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

step 1: 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.

step 2: Create m vertices to represent each of the faculty members. Create an edge of capacity 1 from their rank to the vertex that represents them.

step 3: 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.

step 4: Finally, create a sink t and add an edge of capacity 1 from each department to the sink.

Determining 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.

Finding 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 $O(n + m)$. The edges of the flow graph have an integer capacity so 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

$O(n+m)$. Thus the total runtime will be $O((n+m)n)$.

$\boxed{O((n+m)n)}$

(b) Constructing the graph

step 1: Create a vertex representing each black square that exists on the checkerboard and create an edge with capacity 1 from a source s to each black vertex.

step 2: Create a vertex representing each white square that exists on the checkerboard and create an edge with capacity 1 from each white vertex to a sink t.

step 3: 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 the black vertex that represents that square to it's white vertex neighbor.

Determining if the 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 half the number of squares on the checkerboard, the board can be tiled. Any max flow that is odd or less than half the number of squares indicates tiling the board is impossible.

Correctness: We know each domino must occupy exactly two adjacent squares of the checkerboard (a pair of adjacent squares will always be one white and one black square). Additionally, each square may only be occupied once.

In the constructed flow graph, an augmenting path from $s-t$ represents placing a domino on two squares and contributes 1 to the total flow from $s-t$. These squares may not be used in any other augmenting paths as the edges have capacity 1. If the checkerboard has s squares, we know we have a solution when exactly $s/2$ dominos have been placed. This is identical to having a max-flow of $n/2$.

Running time: Constructing the graph will be linear with respect to the number of squares on the checkerboard with time complexity $O(s)$ where s is the number of squares. The edges of the flow graph have integer capacities so 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 $s/2$ and size of E will be of the order $O(s)$. Thus the total runtime will be $O(s^2)$.

$\boxed{O(s^2)}$

(c) Constructing the graph

step 1: For each vertex $v_i \in V$, create two vertices in the flow graph: $from_i$ and to_i.

step 2: Create a source s and add an edge with capacity 1 from s to each vertex $from_i$.

step 3: For each directed edge(u, v) in G, add an edge to the flow graph with capacity 1 between $from_u$ and to_v.

step 4: Create a sink t and add an edge with capacity 1 from each vertex $from_i$ to t.

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.

Find the cycle cover

If the max flow of the constructed flow graph is equal to $|V|$, a cycle cover exists. Each augmenting path from $s-t$ in the residual network of the flow graph pass through two vertices that represent an edge in the original graph G. For example, vertices $from_1$ and to_2 represents a directed edge from vertex 1 to vertex 2.

The collection of augmenting paths from $s-t$ in the residual network that constitute the flow represent the edges that form a *cycle cover* of G. This collection of edges forms a set of vertex-disjoint cycles in the graph G that covers all the vertices.

Correctness: A *cycle cover* of a directed graph G is a set of vertex-disjoint cycles in the graph that covers all the vertices. Each vertex may only be visited once and all vertices must be visited.

In our constructed flow graph, we applied the constraint each vertex can only contribute a flow of 1 to the total from from $s-t$ by giving the edges out of s and the edges into t a capacity of 1. Therefore, the only way to achieve a max flow of $|V|$ is if every vertex is visited.

Running time: Constructing the graph will be linear with respect to the number of vertices in G with time complexity $O(m+n)$ where m is the number of vertices in G and n is the number of edges in G. The edges of the flow graph have integer capacities so 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 m and the size of E will be of order $O(m+n)$. Thus the total runtime will be $O((m+n)m)$.

$\boxed{O((m+n)m)}$

(d) Proof

Consider the graph G which is a perfect matching bipartite graph. That is, there exists a matching M where M is a set of edges from G s.t. no two edges share a common vertex. M is a perfect matching meaning it matches every vertex in G such that every vertex in G is incident to exactly one edge in M.

Bob's winning strategy: In this game Alice goes first and names an actress. For Bob to guarantee victory, Bob must answer with the actor matched with Alice's actress in the perfect matching. That is a $m \in M$ is an edge that connects an actress with an actor. It is a perfect matching so we know that for every actress Alice can name, there exists a matching actor that hasn't been named yet.

Bob wins this game as he goes second. In the worst case, Bob can simply respond with the matching actor from M until Alice and Bob have named every eligible actress and actor. At which point it's Alice's turn and she has no actress to name and loses the game.

(d bonus) Proof Now let's consider a graph G in which there is not a perfect matching. In this scenario, Alice has a winning strategy that will never fail.

Alice's winning strategy: There are three cases Alice must consider to win the game.

case 1: It's possible Alice has in her list of actresses an actress that has not co-appeared with any of the actors in Bob's list. In this case, all Alice has to do is name that actress and she wins the game as Bob has no valid responses.

case 2: In Alice's list of actresses there exists a subset of this list that has co-appeared

---

4: Unexpected Reductions to Flow/Matchings

(a)

(b)