Aurora Weiden
CS325
Homework 5

Problem 1

a. Dijkstra's algorithm would be used to find the fastest route from the fire station to the intersections. If the fire station is at G than the following are the shortest paths:

Vertex	**Distance**	**Path**
A	12	G, E, D, C, A
B	6	G, H, B
C	8	G, E, D, C
D	5	G, E, D
E	2	G, E
F	8	G, F
G	0	G
H	3	G, H

b. Using Dijkatra's algorithm we could find the shortest path from each intersection to all the other intersections. To do this we would have to iterate through all the possible fire stations and find which one has the shortest farthest distance. Because we are using Dijkstra's algorithm with a runtime of Theta(v^2) and have to loop through each vertex the runtime for this algorithm is Theta(v^3).

c. The optimal location to place the fire station is at E because E has the minimum max of 10.

d. To find an optimal solution for two fire station Dijkstra's algorithm should be performed on each pair of possible locations. The distance between each of the vertices in the pair and all other vertices will be found simultaneously and in each case the shortest between the two distances will be kept. The runtime for this algorithm would be O(f^4).

e. The best locations are H and C with the minimum max of 5.

Problem 2

a. This problem would work well with a BFS implementation that skips over edges that are not at least of weight W. The algorithm would traverse the edges from s to t and if an edge is not of weight W it would not be added to the queue as a possible path. Since

only edges of at least weight W will be considered, the path from s to t will only consist of edges greater than or equal to W.

b. The runtime will be the same as other BFS implementations - Theta(V+E) - since the runtime is dependent on traversing the number of vertices and edges.

Problem 3

a. Pseudocode for an altered BFS algorithm that checks if rivalries exist between wrestlers:

```
def bfs(graph, start):
      queue = start
      while queue is not empty:
            vertex = popped queue
            if first vertex:
                  babyface.add(vertex)
                  queue = rivalries of vertex
            elif vertex not visited:
                  for each babyface:
                        check for rivalry
                        if rivalry == false:
                              babyface.add(vertex)
                        else:
                              for each heeler:
                                    check for rivalry
                                    if rivalry == false:
                                          heeler.add(vertex)
                                    else:
                                          no_solution = true
            queue = rivalries of vertex

      return babyface, heeler, no_solution
```

b. Since at worst case scenario is that the code traverses every babyface and subsequently every heeler the runtime of this algorithm is O(n^2).