ECE358 PS4

**Q1.**

**a)**

G and H share the same set of minimum spanning trees. First it must be proved that any minimum spanning tree for G is also a minimum spanning tree for H and then it must be proved that any minimum spanning tree for H is also a minimum spanning tree for G.

Let be the summation of all weights of a tree in the graph G, and be the summation of all weights of a tree in the graph H.

Let be a minimum spanning tree for G

**Proof:**

Recall, by properties of a MST, an MST for G or H will have V-1 edges. Because each edge in H has a weight that is 1 greater than the weight in G,

**ATAC:** Assume there is a minimum spanning tree A for the graph H that is not a minimum spanning tree for G. By definition, and .

The terms cancel:

But by definition:

Contradiction! If A is not a minimum spanning tree of G, it also cannot be a minimum spanning tree for H.

**ATAC**: Assume there is a minimum spanning tree B for the graph G that is not a minimum spanning tree for H. By definition, and .

The terms cancel:

But by definition:

Contradicition! If B is not a minimum spanning tree of H, it also cannot be a minimum spanning tree for G.

This means that there cannot be any graph that is a minimum spanning tree for H but not G or a minimum spanning tree for G but not H.

Therefore, G and H share the same set of minimum spanning trees.

**b)**

G and H do not always share the same shortest path between all pairs of vertices.

Consider the following graph G

A diagram of a diagram

Description automatically generated

The shortest path from A → F is: A → B → C → D → F, having a total weight of 4.

This same graph in H becomes:

A diagram of a diagram

Description automatically generated

Now the path A → B → C → D → F has weight 8, while the path A → E → F has weight 7. This means that the shortest path A → F is A → E → F. This counter example proves that G and H do not always share the same shortest path between all pairs of vertices.

**Q4.**

Consider the following graph (all numbers on edges represent the capacity of the edge, not the flow):

A diagram of a network

Description automatically generated

Mi represents doctor i, and Hj represents vacation/holiday j. The flow of this graph is the vacation/holiday days that can be worked. Each doctor can only work at most c vacation days, so each edge connecting S to a doctor node Mi has capacity c. Each doctor can only work 1 day per vacation period, so each edge from a doctor node Mi to a vacation node Hj has capacity 1. Note that the amount of connections between a doctor node Mi and the set of vacation nodes H, is related to the doctor’s availability. A doctor node will only flow into vacation nodes if the doctor has availability on that vacation. Each edge connecting vacation node Hj to T has capacity Dj where Dj is the number of days in that vacation period. The flow from Hj to T represents the number of days in that vacation period that can be covered by a doctor. If the flow into T is equal to N, it means that all vacation days were able to be covered.

**Pseudocode Algorithm (Ford-Fulkerson):**

#Setting up graph, this code will create the graph pictured on the previous page

create graph G=(V,E)

add source node S to G

add sink node T to G

add nodes Mi to G for all i ∈ n

add nodes Hj to G for all j ∈ k

add edge (S, Mi) with capacity c to G for all i ∈ n

add edge (Hj, T) with capacity Dj to G for all j ∈ k

for every node Mi do

for every vacation period Hj that doctor Mi is available for do

add edge (Mi, Hj) with capacity 1 to G

#Solving part of algorithm, this code will maximize the flow given the pictured graph:

for each edge (u,v) ∈ E do

(u, v).f = 0

while there exists an s, t path p in Gf do

cf(p) = min{ cf(u,v) : (u,v) ∈ p do

for each edge (u,v) ∈ p do

if (u,v) ∈ E then

(u,v).f += cf(p)

else

(v,u).f -= cf(p)

#checking if solution is valid:

maxFlow = sum((Hj, T).f) for all j ∈ k

if maxflow ≥ N

return “yes”

else

return “no”

**Runtime Analysis:**

It was noted in class that the run time of this algorithm is:

The size of V and E can be calculated as:

There is one vertex for each doctor, one for each vacation period, a sink and a source:

There is an edge connecting each doctor to the source, one connecting each vacation period to the sink, and then up to k edges for each doctor to connect a doctor to all vacation periods they are available for, and since there are n doctors:

Putting this all together, the runtime is: