COSC102 Spring 2015

DS 3: IMDb Movie Homework 7

Due date: **Wednesday, April 22nd at 11:55 p.m. on Moodle**

In this assignment, you will use hash maps and sets to implement efficient analysis of a dataset exported from IMDb, the Internet Movie Database. The dataset contains movies and actors appearing in those movies.

The Dataset

The provided files contains several text files with movie listings. Each of the files has the following format:

- Each line corresponds to one movie from IMDb.
- Each line contains the following fields, separated by forward slashes (`'/'`):
  - The first field is the movie title followed by the year in parentheses. You should treat this whole substring as the name of the movie.
  - The rest of the lines contains names of actors who appeared in that movie. Normally they are in the format of last name, a comma, and then first name. You don't have to change this order when parsing the file. Just assume that each substring between slashes represents the name of an actor, and that actor's name should appear the same way for other movies in which that actor appears.
  - The last actor name for each movie is not followed by a slash, just the end of the line.

Setup

The provided `zip` file contains

1. `Movies.java` which includes the main console program for this assignment.
   You shouldn't modify or submit this file.
2. `MovieGraph.java` which is the template for the data structure you have to code. This file should be submitted.
3. `movies_*.zip`: two `zip` files, one with files saved in Unix line endings and Unicode (UTF-8) encoding, the other with files saved in Windows line endings and Windows Latin 1 (ANSI) encoding.

Use the files for the platform you are working on. Mac users should be able to use either, but might prefer the Unix version.

The `cast.*.txt` files contains data sets for the assignment. There are files of multiple sizes: ranging from about 500 KB up to about 11MB. Start developing your code testing with the smaller files (< 2MB) and try larger files later on. Your program should work efficiently even for the largest file (~ 11MB).

The Main Program

`Movies.java` contains the source code for a console program. This program prints a menu of search queries that the user can perform.

When it begins, it instantiates a `MovieGraph` object providing the constructor with the filename given as the first command-line argument. The `MovieGraph` object is the data structure that you have to code. It basically reads the information from the data file, stores it in an appropriate way, and can then answer the various queries from the console program.

The Queries

The `MovieGraph` object needs to answer queries by implementing the following methods.

1. `getMovies`

   This `MovieGraph` method takes as a parameter the name of an actor and returns a collection of all the movies in which that actor has appeared. If the number of such movies is m, then the target running time for this is $O(m)$.

2. `getActors`

   This `MovieGraph` method takes as a parameter the name of a movie and returns a collection of all the actors appearing in that movie. If the number of such actors is a, then the target running time for this is $O(a)$.

3. `getCoStars`

   This `MovieGraph` method takes as a parameter the name of an actor and returns a collection of all the actors that appeared in any movie with the given actor. If the number of movies in which the given actors appears is m and the total number of co-stars is c, then the target running time for this is $O(m + c)$.

4. `isMatch`

   This `MovieGraph` method takes as a parameter the name of an actor and the name of a movie, and returns whether or not the given actor appeared in the given movie. The running time for this is $O(1)$.

5. `getMovieLinks`

   This `MovieGraph` method takes as a parameter two actor names and returns a collection of all the movies in which both actors have appeared as co-stars. The running time of this is $O(a + b)$, assuming that the first actor has appeared in a total of a movies and the second actor has appeared in a total of b movies.

For any of the methods that are supposed to return a collection of strings (any method except #4 above), you can return any data structure in the Java library that supports iteration (more on this below). Thus, the return type is `Iterable<String>`. Hash sets, linked lists, array lists, etc. all implement this interface, so you can create any one of these to store the items you want to return.

Building Your Data Structure

All of the queries (as an example, take #4 above) are built on looking up information associated with keys (movies or actors) quickly. Note that we don't care about order when answering the queries.
This suggests that using hash tables to implement the data structure would be most efficient.

Thus, your code in `MovieGraph` will have

- instance variables for the hash table(s) and any related information you're going to use to store the data,
- the constructor read the file, *in one forward pass, line-by-line*, and store data using those instance variables and
- methods implementing each query call hash-table methods on those instance variables to quickly retrieve and analyze the data relevant to the query.

Remember that, in Java, there are two basic classes built on hash tables, which you should investigate and use:

- java.util.HashMap, which stores (key, value) pairs and
- java.util.HashSet, which stores a collection of keys.

In both cases, keys must be distinct, as those are the items that are hashed and used to look up/insert information. Both of these classes are implemented using generics, so you should specify the types of things you want to store, for example:

```
HashMap<String, Integer> h = new HashMap<String, Integer>();
```

creates a new hash table storing strings as keys and integers as values.

Do not use `java.util.Hashtable` as it is legacy code. `HashMap` and `HashSet` are the current implementations of hash tables in Java that you should use.

Testing

It is recommended that you use the smaller data files for initially testing your code. To run the program, invoke the main method in `Movies.java` and provide the file name as the first command-line argument.

Given all the queries, a good place to start might be to:

1. design what your data structure will look like;
2. implement the constructor that can parse the data files to extract movie and actor names;
3. implement the `isMatch` method (query #4 above) or `getMovies`/`getActors` (query #1 or #2) if they are more obvious to you.

Then, compile and run your (incomplete) program but check the lookups you implemented.

From there, move on to implementing and testing the other queries. Once the design of the data structure is complete and the file parsing is done, the other methods should be relatively short to code and efficient to run.

Submission

Submit two files:

- `MovieGraph.java` and
- a `readme` file in which you explain how the data structures you used permit to obtain the required runtimes (or better) for each of the queries.

DO NOT submit `Movies.java` or any of the data files—they're big, and there's no point in taking up space on Moodle with extra copies of those files. Thanks!