Runtime analysis for part 2.

Duwei Wang, A10169673

Qihan Guan, A92413483

Test case: 20 times (both exist, noconnection)

UnionFind: 0.7 BFS: 118.98

Tests case: 20 times (Both exist, connection)

UnionFind: 0.69 BFS: 24.61

Test case: 20 times (none exist)

UnionFind: 0.72 BFS:142.68

Test case: 1 times (both exist, noconnection)

Unionfind – 0.78 BFS: 66.89

Tests case: 1 times (Both exist, connection)

Bfs - 25.2

Unionfind – 0.74

Test case: 1 times (none exist)

0.71 for unionfind

For bfs 66.11

Total: 0.71 for UnionFind, 205.85 for BFS

Overall, UnionFind was always faster than BFS. In all cases, it was much faster and always took under a second. UnionFind gave the superior speed for movie_casts.tsv. Meanwhile the BFS was only fastest when there was a connection and even then the best time was 24 seconds.

Theoretically, a correct implementation of an optimized Union Find should have a runtime of O(mlogn) (where m is number of operations, where n is nodes). This is in contrast with a BFS, which has a runtime of O(N+E) (where n is nodes and e is edges). Assuming that both are run on the movie_casts.tsv, with an edge amount of 4016412 and node amount of 11794, the Union Find should be superior (there are 14252 movies, which can possibly be a proxy for number of operations). Thus the union find data structure theoretically outperforms BFS whenever the $mlogn < E+N$; that is, you have to do a small amount of operations on a number of nodes or if the graph is a very densely connected graph. In our case, union find will always be faster due to the sheer amount of edge. Our BFS build method seems pretty slow in particular, since it didn’t seem to matter for the 1 entry vs 20 by very much at all.