1: Unique Minimum Spanning Trees

(a) Let G be a weighted, undirected graph with all edge weights being distinct integers. Prove that there is a unique minimum spanning tree.

Proof by contradiction: Assume there are two distinct MSTs T_1 and T_2. Let e_1 be the edge with the smallest weight that appears in T_1 but not T_2 or in T_2 but not T_1 (assume $e_1 \in T_1$ for the purpose of the proof).

Add e_1 to T_2. By the property of trees, if any edge is added to the tree a simple cycle C is formed.

We know C must contain at least one edge that is not in T_1 as T_1 contains e_1 and if it contained all the other edges in C, T_1 itself would have a cycle (which it can't it's a MST). Let this edge be e_2.

Using the fact $e_1 < e_2$ (e_1 is the smallest weighted edge that appears is only one of T_1 or T_2) if we remove e_2 we will create a spanning tree with a smaller weight than T_2. This is a contradiction as we stated T_2 is a MST.

Proving there is a unique minimum spanning tree if all edge weights are distinct.

(b)

Algorithm: Recall that Kruskal's algorithm starts with each vertex in G being a disjoint tree and all of the edges are sorted by weight. The lightest edge is removed and if it joins two disjoint trees into one tree it is added, otherwise it's discarded.

Run Kruskal's algorithm on G to find a minimum spanning tree. As each edge is added to the MST, mark this edge as visited in the original graph G.

Run Kruskal's algorithm on G a second time with a slight modification. Each time the lightest valid edge is found, check to see if there are edges with the same weight that have not been marked as visited. One at a time, attempt to add the unvisited edge(s) to the MST. If they don't form a cycle and can be added to the MST, we have discovered there is not a unique MST. Otherwise discard them as they don't belong in the MST and repeat this process for the next lightest edge.

If Kruskal's algorithm finishes and all the unvisited duplicate weight edges are invalid for the MST, we know there is a unique MST.

Correctness: We know that Kruskal's algorithm is guaranteed to find a MST. It accomplishes this by making greedy choices considering the lightest edge not already in the MST.

In the our algorithm, we know the first pass of Kruskal's will produce a valid MST. In the second pass of Kruskal's, if there are multiple lightest edges, we are free to choose which lightest edge to consider first and still produce an MST. Thus if it is valid to include an edge in the second pass, that weren't used in the first pass, there exists more than one MST.

If there are no valid edges to include on the second pass, that weren't used in the first pass, there is a unique MST.

Running time: Kruskal's algorithm runs in $O(E \log V)$ time on the first pass. The modified Krusakal's algorithm for the second pass requires us to look forward for duplicate weighted edges. But while checking these edges, if they can't be added to the MST, they are discarded. Therefore,

we still only touch each edge once. Thus the runtime is two passes of Kruskal's algorithm or $O(E \log V)$.

$\boxed{O(E \log V)}$

2: Max Flow Basics

(a)

(b)

(c)

3: More Reductions to Flow

(a) Constructing the graph: starting with a source s, create an edge with capacity $\frac{m}{3}$ to each of three vertices representing the ranks *assistant*, *associate*, and *full* professors.

Create m vertices to represent each of the faculty members. Create an edge of capacity 1 from their rank to their vertex.

Create n vertices to represent each of the departments. Create an edge of capacity 1 from each faculty member to **each** department they work for.

Finally, create a sink t and add an edge of capacity 1 from each department to the sink.

Determine if a committee exists: Run the Ford-Fulkerson algorithm to find the max flow of the constructed graph. If the max flow is equal to n (the number of departments) we can conclude a committee is possible. Otherwise we can conclude it is impossible to find such a committee. Note that a committee with equal representation from all ranks of facility isn't possible if the number departments isn't a multiple of 3.

Find the committee: finding the committee if it exists is simple. Consider each flow from a rank vertex to a faculty member vertex. If the flow is equal to one, that faculty member belongs on the committee.

Correctness: We know that the Ford-Fulkerson algorithm will find the maximum flow. We limit the flow from s by $\frac{m}{3}$ for each of the ranks, upholding the restriction of equal representation for the committee. Additionally, we will have at most n flow as there is n possible flow into the sink t. Finally, each faculty member can only represent one department as they only have at most 1 flow flowing through their vertex.

Running time: Constructing the graph with be linear with respect to the number of departments and faculty with time complexity $O(n + m)$. Ford-Fulkerson is bounded by $O(E * M)$ where E is the number of edges and M is the maximum flow. The maximum flow will be n and size of E will be of the order $O(n + m)$. Finally, we will traverse the flow into faculty Thus the total runtime will be $O((n + m)n)$.

(b) Constructing the graph: starting with a source s, create an edge with capacity 1 to each vertex representing a black square that exists on the checkerboard. Store the location of the

square in the vertex.

Create a vertex representing each white square that exists and create an edge with capacity 1 from each white vertex to a sink t. Store the location of the square in the vertex.

Finally, for each adjacent neighbor a black square has (up, down, left, and right of the square - the neighbors will always be white squares) add an edge of capacity 1 from that black square to each of it's neighbors.

Determine if board can be tiled: Run the Ford-Fulkerson algorithm to find the max flow of the constructed graph. If the max flow is equal to the number of squares on the checkerboard, there exists a way to tile the board. Otherwise, it's impossible.

Find the tiling: Each augmenting path from $s - t$ that compose the max flow will pass through a white square and a black square. This pair represents a domino that will be placed. Look at the location of the two squares (stored in the vertex) and place a domino in that location.

Correctness:

Running time:

(c) Constructing the graph: create two vertices for each vertex in G, calling one set *from* and the other *to*.

Create a source s and add an edge with capacity 1 from the source to each vertex in the *from* set. Create a sink t and add an edge with capacity 1 from the sink to each vertex in the *to* set.

For each directed edge(u, v) in G, add an edge to the flow graph with capacity 1 between $from_u$ and to_v.

Determine if a cycle cover exists: Run the Ford-Fulkerson algorithm to find the max flow of the constructed graph. If the max flow is equal to $|V|$, a cycle cover exists. Otherwise, it's impossible.

Correctness:

Running time:

(d) Proof: Consider the graph G which is a perfect matching bipartite graph. That is, for ever

4: Unexpected Reductions to Flow/Matchings

(a)

(b)