CS 325 – HW 5 - Solutions

1. *(6 points)* Consider an undirected graph G=(V,E) with nonnegative edge weights $w(u,v) \geq 0$. Suppose that you have computed a minimum spanning tree G, and that you have also computed shortest paths to all vertices from vertex $s \in V$. 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 it does not change

Proof: Let T = {e_1,e_2,..e_{n-1}} be the minimum spanning tree with total weight w(T) for the graph G(V,E) with weights w(e). Let the graph G'(V,E) be the graph G(V,E) but with weights w'=w(e)+1 for all e in E. I will show that a MST tree T for G is also a minimum spanning tree T' for G'.

Let T' = T = {e_1,e_2,..e_{n-1}} then the weight of T'

$$w(T') = w'(e_1) + w'(e_2) + \dots + w'(e_{n-1}) = (w(e_1) +1) + (w(e_2) +1) + \dots + (w(e_{n-1})+1)$$

$$= w(e_1) +1+ w(e_2) + \dots + w(e_{n-1}) +(n-1)$$

$$w(T') = w(T) + (n-1)$$

Since there are (n-1) edges in the MST and 1 has been added to each edge the weight of the MST in T' must be (n-1) more than the weight of the MST in T for if w(T') < w(T) +(n-1) this would imply that w(T) was not the MST of T. Therefore any MST in G is a MST in G'.

OR

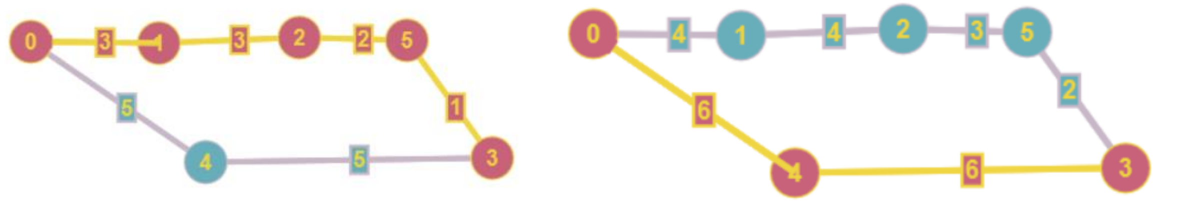
Proof: That the MST does not change when each edge weight is increased by 1.

Suppose that the minimum spanning tree of the original graph was T. After each edge weight is increased by 1, the minimum spanning tree changes to T'. If T' and T are different there will be at least one edge (u, v)∈ T but (u, v) ∉T'. Suppose we add edge (u, v) to the tree T'. Let T*=T'+ (u, v). Since (u, v) was not in T', (u, v) must be the longest edge in the cycle C formed in T*. But since (u, v) is the longest edge it cannot be in the MST T'. Since (u, v) is the longest edge when we decrease each edge weight by 1, (u, v) will still be the longest edge in cycle C formed in T. But the longest edge(u, v) cannot be contained in MST T. Therefore (u, v)∉T which is a contradiction. It implies that trees T and T' are the same.

(b) Do the shortest paths change? Give an example where they change or prove they cannot change.

YES. Give counterexample. See graph below SSP changes

CS 325 – HW 5 - Solutions



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.

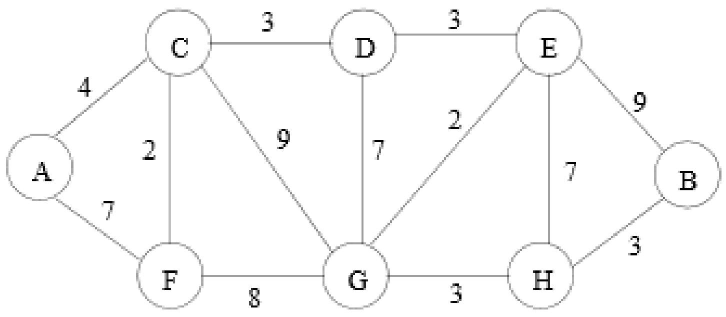
(a) Describe an efficient algorithm to solve this problem.

Perform a BFS of G starting at s and ignoring all edges with weight < W until edge t is reached. If t is not reached before the BFS terminates then there is no path from s to t with weight at least W.

(b) What is the running time of youor algorithm.

O(E+V)

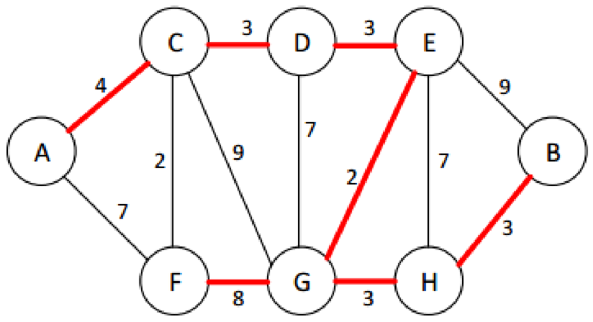
3. *(4 points)* A region contains a number of towns connected by roads. Each road is labeled by the average number of minutes required for a fire engine to travel to it. Each intersection is labeled with a circle. Suppose that you work for a city that has decided to place a fire station at location G. (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.

CS 325 – HW 5 - Solutions

If the fire station is placed at G, using Dijkstra's algorithm, we find the following shortest paths to each intersection (the red thick edges):



Shortest Paths and Distances from G

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) 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. (Again several possible answers).

Algorithm Description:

To determine the optimal location for the fire station such that it minimizes the distance to the farthest intersection, one possible method is, assign EACH of the vertices/intersections of the graph as the source and calculate the shortest paths to every other intersection using Dijkstra's algorithm. Let L(f) be the farthest intersection from any source vertex/intersection, i.e. the longest of the shortest paths from that intersection. To determine the optimal location for the fire station, iterate through each of the candidate fire station location intersections and determine the overall minimum L(f). This is the optimal location for the fire station.

CS 325 – HW 5 - Solutions

Pseudocode:

```
Firestation(G)
  minimum = ∞ // minimum of the farthest intersections
  location = NULL // optimal location of the fire station
  For f ∈ V(G) ← $\Theta(V)$                                                                                  } $\Theta(V^3)$
    distances = DijkstraSSSP(G, f) // distances to each other vertex from source f      } $\Theta(V^2)$
    L = MAX(distances) // maximum of the shortest paths, furthest intersection distance  } $\Theta(V)$
    if L < minimum // if current farthest inters'n closer than prior min. farthest inters'n...  }
      location = f // update optimal location                                             } $\Theta(1)$
      minimum = L // update minimum farthest intersection                                 }
  return location
```

Running Time:
This algorithm is based around repeated use of Dijkstra's single-source shortest path algorithm. Dijkstra's algorithm has an asymptotic running time of $\Theta(V^2)$.

```
Dijkstra(G, w, s)
  InitializeSingleSource(G, s)   }
  S ← ∅                          } $\Theta(V)$
  Q ← V[G]                       }
  while Q ≠ 0 do ← $\Theta(V)$          }
    u ← ExtractMin(Q)                   }
    S ← S ∪ {u}                         } $\Theta(V^2)$
    for v ∈ Adj[u] do  }                }
      Relax(u,v,w)     } $\Theta(V)$    }
```

The algorithm defined above executes Dijkstra's algorithm $\Theta(V)$ times in a loop, for each vertex, to find the shortest paths to all other vertices from that vertex. As such, the overall asymptotic running time of the algorithm to find the optimum location of the fire station is $\Theta(V^3)$. **In the nomenclature defined by the question, if there are *f* possible locations for the firestation and *r* roads, this algorithm runs in $\Theta(f^3)$ time.**

(c) Location E is optimal for the fire station since it's longest shortest path is 10 from E to A. All other locations have at least one longer shortest path.

In the table below we see the lengths of the shortest distances from each intersection to another intersection. E has the minimum max of 10. Table not required.

	A	B	C	D	E	F	G	H	**max**
A	0	18	4	7	10	6	12	15	**18**
B	18	0	14	11	8	14	6	3	**18**
C	4	14	0	3	6	2	8	11	**14**
D	7	11	3	0	3	5	5	8	**11**
E	10	8	6	3	0	8	2	5	**10**
F	6	14	2	5	8	0	8	11	**14**
G	12	6	8	5	2	8	0	3	**12**
H	15	3	11	8	5	11	3	0	**15**

CS 325 – HW 5 - Solutions

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.

Several possibilities modifying BFS or DFS

Create 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.

(b) What is the running time of your algorithm?

Depends on implementation. If using the algorithm described above

This solution would take O(n+r) time for the BFS, O(n) time to designate each wrestler as a babyface or heel, and O(r) time to check edges, which is O(n+r) time overall.

(c) **Implemen**t: Babyfaces vs Heels in C++.

Input: Input is read in from a file specified in the command line at run time. The file contains the number of wrestlers, n, followed by their names, the number of rivalries r and rivalries listed in pairs. *Note: The file only contains one list of rivalries*

Output: Results are outputted to the terminal.

- Yes, if possible followed by a list of the Babyface wrestlers and a list of the Heels .
- No, if impossible.

Sample Input file:

```
5
Ace
Duke
Jax
Biggs
Stone
6
Ace Duke
Ace Biggs
Jax Duke
Stone Biggs
Stone Duke
Biggs Jax
```

Sample Output:

```
Yes
Babyfaces: Ace Jax Stone
Heels: Biggs Duke
```

Submit a copy of your files including a README file that explains how to compile and run your code in a ZIP file to TEACH.