Graph Design Analysis

1. Implementation

ActorNode class, ActorEdge class, Movie class

ActorNode:

1.A string stores the name of actor

2.An int stores the distance for Dijkstra Algorithm

3.A boolean stores the whether the node is visited in order to eliminate duplicated exploration.

4.A hashmap to store edges that includes this actor

5.Two pointers that store the previous edge and previous actor to help find the restore the path and output

6.A hashmap to store movies that this actor has been participated

7.A hashmap to store actors that have been incorporated with this actor

8.An int called rank in order to help disjoint set implementation which store the number of nodes in the set that this actor represents.

9.A pointer that points to its parent node in disjoint set

10.An int that store the first year that this actor appeared to help find actor connections.

ActorEdge:

1. Movieinfo that on this edge
2. Two ActorNode pointers to store two actors on this edge
3. An int that stores weight to help the Dijkstra Algorithm

Movie:

1. A string that stores movie name
2. An int that stores movie year

In ActorGraph class:

1. A hashmap that stores all the global ActorNode in this graph
2. A hashmap that stores all the global Movie in this graph
3. A hashmap that stores all the global ActorEdge in this graph
4. A hashmap to store movies categorized by year

Two different kinds of build graph methods: One is for pathfinder, another is for actorconnections which build graph by specified year

2. Reason

Use hashmap in everywhere to reach optimal runtime.

Use four hashmaps in ActorGraph that store all the informations about the graph.

1. Very fast to access information
2. Very fast to insert information
3. Easy to deallocate: only need to loop through hashmap and free all the pointers, avoid double free issues

All the classes are designed based on real life situations. Highly abstraction because member variables are based on real life and also easy to understand.

Actorconnection Runtime Analysis

Edge case: actor not in graph

Same pair 100 times

num_iteration	bfs runtime
0th	29617212771
1th	27756696513
2th	32967483005
3th	38603648526
4th	38202955271
5th	27351712221
6th	26962769031
7th	31803497676
8th	32058624768
9th	37306330386
average	32263093017

num_iteration	union-find runtime
0th	1214224521
1th	1146621240
2th	734662863
3th	714690851

4th	663912513
5th	1228096908
6th	976039845
7th	1021783805
8th	679507641
9th	599784527
average	897932471

36

Different 100 pairs

num_iteration	bfs runtime
0th	47024199662
1th	43772362982
2th	46529103859
3th	35038345497
4th	37074326043
5th	47154617979
6th	41292672856
7th	47683172007
8th	47765672485
9th	52044767258
average	44537924063

num_iteration	union-find runtime
0th	2828196398
1th	2773938714
2th	2699743353
3th	2674977147
4th	2453047369
5th	2693229899
6th	2225479816

7th	2648359758
8th	2506954900
9th	2701988295
average	2620591565

17

1.

Obviously, union-find is a better choice to do actorconnection.

2.

The union-find is always better than BFS in each case. Especially when the pairs are the same, the union-find is faster than searching different 100 pairs. From the perspective of statistics, when we take the input of 100 same pairs, the Union-Find method takes 1/36 of runtime of the BFS method. And when we take the input of 100 different pairs, the Union-Find method takes 1/17 of runtime of the BFS method.

3.

When the size of the graph or set becomes bigger and bigger, union-find tends to be way much faster than BFS since by using path compression, the runtime is only O(logN). But for BFS, since there are huge amount of edges. In worst case, BFS will go through all the edges, which is O(E) and will consume lots of time. Also, rebuilding the disjoint set is faster than rebuilding graph since for disjoint set, we only need to rebuild the forest whereas for graph, we need to rebuild all the adjacent actors. These are the reasons that the Union-Find method is significantly efficient than the BFS method.

Furthermore, we find out that the runtime with same 100 pairs is much faster than the runtime with different 100 pairs. We think this difference depends on which “same” input we use. In our case, we used the input with a short path and repeated the input for 100 times. So if we used a input with a long path, the “same 100” input might requires a longer runtime. For the pair.tsv pre-provided, all of the 100 pairs are different, so it is an average case of runtime, in general.