AI with Python: Search

Programming Assignment 1

Summer term 2025

Overview

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

Example:

```
$ 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
```

Submission

- Form groups of **up to 5 students**.
- **Submission is mandatory** for all students.
- Submit your solution via **Moodle**.
- Indicate to which group the submitted solution belongs to.
- Submission deadlines:
  - WWIMBIT22A: **20.05.2025, noon**
  - WWIMBIT22BB: **22.05.2025, noon**

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

1. Download the distribution code provided with the assignment, which includes:
    - `degrees.py`
    - `util.py`
2. Download the two folders: `large/` and `small/` containing the datasets provided in the folder `Data for programming assignments/`
3. Place all files and folders into a single project directory on your local machine.
4. Open a terminal and navigate to your project directory and create a virtual environment in Python:
5. Install required libraries if necessary.

Understanding the Data

There is a total of six datasets in CSV format: three in the `large/` directory and three in the `small/` directory. `large/` and `small/` contain the same file structure, but `small/` is a simplified dataset intended for testing and experimentation.

Each folder contains the following three files:

- `people.csv`: Contains information about individuals, including a unique `id`, their `name`, and `birth` year. IDs correspond to entries in IMDb's database.
- `movies.csv`: Contains movies, each with a unique `id`, a `title`, and a release `year`.
- `stars.csv`: Represents relationships between people and movies. Each row contains a pair of values: a `person_id` and a `movie_id`, indicating that the person starred in the movie.

For example, the entry `102,104257` in `stars.csv` indicates that the person with ID `102` (Kevin Bacon) starred in the movie with ID `104257` ("A Few Good Men").

Understanding the Code

The core logic of the project is implemented in `degrees.py`. At the top of the file, several key data structures are defined to store the information loaded from the CSV files:

- `names`: A dictionary that maps lowercase names to a `set` of person IDs. This accounts for cases where multiple people share the same name.
- `people`: A dictionary mapping each person ID to another dictionary containing the person's `name`, `birth year`, and a `set` of movie IDs they starred in.
- `movies`: A dictionary mapping each movie ID to a dictionary containing the movie's `title`, `release year`, and a `set` of person IDs representing its stars.

These data structures are populated by the `load_data` function, which reads the CSV files and loads their contents into memory.

The program's main function performs the following tasks:

1. Loads the dataset from the specified directory (either `large/` or `small/`), can be specified by a command-line argument.
2. Prompts the user to input two names.
3. Uses `person_id_for_name` to resolve each name to a unique person ID. If a name corresponds to multiple people, the function prompts the user for clarification.
4. Calls the `shortest_path` function to compute the shortest connection between the two individuals.
5. Prints the sequence of movies and people involved in that connection.

The `shortest_path` function itself is left unimplemented. Your task is to complete this function so that it correctly determines the minimum number of connections (degrees of separation) between two actors.

Specification

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. The file `util.py` contains the lecture implementations for `Node`, `StackFrontier`, and `QueueFrontier`, which you're welcome to use (and modify if you'd like).