Graph Design Analysis

1. Describe the implementation of your graph structure (what new classes you wrote, what data structures you used, etc)

We designed three different classes, ActorNode, MovieNode, and Edge class

Our ActorNode class consists of vector of Edge* that links the actors to movies, and the object being created on the heap.

Our MovieNode is created on the heap, but there are pointers (MovieNode*) stored in the vectors on the stack for easy access from the map.

Our Edge class has a movieNode and actorNode to define the relationships between the actors and movies(title,year), also created on the heap.

Disjoint set was added for the final submission that uses the pointers to build the up-tree that is stored in the map of ActorNode*.

2.Describe *why* you chose to implement your graph structure this way (Is your design clean and easy to understand? what are you optimizing?, etc)

We choose to use vectors inside our Node classes because it would be quicker to access from the MovieNode and ActorNodes, it would automatically give us O(n) operations such as iterating through every element and creating links between them.

We think that using maps for our MovieNode and ActorNode would give us the optimized runtime, since the insertion and find average case runtime analysis is O(1).

We also used priority queue for BFS and Dijkstra which can be used for weighted and

unweighted mode. The Pathfinder algorithm using a stack container that can traverse back to find it's path.

3. Compare the run times of each implementation on a file containing actor pair pairs that you generate yourself. See how the run times compare when you repeat the same query multiple times. The file should be in the same format as test_pairs.tsv.

a) Which implementation is better and by how much?

The average running analysis for ufind, after 10 trials with using the same actor 100 times, came out to be 350 ms, but for running the algorithm 10 times using 100 different actors came out to be 1800 ms.

The average runtime of bfs after 10 trials with using the same 100 actors came out to be 600 ms, but for running the algorithm 10 times using 100 different actors came out to be 2500 ms.

So the ufind is much faster than bfs for the the particular data that we are using, however, it's possible that bfs outperform ufind, depending on the code structure and data used.

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

It depends on the size of the file, because for small files, BFS can be even faster because it doesn't need to do the path compression which makes a minimal improvement for small files; however, for bigger files, if we are doing a lot of finding, then union find will be faster due to path compression.

c) What arguments can you provide to support your observations?

We created a file with the same actors 100 times and observed that union find is significantly faster than BFS, because union find's search and insert use path compression and the runtime of search after path compression is O(1), to merge sets. But the worst-case runtime analysis of BFS for connecting the actors and searching to find the target is O(n^2).