Homework 1
CX 4140
Hardik Sangwan

1. Design an algorithm to compute the heist-closeness centrality. Write the high- 1level idea, then concrete details and pseudo-code for the algorithm.

With an input of all the different vertices acting as heist nodes, we will iterate through each heist node to compute the closeness centrality for that node. The closeness centrality is simply the inverse of farness, which can be calculated by adding shortest pair distances from the vertex to all other vertices (connected graph). To calculate the farness/shortest distances, Breadth First Search can be used.

Here is pseudocode for the algorithm:
Input: Graph g, VertexSet h
Output: Closeness Centraltiy cHc for all vertices in h

```
For each s belongs to h do
    Q = empty queue
    dist[v] = inf, for all v belongs to h
    Q.push(s)
    dist(s) = 0
    farness(s) = 0
    while Q is not empty do
        v = Q.pop()
        for all w belongs to edges connected to v do
            if dist[w] = inf then
                Q.push(w)
                dist[w] = dist[v] + 1
                farness[s] = farness[s] + dist[w]
    cHc = 1/farness[s]
    g.setclosenesscentrality(v,cHc)
```

2. Prove the feasibility of the algorithm. That is, prove that the algorithm will correctly compute the heist-closeness centrality for each vertex.

The feasibility of the algorithm can be proven by assuming that Breadth First Search is feasible (considered as a lemma). The BFS routine is run inside a for loop, increasing the time complexity by O(V* big-O of BFS). This still computes in polynomial time. Since we are given a connected graph, all vertices are connected to all others and no distance is infinite. Thus a closeness centrality value will be calculated for each vertex.

3. Compute and prove the run-time and space complexity of the algorithm. Identify if different data structures will impact the complexity.

BFS was used to calculate the sum of shortest distances, with a time complexity of O(V+E) (V= number of vertices, E = number of edges, denoted as n and m in Graph g). Running the BFS inside a for loop for all vertices gives us a total time complexity of O(V(V+E)) for calculating heist-closeness centrality.

4. Using one of the provided templates, implement your new algorithm and ensure it runs correctly on the sample graphs provided.

File attached separately.

5. Describe your implementation in writing and perform an evaluation that includes a plot containing the run-time as the number of edges increase. Describe at a high level how different input graph topologies may impact your run-time.

The implementation uses a queue to keep track of visited/unvisited vertices and then assigns total distances for each vertex using an implementation of a BFS. This routine is run for all the provided vertices and then the heist closeness centrality is set for each vertex in the graph using the set_vdata function.

Time complexity of the BFS will vary from O(n+n) = O(n) to O(n+n^2) = O(n^2) depending on the graph topology. Therefore the overall complexity will vary from O(n^2) to O(n^3) for different graph topologies.