

Degrees

Write a program that determines how many “degrees of separation” apart two actors are.

```
$ python degrees.py large
Loading data...
Data loaded.
Name: Emma Watson
Name: Jennifer Lawrence
3 degrees of separation.
1: Emma Watson and Brendan Gleeson starred in Harry Potter and the Order of the Phoenix
2: Brendan Gleeson and Michael Fassbender starred in Trespass Against Us
3: Michael Fassbender and Jennifer Lawrence starred in X-Men: First Class
```

When to Do It

By Monday, January 1, 2024 at 6:59 AM GMT+2

How to Get Help

- 1. Ask questions via Ed!
- 2. Ask questions via any of CS50's communities!

Background

According to the Six Degrees of Kevin Bacon game, anyone in the Hollywood film industry can be connected to Kevin Bacon within six steps, where each step consists of finding a film that two actors both starred in.

In this problem, we’re interested in finding the shortest path between any two actors by choosing a sequence of movies that connects them. For example, the shortest path between Jennifer Lawrence and Tom Hanks is 2: Jennifer Lawrence is connected to Kevin Bacon by both starring in “X-Men: First Class,” and Kevin Bacon is connected to Tom Hanks by both starring in “Apollo 13.”

We can frame this as a search problem: our states are people. Our actions are movies, which take us from one actor to another (it’s true that a movie could take us to multiple different actors, but that’s okay for this problem). Our initial state and goal state are defined by the two people we’re trying to connect. By using breadth-first search, we can find the shortest path from one actor to another.

Getting Started

- Download the distribution code from https://cdn.cs50.net/ai/2020/x/projects/0/degrees.zip and unzip it.

Understanding

The distribution code contains two sets of CSV data files: one set in the large directory and one set in the small directory. Each contains files with the same names, and the same structure, but small is a much smaller dataset for ease of testing and experimentation.

Each dataset consists of three CSV files. A CSV file, if unfamiliar, is just a way of organizing data in a text-based format: each row corresponds to one data entry, with commas in the row separating the values for that entry.

Open up small/people.csv. You’ll see that each person has a unique id, corresponding with their id in IMDb’s database. They also have a name, and a birth year.

Next, open up small/movies.csv. You’ll see here that each movie also has a unique id, in addition to a title and the year in which the movie was released.

Now, open up small/stars.csv. This file establishes a relationship between the people in people.csv and the movies in movies.csv. Each row is a pair of a person_id value and movie_id value. The first row (ignoring the header), for example, states that the person with id 102 starred in the movie with id 104257. Checking that against people.csv and movies.csv, you’ll find that this line is saying that Kevin Bacon starred in the movie “A Few Good Men.”

Next, take a look at degrees.py. At the top, several data structures are defined to store information from the CSV files. The names dictionary is a way to look up a person by their name: it maps names to a set of corresponding ids (because it’s possible that multiple actors have the same name). The people dictionary maps each person’s id to another dictionary with values for the person’s name, birth year, and the set of all the movies they have starred in. And the movies dictionary maps each movie’s id to another dictionary with values for that movie’s title, release year, and the set of all the movie’s stars. The load_data function loads data from the CSV files into these data structures.

The main function in this program first loads data into memory (the directory from which the data is loaded can be specified by a command-line argument). Then, the function prompts the user to type in two names. The person_id_for_name function retrieves the id for any person (and handles prompting the user to clarify, in the event that multiple people have the same name). The function then calls the shortest_path function to compute the shortest path between the two people, and prints out the path.

The shortest_path function, however, is left unimplemented. That’s where you come in!

Specification

An automated tool assists the staff in enforcing the constraints in the below specification. Your submission will fail if any of these are not handled properly, if you import modules other than those explicitly allowed, or if you modify functions other than as permitted.

Complete the implementation of the shortest_path function such that it returns the shortest path from the person with id source to the person with the id target.

- Assuming there is a path from the source to the target, your function should return a list, where each list item is the next (movie_id, person_id) pair in the path from the source to the target. Each pair should be a tuple of two strings.
 - For example, if the return value of shortest_path were [(1, 2), (3, 4)], that would mean that the source starred in movie 1 with person 2, person 2 starred in movie 3 with person 4, and person 4 is the target.
- If there are multiple paths of minimum length from the source to the target, your function can return any of them.
- If there is no possible path between two actors, your function should return None.
- You may call the neighbors_for_person function, which accepts a person’s id as input, and returns a set of (movie_id, person_id) pairs for all people who starred in a movie with a given person.

You should not modify anything else in the file other than the shortest_path function, though you may write additional functions and/or import other Python standard library modules.

Hints

- While the implementation of search in lecture checks for a goal when a node is popped off the frontier, you can improve the efficiency of your search by checking for a goal as nodes are added to the frontier: if you detect a goal node, no need to add it to the frontier, you can simply return the solution immediately.
- You’re welcome to borrow and adapt any code from the lecture examples. We’ve already provided you with a file util.py that contains the lecture implementations for Node, StackFrontier, and QueueFrontier, which you’re welcome to use (and modify if you’d like).

How to Submit

You may not have your code in your ai50/projects/2020/x/degrees branch nested within any further subdirectories (such as a subdirectory called degrees or project0a). That is to say, if the staff attempts to access https://github.com/me50/USERNAME/blob/ai50/projects/2020/x/degrees/degrees.py, where USERNAME is your GitHub username, that is exactly where your file should live. If your file is not at that location when the staff attempts to grade, your submission will fail.

1. Visit this link, log in with your GitHub account, and click Authorize cs50. Then, check the box indicating that you’d like to grant course staff access to your submissions, and click Join course.
2. Install Git and, optionally, install submit50.
3. If you’ve installed submit50, execute

```
submit50 ai50/projects/2020/x/degrees
```

Otherwise, using Git, push your work to https://github.com/me50/USERNAME.git, where USERNAME is your GitHub username, on a branch called ai50/projects/2020/x/degrees.

4. Submit this form.

You can then go to https://cs50.me/cs50ai to view your current progress!

Acknowledgements

Information courtesy of IMDb. Used with permission.