/*
* Name: Megan Chu
* PID: A12814536
* Date: May 25, 2017
* CSE 100 PA4 C++ Kevin Bacon
*/

I created a new “Actor” class for nodes for each actor, a “Movie” class to define the title and year of a unique movie. Within the Actor class, I had a string storing the actor name, and a vector storing pairs of connections to other actors (had pointer to another actor, and pointer to overlapping movie). The reason I used the pairs was for easy access to the movie information after I had found the shortest path through the actor nodes. I also had a singular pointer to the “previous” actor to use to track the shortest path between two actors. I also had a bool variable to check if this particular actor had been traversed when looking for the shortest path. Within the movie class, I had a string for the movie title, and an int for the movie year. I also had a vector containing pointers to all actors in the movie. The vector of actors in the movie class was used mainly to created the connections between the actor nodes, with this implementation, I could easily add connections from one actor to all other actors in the same movie. I think overall I store more information than necessary in the Actor and Movie classes, and this takes up extra memory, however, when the code is running I am able to more eloquently obtain the the data I need, and output it cleanly.

My code previously ran for ~70 seconds for the full movie cast compared to the ~4 seconds the reference solution took. I was able to ask a tutor, and I cut down most of my runtime by changing the vectors that stored my actor nodes and movies to unordered_maps or hashmaps. I was able to end up with a runtime of less than 3 seconds. This was because I did not have to iterate through all movies and actors to check if I had created a new object for them.

To implement the weighted connections for the final submission, I used a priority queue to decide which nodes to traverse before others from a given node. This gave me the shortest weighted path.

To do the bfs and union find connections, I add edges/unioned sets based on the year starting with the oldest movie year. For bfs I added edges between actors for movies older than the given year. For union find, I merged actors in movies older than the given year to my “main set” in which I would find actors and movies.

"Actor connections running time":

Which implementation is better and by how much?

Although I did not get to complete UnionFind perfectly, in theory Union Find should be faster, and it also ran much faster than my bfs even with merging all the nodes for every test case. When I tested with movie casts.tsv and test pairs, bfs took at least 500 milliseconds whereas UnionFind took less than 100 milliseconds to union all the movie sets for every pair. I believe the path compression through setting each node's parent to the true topmost parent cuts down the run time drastically.

When does the union-find data structure significantly outperform BFS (if at all)?

I believe unionFind outperforms BFS for large data sets, as in data with many nodes. This is because of the aforementioned path compression which allows near constant time to check if two actors have been connected vs. the "length of path" (possibly going through all nodes) time it takes for bfs to check for a connection.

What arguments can you provide to support your observations?

As mentioned above, the path compression keeps runtime stable as there are more and more nodes in the dataset. However, the more nodes in bfs, the more exponentially longer it will take to traverse connections especially if there are many connections as well.