CS 245 - Assignment 2 Fall, 2019

Assignment 2 - Six Degrees of Kevin Bacon

The goal of this assignment is to demonstrate your general use of data structures and particular mastery of Graphs and their algorithms. This will be accomplished by reading and manipulating data about actors[1] and the possible associations by the movies they have acted in.

Background

Six Degrees of Kevin Bacon is a knowledge game based on the "small world" or "six degrees of separation" concept. It is possible for a large network to be linked by a limited number of steps through one or more who are well-connected. As it turns out, Kevin Bacon may be less prolific than others — Meryl Streep or Ben Kingsley, for example — so this game may belie the biases of the inventors.

In this assignment, your implementation will read data about movie credits. Once read, your implementation will allow the user to search for pairs of actors and connect them using one of the shortest possible paths between pairs.

Requirements

Your implementation will read data from a file representing actors, allow the user to search this data for individual actors and find the possible connections between pairs of actors through the movies in which they have acted.

Implementation requirement 1: Read the source file
Your implementation must read the file `tmdb_5000_credits.csv`, which is available for download through the Kaggle's TMDB 5000 Movie Dataset[2]: https://www.kaggle.com/tmdb/tmdb-movie-metadata# or via this link on Google Drive. According to the Kaggle site, this data is more accurate than what is available through IMDB. In order to download this file, you will need credentials for the Kaggle site.

In your implementation, you must expect the user to provide the location of this file on command line. For example, if your program is called A3.java, the user is expected to launch your implementation with:

```
java A3 /tmp/data_files/tmdb_5000_credits.csv
```

---

[1] Here, I use "actors" in a gender neutral sense — i.e. this refers both to female and male actors.
[2] By my calculation, this dataset contains fewer than 5000 movies. However, it does contain about 100,000 actor records.

The `tmdb_5000_credits.csv` file is officially a CSV file; however, two of the fields (cast, crew) are in JSON format. As such, it may be useful to use a JSON parser such as the one found in JSON-simple. If your implementation uses a JSON parser other than JSON-simple, you must indicate so in your submission comments along with instructions on how to install the package you use.

Implementation requirement 2: Find shortest path
Your implementation must find one of the (possibly many) shortest paths between pairs of actors. The user is expected to provide actors by their names. If the actors' names are not found, your implementation must indicate so. When a valid pair of actors are found, your implementation must find one of the shortest paths between the pair. If no such path exists, your implementation must indicate this. Here is an example, with the output of the implementation in bold:

```
Actor 1 name: David Guy Brizan
No such actor.
Actor 1 name: Hailee Steinfeld
Actor 2 name: Abigail Breslin
Path between Hailee Steinfeld and Abigail Breslin: Hailee Steinfeld -->
Abigail Breslin
Actor 1 name: Asa Butterfield
Actor 2 name: Paul Dano
Path between Asa Butterfield and Paul Dano: Asa Butterfield --> Abigail
Breslin --> Paul Dano
```

By default, the names of actors are case sensitive. For example, a search for actor "Benicio Del Toro" will yield no results by default although there are two dozen records of his work in the data file. Ensure that your search works for actors' names regardless of the uppercase/lowercase characters provided by the user.

Submission

Submit the source code for your implementation on Canvas. You may also add any comments in a README file (text, PDF or Word document) to help the grader understand or execute your implementation.

Submission

Submit the following on Canvas:

1. The Java source code for your implementation conforming to the Implementation Requirements. This must include your search tree and trie classes and a main file.
2. Your analysis of the running time for the Analysis Requirement in a README file (text, PDF or Word document).
3. Your optional comments in a README file (text, PDF or Word document) to help the grader understand or execute your implementation. Describe any extra credit submission(s) here as well.
4. Your optional extra credit implementations.

Grading

Your grade for this assignment will be determined as follows:

- 55% = Implementation: your class implementations must run successfully with the source files and data provided. It must produce the expected results, a sample of which appears in the Implementation section above. Any deviation from the expected results results in 0 credit for implementation.
- 20% = Decomposition: in the eyes of the grader, your solution follow the suggestions above or otherwise must represent a reasonable object-oriented and procedural decomposition to this problem.
- 15% = Efficiency: your code must consistently use the most efficient data structures for the task and must consistently use the most efficient algorithms on those data structures.
- 10% = Style: your code must be readable to the point of being self-documenting; in other words, it must have consistent comments describing the purpose of each class and the purpose of each function within a class. Names for variables and functions must be descriptive, and any code which is not straightforward or is in any way difficult to understand must be described with comments.