CelebriGator Project Report

# Administrative

* Team Name: Aman’s A-Listers
* Team Members: Pablo Hernandez-Perretti, Joseph Fleming, Samuel Falzone
* GitHub URL: <https://github.com/bizononlee/Project-3>
* Video Link: INSERT LINK HERE

# Extended and Refined Proposal

## **2A. Problem**

The problem our project is designed to solve is that of understanding the entertainment industry. Specifically, our project enables users to determine the strength of relationships between actors based on overlaps in relevant work history. In doing so, our project also enables users to make more educated decisions regarding which new titles/genres to explore based on the interconnectedness of new actors with those they know and love. Moreover, our project will be solving the question of which SSSP algorithm, between Dijkstra’s Algorithm and the Bellman-Ford Algorithm, performs best in a social-network type of application such as this one. Additionally, our project includes a comparison of DFS and BFS in detecting whether any connection exists between two actors (based off mutual connections in work history); providing simpler and computationally cheaper output to those who would like a binary characterization of the relationship between two actors. We have utilized parallel data structures and all other programmatic constructs to isolate algorithmic performance for our comparison.

## **2B. Motivation**

## Cinema fanatics who appreciate the brilliant performances in their favorite films are often searching for new content to consume that is in line with their interests. Doing so typically entails the costly (time and money) trial and error of the various new films or performances that circulate the forefront of social media platforms. Our project enables fans to swiftly verify the probability that any given movie will align with their interests based on confirmation of the connectedness of leading cast members. Moreover, countless cinema fanatics keep themselves up to date on the latest celebrity gossip; our project enables them to better understand the professional underlay that likely drove many popular celebrity partnerships and feuds.

## From a technical perspective, our project enables future software engineers to directly compare the performance of two popular SSSP algorithms (Dijkstra’s & Bellman-Ford) in a simplified social network graph that utilizes real world data. Our project’s stopwatch functionality will enable a direct comparison of the performance of two parallel implementations of these algorithms. This knowledge is valuable context for those deciding which SSSP algorithm is optimal for their prospective application.

## **2C. Features Implemented**

## Our project implements two primary features: binary characterization of the relationship between two actors, and calculation of the exact strength of the relationship between two actors. Binary characterization of the relationship between two actors entails utilizing both BFS and DFS to determine whether there is any relationship between the two passed in actors. Results from both the BFS and DFS algorithms are timed, and the efficiency comparison will be outputted to users. Calculating the strength of the relationship between two actors entails running both the Dijkstra’s and Bellman-Ford SSSP algorithms to procure the shortest distance between to actors. Across both features, we have implemented stopwatch functionality that will assess the efficiency of all four algorithms for every unique query. Our project does not have a graphical user interface (GUI), users can only interface with our application through the CLI by typing simple commands or the names of actors they’d like to compare.

## **2D. Data Description**

## Our database of choice for this project can be found at <https://datasets.imdbws.com>, and is downloadable via the hyperlink “name.basics.tsv.gz”. Our database of choice is one of several public databases maintained my International Movie Database (IMDB). Moreover, our database can be visualized as a table where rows would represent a given actor and columns would represent various attributes of each actor. Specifically, there are over ten million actors included in this database, each of which having six listed attributes in the database. To better align with the restrictions and purpose of this assignment, we altered the database for our own personal dataset to include only the names and most notable previous roles held by roughly ninety-four thousand actors (93,968). We reduced the number of entries in our dataset by sorting actors out who: had passed away, were born prior to 1950, or had less than 4 notable film appearances throughout their career.

## **2E. Tools, Languages, APIs, & Libraries**

## The entirety of the programming for our implemented algorithms and graph API relies on nothing other than the C++ STL. On the other hand, the source code for manipulating the public IMDB database into our usable dataset is written in Python and utilizes various public libraries and tools to manipulate data more efficiently at such a large scale. Specifically, the source code for generating a usable dataset implements both pandas [1] and NumPy [2], which are extremely popular public libraries containing useful mathematical and data manipulation operations. Additionally, the database alteration source code takes advantage of useful tools such as Jupyter Notebooks [3] and Conda [4], which are a common editor and file management system, respectively.

## **2F. Data Structures & Algorithms**

## The algorithms at the center of our project’s functionality are Breadth-First Search (BFS), Depth-First Search (DFS), Dijkstra’s Algorithm, and the Bellman-Ford Algorithm. Each of the previously mentioned algorithms are implemented exclusively utilizing data structures within the C++ STL. The previously mentioned algorithms rely on several common containers in the C++ STL, including unordered\_map, vector, priority\_queue, stack, queue, and set. Each of the previously mentioned have unique performance metrics for varying tasks, when and where a data structure was implemented into the various algorithms will be reflected in the complexity analysis component of this report. Moreover, the graph API itself takes advantage of the unordered\_map data structure to store an adjacency list, as well as to support rapid access of an actor ID based on a given name and vice versa.

## **2G. Distribution of Responsibilities**

## The distribution of responsibilities for the completion of this project was as follows: Pablo Hernandez-Perretti was responsible for generating the dataset, implementing the graph API, implementing DFS, and writing sections 1 and 2 of this report; Joseph Fleming was responsible for implementing Dijkstra’s algorithm, implementing BFS, implementing the application’s CLI UI, and writing sections 3 and 4 of this report; Samuel Falzone was responsible for implementing the Bellman-Ford algorithm.

# Analysis

## **3A. Alterations Since Proposal**

## Format

## **3B. Complexity Analysis**

## Format

# Reflection

## **4A. Group Experience**

## Format

## **4B. Challenges**

## Format

## **4C. Lessons**

## Format

# References

[1] pandas, Data Manipulation. <https://pandas.pydata.org/>

[2] NumPy, Mathematical Functions. <https://numpy.org/>

[3] Jupyter Notebooks, Text Editing. <https://jupyter.org/>

[4] Conda, File Management System. <https://conda.io/>