Arthur Liou
CS325

HW 5

1. (6 points)
Consider an undirected graph G=(V,E) with nonnegative edge weights w(u,v)□0. Suppose that you have computed a minimum spanning tree G, and that you have also computed shortest paths to all vertices from vertex s□V. Now suppose each edge weight is increased by 1: the new weights w'(u,v) = w(u,v) + 1.
(a) Does the minimum spanning tree change? Give an example it changes or prove it cannot change.
No, the minimum spanning tree does not change. Example: lets run Kruskal's algorithm to find the MST. Because there are distinct edge weights (and they remain distinct after being increased by 1), there is exactly one MST. Kruskal's algorithm will consider the edges in the same order, because we increased all of the edge weights by the same amount. Since the graph structure is the same, Kruskal's algorithm will add / have the same edges in the MST. Thus, the MST will not change.

(b) Do the shortest paths change? Give an example where they change or prove they cannot change.
Yes, the shortest paths may change. The length of a path increases by the number of edges on that path so paths with a shorter number of edges may be shortest paths in the modified graph but not the original graph. For example, one path of 1-2-3-4 and 5-6. The former would be the shortest path, but after +1 (2-3-4-5 & 6-7), the 6-7 path is now the shortest path.

2. (4 points) In the bottleneck-path problem, you are given a graph G with edge weights, two vertices s and t and a particular weight W; your goal is to find a path from s to t in which every edge has at least weight W.
(a) Describe an efficient algorithm to solve this problem.
Do a Breadth First Search (BFS) that ignoring any edges of weight less than W.

(b) What is the running time of your algorithm.
O(V+E), where V is the number of vertices and E is the number of edges

3. (5 points)
A region contains a number of towns connected by roads. Each road is labeled by the average number of minutes required for a fire engine to travel to it. Each intersection is labeled with a circle. Suppose that you work for a city that has decided to place a fire station at location G. (While this problem is small, you want to devise a method to solve much larger problems).

(a) What algorithm would you recommend be used to find the fastest route from the fire station to each of the intersections? Demonstrate how it would work on the example above if the fire station is placed at G. Show the resulting routes.
Dijkstra's algorithm – this would find the shortest (fastest) route between G and every other node
G-A: G-E-D-C-A

G-F: G-F
G-C: G-E-D-C
G-D: G-E-D
G-E: G-E
G-H: G-H
G-B: G-H-B

(b) Suppose one "optimal" location (maybe instead of G) must be selected for the fire station such that it minimizes the distance to the farthest intersection. Devise an algorithm to solve this problem given an arbitrary road map. Analyze the time complexity of your algorithm when there are f possible locations for the fire station (which must be at one of the intersections) and r possible roads.

For each f, I would apply (and modify) Dijkstra's algorithm to each other node & find the distance of the node's farthest intersection. After iterating through each f, I would compare / find the shortest distance of all the farthest intersections. (O(n))
Dijkstra's algorithm worst-case time complexity: O(|R| + |F|log|F|).
Then add in that we do this for each F, so N* Dijkstra's algorithm = O(N (|R| + |F|log|F|))
Time Complexity: O(N (|R| + |F|log|F|))

(c) In the above graph, what is the optimal "location" to place the fire station.
Location E (Longest "shortest" route is 10)

4. (15 points)
Suppose there are two types of professional wrestlers: "Babyfaces" ("good guys") and "Heels" ("bad guys"). Between any pair of professional wrestlers, there may or may not be a rivalry. Suppose we have n wrestlers and we have a list of r pairs of rivalries.

(a) Give pseudocode for an efficient algorithm that determines whether it is possible to designate some of the wrestlers as Babyfaces and the remainder as Heels such that each rivalry is between a Babyface and a Heel. If it is possible to perform such a designation, your algorithm should produce it.

- Create a list to represent a graph G where each vertex represents a wrestler and each edge represents a rivalry. The graph will contain n vertices and r edges.
- Perform as many BFS's as needed to visit all vertices.
- Assign all wrestlers whose distance is even to be babyfaces and all wrestlers whose distance is odd to be heels.
- Then check each edge to verify that it goes between a babyface and a heel.
- If an edge cannot be verified, utilize an invalid flag, flip to true if not valid.
- If valid, print both arrays/lists. If not, then "Not Possible"

(b) What is the running time of your algorithm?
Since it is BFS, it is **O(n + r).** That includes O(n) to designate each wrestler & O(r) to check the edges

(c) Implement: Babyfaces vs Heels in C, C++ or Python. Name your program wrestler and include instructions on how to compile and execute your code in the README file.