Kesong Xie

Graph Design Analysis

1. Graph structure implementation

ActorGraph:

vertices: the *vertices* is a member variable that is of type unordered_map<string, ActorNode*>, the key is the name representation for the actor/actress, and the value is the pointer for the ActorNode object
Reason: O(1) when checking whether a given actor is already in the graph or not, and therefore for speeding up the speed for adding edges for given actor.

movies: the *movies* is a member variable that is of type unordered_map<string, ,Movie*>, the key is the movie as a string representation, and the value is a pointer for a movie object
Reason: O(1) when checking whether a given movie is already in the graph or not, and therefore for speeding up the speed for adding actors to a given movie, which later on can be used to add necessary connections between different actors based on the move title and year.

numberOfEdges: the *numberOfEdges* is a member variable that keep track of the current edges in the graph

ActorNode:
name: the *name* is a member variable that is of type string, which is a name of a given actor

adj: the *adj* is a member variable that is of type vector<ActorEdge*>, which contains the adjacent edges for this given actor

Reason: use the ActorNode struct to encapsulate all the relevant information about the actor, which includes the name and his or her connected actors. Follow object oriented design principle.

Movie:
title: the *title* is a member variable that is of type string, which is a title of a given movie

title: the *year* is a member variable that is of type int, which is the year of a given movie

act_in: the *act_in* is a member variable that is of type unordered_set<string>, which contains the a list of actor names in this movie

Reason: use the Movie struct to encapsulate all the relevant information about the movie, which includes the title and year and all the actors that casted in this given movie. This is beneficial for later on adding edges among all the actors in this same movie and improve the speed for computing.

ActorEdge:
source: the *source* is a member variable that is of type ActorNode*, which is a pointer to a given ActorNode and the source of this edge

dest: the *dest* is a member variable that is of type ActorNode*, which is a pointer to a given ActorNode and the destination of this edge

movie: the *movie* is a member variable that is of type Movie*

Reason: use the ActorEdge struct to encapsulate all the relevant information about the graph's edge, which includes the information of the source vertex and destination vertex. This is beneficial for later on outputting the shortest path.

Kesong Xie

Final Submission:

ActorConnections:

100 Identical Pairs:

BFS Implementation:
Time cost:
8s

Union-Find implementation:
Time cost:
3s

100 Unique Pairs:

BFS Implementation:
Time cost: 128s

Union-Find implementation:
Time cost: 7s

Actor connections running time:

1. Which implementation is better and by how much?
In the case of 100 identical pairs, the running time of Union-Find implementation is better than BFS implementation by 5s

In the case of 100 unique pairs, the running time of Union-Find implementation is better than BFS implementation by 121s

2. When does the union-find data structure significantly outperform BFS (if at all)?
When running unique pair, the union find data structure significantly outperform BFS

3. What argument can you provide to support your observation?
The union find is considered almost constant when performing a find operation while the breath first search performs | V + E |