Final Project Report

Jinyi Gao

Project code:

<https://github.com/gaojinyigg/si507_final_project>

package needed: requests, json, csv, webbrowser, pprint

Data sources:

One data is from <https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata> which is a related movie CSV dataset with 5000 movies information and the corresponding credits. In addition, another data source is an api which is used to catch some information of IMDB especially the rating on IMDB and the response is json format (url: <http://www.omdbapi.com/?apikey=9da705df>). Furthermore, as for every movie, itunes api is used to find the link to watch the movie (url: https://itunes.apple.com/search?term=xxx&limit=xxx).

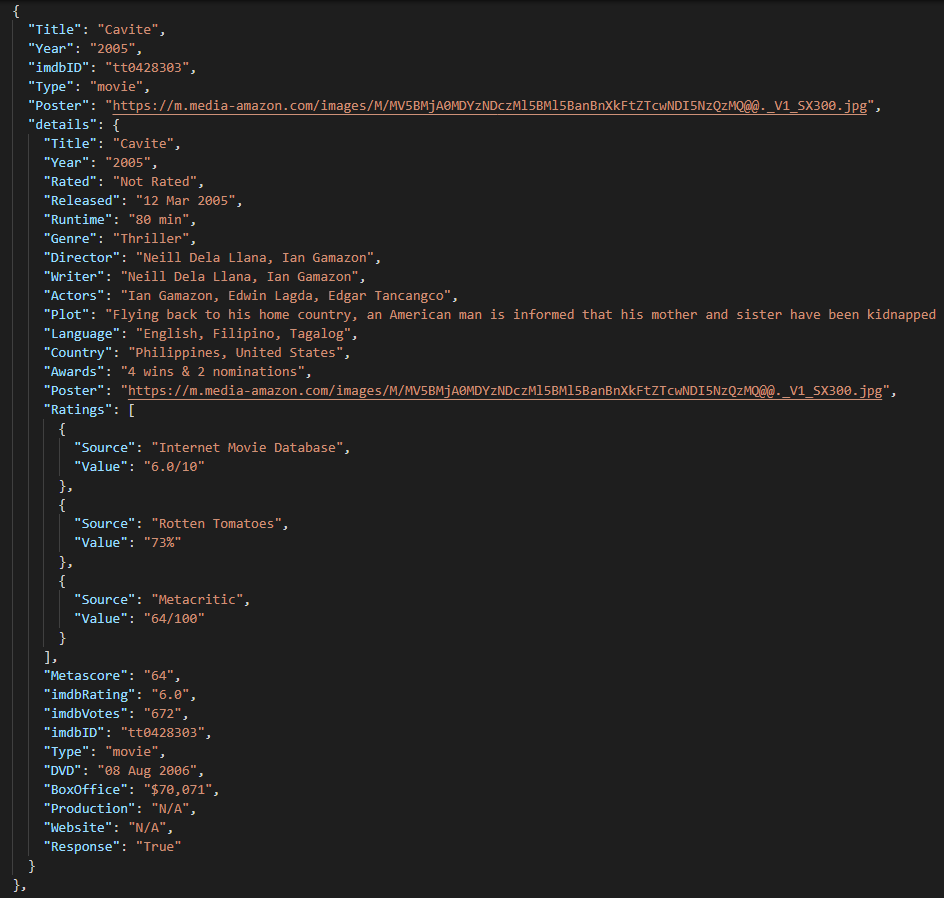
The csv data is accessed just directly by accessing the files and translating csv into json to maintain the consistency of all data. And meanwhile I keep part of the data as one my parameters for api requests, such as using title as a search parameter.

Then for all data, different part of the data will be gathered and form a new data chunk(final data used). Moreover, I access another two data set from caches since sending a request for every movie is too slow (sending a get request to the api and get response for the first time, then save them into the cache for accessing later).

As for the first data, attributes are as follows: *budget, genres, homepage, id, keywords, original\_language, original\_title, overview, popularity, production\_companies, production\_countries, release\_date, revenue, runtime, spoken\_languages, status, tagline, title, vote\_average* and *vote\_count.*

Additionally, the second data from api is like figure 1 and itunes’ data is just like those in project 1.

For the first data set, the information missed in the response of api will be kept such as *budget.* The mainly attribute from the first data set is *title* since it can be used as the search term for the api. Then the response will be added as a new key of the dictionary named *details.* The most important attributes of the response are *Genre, Actors* and *Directors.* These three attributes are used to build different kinds of graphs. In addition, *imdbRating* is used to select some movies from the set as the candidates for user to check the connectivity of two movies. Furthermore, as for the response of iTunes api, *collectionViewUrl* is used to let user know where to watch the movie.



Firgue 1 attributes from api

For the cache part the evidence are as follows:



Figure 2 the functions implemented for caching

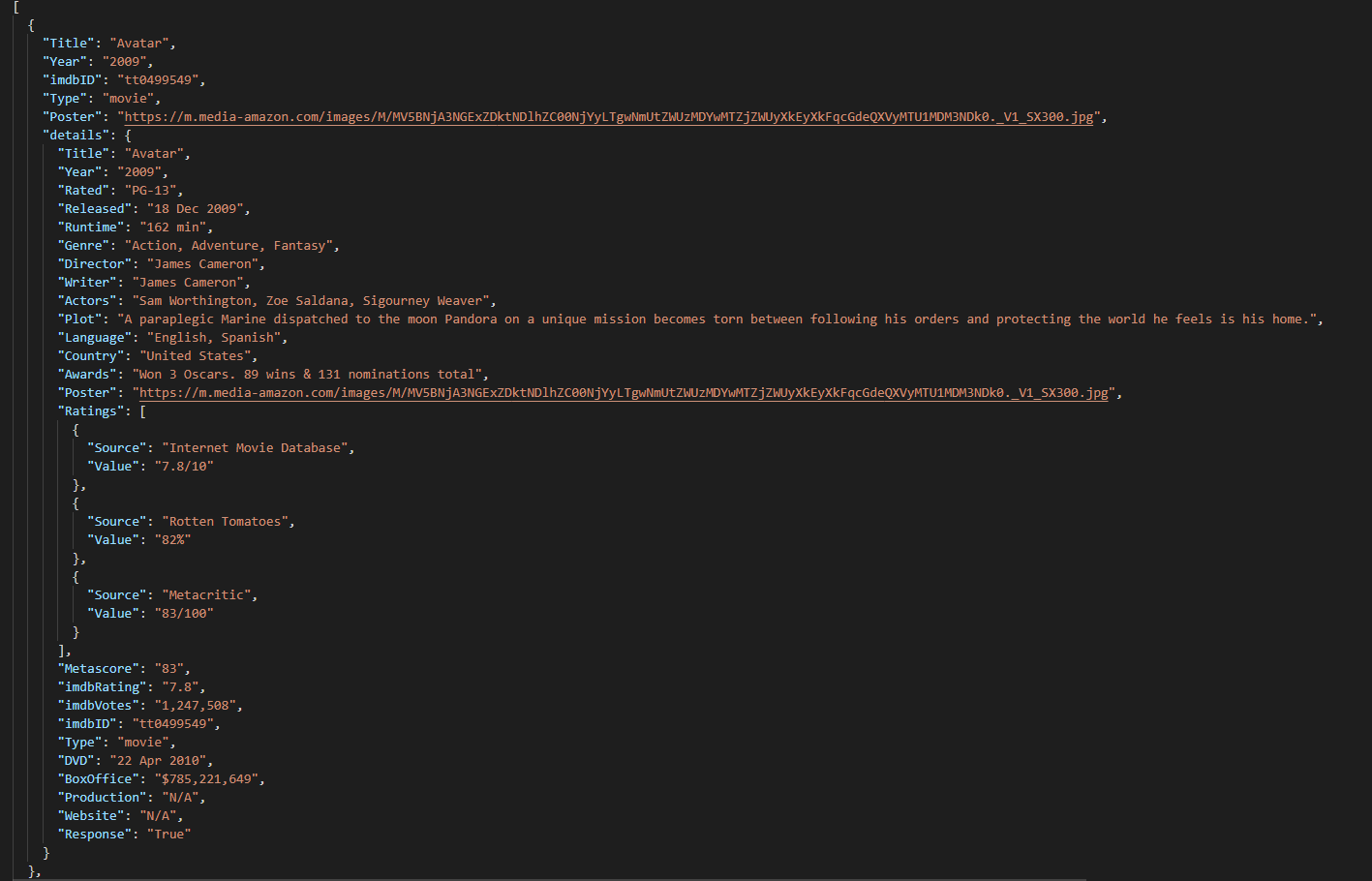
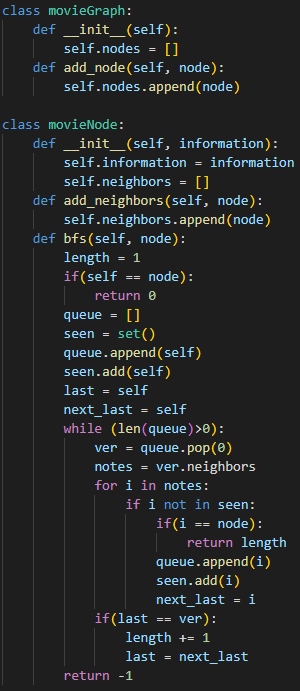
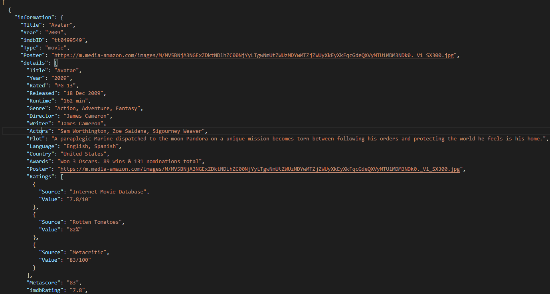


Figure 3 part of chunks of cache data

Data Structure:

A graph is used to organize the data. Every movie will be a node. And the edge between nodes will depends on the two kinds of relationships. According to different options for the user, four kinds of graph will be built. As for genres, if two movies with the same genres, they will be added as neighbors of both. As for credits, if a character appeared in another movie or the director of two movies are the same, they will be added as neighbors of both. It is similar as other two options(writer and language).

Graph.py generates the data structure. Class *moviegraph(*for whole picture of the graph*)* and *movienode(*a movie with information and neighbors*)* are designed. Main work of building graph is finished by function *buildgraph(select)*. According to user’s options, different kinds of trees will be built. Graph.json file shows the json format of the datastructure and part of the data designed is shown as figure 4.

Firgure4 json format of json file and the class of graph

As for reading the json file of data structure to the system, because one movie node contains too much information and runtime of translating it into data structure is almost the same as rebuilding the data structure, for system the structure will be rebuilt based on the user’s options. But *read\_graph.py* also implements the method used to read the json file.

Interaction and Presentation:

The technical way used in interaction is command line. The instructions are as follows.

At the beginning, there will be four options for you build two different kinds of graph based on different connection. (genre, character, writer, language). You must type "0", "1", "2", "3", "4" to continue.

Then there will be three options. "1" is to check the most connected movie (the movie with most neighbors in the graph) and "2" is to check the most non-connected movie. In addtion, "3" is to check the connectivity between two movies.

If you choose "1" and "2", the basic information will be shown by the command line and you can also choose to look at the poster of the movie("1") or to watch the movie on itunes directly("2"). If you choose to watch, the webbrowser will open the corresponding link and show you the results.

If you choose "3", the system will provide you with a movie data set and you can choose two of them to check the connectivity (how many steps needed to find another movie ).

Once you finished one of these three options, you can try one more time.

For every step, "0" is for exit the whole system.

For different kinds of graphs, the system will show different results.