case above to the `TestTiny` class in `test.py`. You can use this test to help make sure your function is implemented correctly.

5.1.1) Speed

When implementing the path-finding algorithm, you should optimize your code to handle the large database, which our testing infrastructure will use when testing your code.

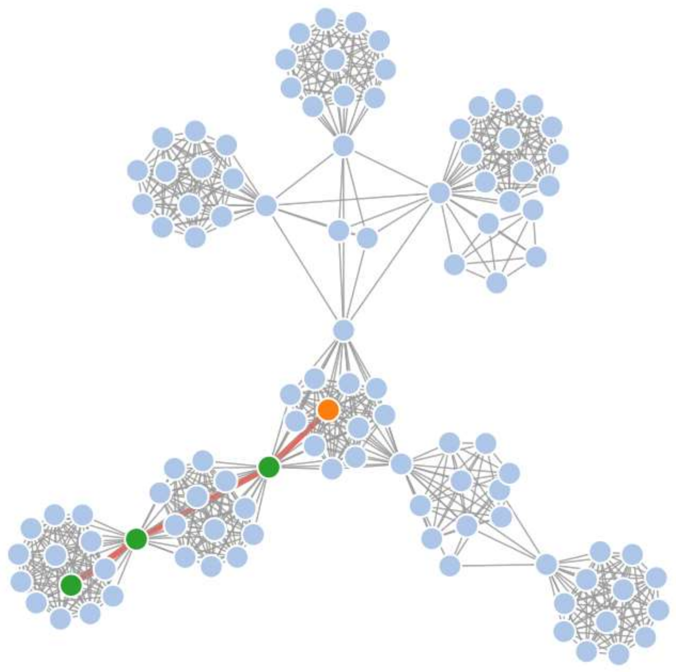
In particular, here are a few ideas about speed:

- Membership tests (the `in` operator) on long lists can be very slow. By contrast, the `in` operator is very fast on sets and dictionaries (regardless of the lengths of these objects). However, sets and dictionaries do not retain information about the order of their elements. Consider whether there are cases in your code where a set or dictionary can be used in place of a list.

- Running `L.pop(0)` on a long list is also slow. If you find yourself doing this, ask: do you really need to pop? Or can you just use an index to keep track of which list element you're working on?

- Searching through data using a `for` loop can be slow. Can you reorganize the data so that your search can be implemented with a single dictionary lookup or set-containment check?

You will also need to be careful about your overall algorithm. In particular, ***avoid repeatedly iterating through all of*** **`data`**. For example, consider the following graph, with a path highlighted:



Here we've started from Kevin Bacon and successfully expanded out our search until we got to the actor we were looking for. What do we need to keep track of during our search if we want to get the path without looking for the actor again?

When you have implemented your function and it passes your tests, use it to answer the question below:

According to the `large.json` database, what is the path of actors from Kevin Bacon to Bruno Bergonzini? Enter your answer as a Python list of **actor names** below:

This question is due on Friday September 20, 2019 at 04:00:00 PM.

5.2) Arbitrary Paths

What we've done so far is pretty good, but it raises an important question: what makes Kevin Bacon so special? So far, everything we've done has centered around him. Let's expand things a bit and find the path that connects two *arbitrary* actors to each other.

Complete the definition of `get_path` in `lab.py`. The function should take three arguments in order:

- The database to be used (the same structure as before),
- Two IDs representing actors

Your function should produce a list of actor IDs (any such shortest list if there are several) detailing a path from one actor to the other.

Add at least one test case for `get_path` to `TestTiny` based on the contents of the `tiny.json` database. It should find the minimal path between two non-Bacon actors.

When you have implemented this function and it passes your tests, use it to answer the question below:

According to the `large.json` database, what is the minimal path of actors from Katherine Griffith to John Carroll Lynch? Enter your answer as a Python list of **actor names** below:

This question is due on Friday September 20, 2019 at 04:00:00 PM.

6) Movie Paths

After completing the work above, you might be interested to know what sequence of movies you could watch in order to traverse the path from one actor to another. For example, to move from Kevin Bacon to Julia Roberts, one could watch movie ID `94671` ("Jayne Mansfield's Car," which connects Kevin Bacon to Robert Duvall) and `18402` ("Something to Talk About," which connects Robert Duvall to Julia Roberts).

Add some code to your `lab.py` to determine the list of *movie names* that connect two arbitrary actors. To this end, we have included the `movies.json` database, which maps movie names to ID numbers.

When you have finished this code, use it to answer the following question:

According to the `large.json` database, what is the minimal path of *movie titles* connecting Lucas Denton to Anton Radacic? Enter your answer as a Python list of **movie names** below:

This question is due on Friday September 20, 2019 at 04:00:00 PM.

7) Code Submission

When you have tested your code sufficiently on your own machine, submit your modified `lab.py` below.

When submitting `lab.py`, the server will run the tests and report back the results (including timing) below.

On the server, we will run several additional tests, comprised of the following:

- we will test that `get_actors_with_bacon_number` can handle arbitrarily large Bacon numbers (not just $n \leq 6$)
- we will test `get_actors_with_bacon_number` with large numbers as input to make sure your code stops looking for actors as soon as we know there will not be any actors with the given Bacon number (instead of looping all the way to those large numbers)
- we will test `get_path` on a few pairs from the `large.json` database
- we will test `get_path` with a large Bacon number
- we will test `get_path` on a pair for which a path does not exist (but for which both actors are in the database)

Submit your `lab.py` in the box below:

Select File `No file selected`

This question is due on Friday September 20, 2019 at 04:00:00 PM.

8) Checkoff

Once you are finished with the code, please come to a lab session or office hour and add yourself to the queue asking for a checkoff. **You must be ready to discuss your code in detail before asking for a checkoff.** Since the clarity of your code will be evaluated as part of the checkoff, you may wish to take some time to comment your code, use good variable names, avoid repetitive code (create helper methods), etc.

Be prepared to discuss:

- Your additional test cases in the `TestTiny` class.
- Your implementation of `get_actors_with_bacon_number`.
- Your implementation of `get_path` and `get_bacon_path`.
- How you computed paths of movies that connect two actors.

8.1) Grade

You have not yet received this checkoff. When you have completed this checkoff, you will see a grade here.