Kevin Bacon Project Write-Up

Ashley Chen, CS 330 May 5, 2017

Overview

This project allows users to explore and understand the connections between actors in box-office movies through three features: Visualizer, Search, and Game. The graph is essentially made with nodes of unique actors and edges of movies. The web app is able to find connections between actors using all three features. The visualizer shows a graph that is interactive. The calculator prints the connections between actors. The game allows users to guess what connections they can make with two generated actors in seven degrees or less.

This web app was created using HTML, CSS, and Javascript's Node.js and D3.js. Information about movies and actors was collected from an API called MovieDB.

Project Implementation

The implementation of this project is split into these parts: data collection, data parser, visualizer, search/calculation, and game. These features are connected using HTML and external Javascript links. Most of these functions are executed by the click of a button, but functions are also called in a individual Javascript file.

Data Collection

In order to get the information I wanted for this web app, we needed to collect information about actors and movies. I chose MovieDB and a Node.js wrapper for the API. Because it's a RESTful API, the information comes in JSON objects. After testing its callbacks, it turns out that MovieDB is only able to get actors through list of movies. The API also limits the callbacks you can make. To go around this, I would print the objects of movies and its cast in 10 movie increments. MovieDB makes it easy to iterate through movies as movies are called by id numbers, which are numerical. The print statements would go into a JSON file, which was set up as an output console file in my IDE.

Data Parser

To optimize the performance of the code, I created three maps; each with a specification of a string as the key and a set of strings as the value. These maps link names of actors to a set of connected actors, names of actors to a set of movie names, and the movie ID to the name of the movie.

Visualizer

The visualizer uses a JSON file that specifies nodes and links. Nodes are created from the list of actors that was made by the map. Links are taken from the actor to actor map. The graphical interface of the visualizer is made with D3.js and uses a specific function called force, which is able to constantly update the coordinates of each node, name, and link in the HTML. Attributes are also added in D3 using its built-in functions for appending HTML styles.

Search/Calculation

Between two actors in the graph, I created three functions to find either a path of connections or not a path at all. I tested using a breadth-first search, depth-first search, and a recursive function. I found breadth-first search to be the best fit for this graph. Along with implementing this search, I also created a function that returns the movies which are associated with each connection, so when the search prints the connections, the app can also print the movies each pair of actors were both in.

Game

The game randomly selects two actors from the list of actors in the actor to actor map. The second actor is selected particularly by traversing the first actor's connection between 2-6 times. A user can enter up to 6 names to make a connection. To check whether the user's answer is correct, a function goes through the traversal and returns whether or not the destination matches the second actor or a connection is broken.

Project Execution

The code can either be accessed locally or on an Apache Server. The structure of the code is very simple; by opening index.html, the web app will link all scripts files in a browser. For convenience, I've deployed the project onto Heroku and can be accessed through here. If modifications are made to the project, you must empty your cache to push it as it has caused problems in the past with updating stylesheets and scripts.

Future Work

There are a lot of improvements I can make to this app. First, I can migrate my backend work to all Node.JS for simplicity and better performance. This way, I dont' have to prepare the data prior to opening the web app. Instead, it can stream in information from the API and have more options for the users to pick from than the predetermined data. Another implementation I can improve on would be the search for connections between actors. As the search made by the computer does not always find connections 7 degrees or less, implementing either a function that can do this or find the shortest path will help improve the prupose of the feature.