1: Unique Minimum Spanning Trees

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

Proof by contradiction
Let us assume that there are 2 Minimum Spanning Trees = Tree1 and Tree2.
Let Tree1 has the following edges: {edge11,edge12,edge13,... ,edge1n}.
Let Tree2 has the following edges: {edge21,edge22,edge23,... ,edge2n}.
If both the tree, Tree1 and Tree2 are not same then the edge set of Tree1 and Tree2 should be different.
Lets consider the smallest cost edge as 'esmall', having the smallest cost. 'esmall' is only present in Tree1 and not in Tree2. As by the definition of tree, we can tell that deleting that edge 'esmall' from Tree1 will create 2 components out of Tree1. As, Tree1 has that edge, 'esmall' must be having the least cost crossing edge, for those 2 particular components. But, as we know, the smallest cost edge 'esmall' must be present in every MST, as defined by the cut property. Therefore, 'esmall' must be present in Tree2 also. But this contradicts our assumption that 'esmall' is only present in Tree1 and not in Tree2. Thus both MST are same and only one Unique MST exists.

(b) General description of the stated problem:

Lets assume G to be the Initial Graph, such that $G = (V, E)$
If we assume that there exist two different Minimum Spanning Tree such that Tree1=(V,E1) and Tree2=(V,E2). Lets assume that Tree1 and Tree2 are different, the set formed by E2-E1 and similarly set formed by E1-E2 must not be empty sets. so there exist an edge 'e1' that belongs to E1-E2.
In this case as we see that an edge 'e1' doesnt belong to E2. So if we add 'e1' to Tree2, a cycle will be formed.
Following the conditions of cycle property, the max expensive edge of this cycle (lets assume 'e11') will not be present in any Minimum Spanning Tree.
So now, if e11=e1, then e11 is present in E1.
or else if e11 is not equal to e1, then e11 is present in E2.
But as we see these above two statements contradicts the fact that e11 is not in any MST.
Therefore that particular case is not possible.

Algorithm: According to Kruskal Algorithm, we can find the minimum spanning tree, but however we may get more than one spanning tree.
To find a unique spanning tree, we may use the below steps:
1. Use Kruskal algo to find minimum spanning tree of a particular Graph1.
2. Lets name the spanning tree as Span1.
3. For every edge that is same in Graph1 and Span1, consider incrementing the weight by 1 of those edge. Other edges that were not present in Span1, but did exist in Graph1 are left untouched.
4. The previous modified graph with new weights is named Graph11. Now run Kruskal on Graph11. In case the new spanning tree Span11 is same as Span1, We can conclude that the provided graph has a Unique MST.

Running time: The time comlexity of this algorithm is that of Kruskal. Verifing if edges of Span1 are in Graph1, will take $O(E)$ where E is no. of edges present in Span1. Therefore, total time complexity is $O(V \log E)$.

Correctness: As we have updated the weights, we can now choose the path that has lesser weight. The particular minimum spanning tree that we will get now will have the minimum no. of edges in common with the initial tree. Therefore, if more than one minimum spanning tree is possible, the above given algo will definitely get it. Otherwise exactly one MST will be found.

2: Max Flow Basics

(a) To Prove: If $\overrightarrow{F1}$ is integral, and there exists a max flow, then there must exist a max flow that is integral.

Proof: We can prove this by setting a practical flow, that represents an integral flow of largest value. We know by theorem that any practical flow $\overrightarrow{F}$ is maximum, if and only if there exist no $\overrightarrow{F}$ augmenting path. Lets assume that there exist an $\overrightarrow{F}$ augmenting path, the contradictory statement may be formalised as, $\overrightarrow{F}$ is not the integral flow of maximum value, as we can say that new flow can be integral when $\overrightarrow{F}$ and $\overrightarrow{F1}$ are integral. Therefore there exist no $\overrightarrow{F}$ augmenting path, and thus $\overrightarrow{F}$ must be the max flow. Hence it is proved.

Reference: Combinatorial Optimization Lecture Notes by Junhui Jia.

(b)Algorithm: Algorithm for required cases is as described:

First let us consider the case when edge capacity increases by one unit:
Let us assume that an edge from 'a' to 'b' has increased it's capacity by 1.
According to the last value of max flow Φ, we just need to do a new iteration of the Ford - Fulkerson algo, using the new changed version of graph.
We update the residual graph, as the edge capacity (a,b) is increased by one in the previously used residual graph.
We will do a graph search in time complexity $O(|V| + |E|)$ to check if any path is present in the residual graph, such that along the path, the flow can be incremented.
If such path exist, a flow of size 1 must be present (as all flow are having integer values).
In case if there exist no flow in the residual flow graph, Φ will still be the max flow.

Running time: The running time of above steps is $O(|V| + |E|)$, as we do a graph search to check if any path is present in the residual graph, such that along the path, the flow can be incremented.

Correctness: As we do a graph search and check if any path is present that can increase the flow value. So the new incremented weight value will definitely be added in the flow, in case that edge is a part of the flow. So correctness is 100%

Second, let us consider the case when edge capacity decreases by one unit:
In case if the last max flow Φ hasn't utilised the full capacity of (a,b), any change won't make any

affect.
Else we will be using the following steps to update the flow:
As the flow, going inside 'a' is one point more than the flow going out of it, and the flow going inside 'b' is one point less than the flow going out of it, there exist no way we must transfer a unit flow from 'a' to 'b'.
Therefore, we will search in the residual flow graph to find a path which goes from 'a' to 'b', such that along with that particular path, the flow will increase by one point. This can be done by visiting the graph in time complexity $O(|V| + |E|)$. In case such a path exists, we will update the Φ with the flow value.
In case no such path exists, the flow from start to 'a' and 'b' to end point must be reduced by one point.
We will accomplish this by getting a path from 'b' to start node inside the residual flow graph, along which we can increment the flow by one point, and by getting a path from end node to 'b' along which we can increment the flow by one point. Such path must be present as a flow from start to end node was existing passing through (a,b).
Now we can update Φ and include these two flows.

Running time: The running time of above steps is $O(|V| + |E|)$, as we search the residual graph to find a path which 'a' to 'b', such that along with that particular path the flow can increase by one point.

Correctness: As we are getting a path from 'b' to start node inside the residual flow graph, along which we can increment the flow by one point, and by getting a path from end node to 'b' along which we can increment the flow by one point. Such path must be present as a flow from start to end node was existing passing through (a,b).
After that we are updating Φ and include these two flows. Thus correctness is 100%.

(c) The given statement can be proved using contradiction.
Lets assume that less than k path exists from nodes t to s.
Remove all the existing path from t to s, thus same number of path from s to t will also be removed.
Hence, now we have a graph in which all properties (indegree and outdegree is same for every vertex) are present.
So, we have some path from s to t, but not from t to s.
Now, lets consider a set of all vertices reachable from t.
s is not present in this set (the set containing all vertices reachable from t) .
Now, there exists no paths from this set to any unreachable nodes but there are some paths from unreachable nodes to this set
We can compare sum of indegree and outdegree of this set.
This gives a contradiction after using the invariant (in degree of every vertex is equal to it's outdegree)
Similar case will arise if we see the situation from other perspective
Hence, proved by using contradiction thatfor any two vertices s and t, and integer k such that k is greater than equal to 1, there exist k edge disjoint path from s to t iff there exist k edge disjoint path from t to s.
Hence proved

3: More Reductions to Flow

(a) We can imagine a flow network for the above requirements, constrains are put up as given in question.
a. Consider layer for faculties, with nodes = m
b. Consider layer for departments with nodes = n
c. One edge is present from each faculty to department
d. Committee has need of equivalent representatives = ranks of professors (assistant, associate, full) thus same flow goes from source to nodes of ranks.
We can say each node has input as n/3.

Running time: Max flow for this network must be = n, so as to form a committee as max n members are possible for the committee. If flow exceeds n, committee formation is not possible. We can verify this in polynomial time using Ford Fulkerson algo.

Correctness: As now the problem is reduced to max-flow situation, we can prove the correctness using:
a. The formation of committee is dependent on value of max-flow. Equivalent flow n/3 goes from source to rank node. The weight=1 for every edge connecting rank node to faculty, summation of which can be max =n. Similar connection from faculty to department, can be made, every faculty can have connection to more than one dept. that implies, summation of faculty-dept edge can be greater than n. As we see, only n dept are present, and every dept. has one edge (weight 1) connecting it to committee, so max flow going inside sink is n. So max no. of representation committee will be receiving is equivalent to n, with same representatives from every rank n/3. So max-flow can tell us if committee will be formed or not.
b. If committee is formed then max flow is equal to n. As committee needs n members, if it is made then max flow computation is accurate, also constrains are successfully applied.
Thus the required statement is proved.

(b) The Tiling problem:
Here we can observe that if number of black and white tiles are not equal then no solution is possible.
To find a possible solution, if it exists, we can use the follwing algorithm:
1. Assume a graph with vertices, equal to unit square for the plan.
2. Here any two vertices will only be connected through edge iff the corresponding squares will be neighbours.
3. The graph made will be bipartite as both partitions are divided by colors.
4. Every titling for the plan, will be a perfect match, as each vertex will be covered using exactly 1 edge of tile.

This can be extended further to find a maximal matching, for a bipartite graph, in polynomial time complexity.
Steps:
1. Start with null match
2. If matching is present, then augmenting path can be present.
3. If augmenting path is not present then matching is maximal beforehand.
4. In case we have a augmenting path present, we can create a bigger match by exchanging edges

along the path.
5. Thus, we can find an augmenting path (if it is present) using bfs by starting from the non matched vertices of the particular partition.
6. At last after getting maximal matching we can further check if it is perfect or not perfect.

(c) Following algorithm can be followed for finding a cycle cover.

Algorithm:
1: We know that finding a max (weighted) matching inside any weighted graph will need a polynomial time complexity (by applying Ford - Fulkerson algorithm = $O(Ef)$), thus we get a perfect matching. Such kind of Perfect matching is max everytime, and we will always be able to find a perfect match if it exists. As given a graph G having V, vertices and E, edges. We can create a bipartite graph G1, such that $G1^{(}V \cup V^{`}, E^{`})$. This will be done using the below steps: For the particular set of vertices, there is a copy $V^{`}$ in $G1$.And similarly for each directed edge $e = (u, v)\epsilon E$ in Original Graph G, an undirected edge will be present $e^{`} = (u, v^{`})\epsilon E^{`}$ in $G1$. also the weight of $e\epsilon E$ will be equal to weight of $e^{`}\epsilon E^{`}$.
2: After creation of bipartite graph in above step we will try to get the max cost perfect matching for bipartite graph as described in 1st point above.
3: Now, cycle cover will be found which is given as $C = (u, v) : u, v^{`}\epsilon M, u\epsilon V, v^{`}\epsilon V^{`}$
4: Output the cycle cover if it is present
5: Else return False

Correctness: The algorithm will give cycle cover if it is present. Hence correctness is 100%

Running time: Time complexity = polynomial time. Making a bipartite graph will take polynomial time. Searching for max matching will also need poly time, hence total complexity will be polynomial time.

(d) Here we need to show that if perfect matching exist in Graph, then there exists a winning strategy for Bob. Thus, to show the above, let Bob fix a perfect matching Z in the Graph. Here Graph = bipartite graph between actresses and actors, in which there is an edge iff the two have co-appeared in a movie, as stated in the question. When it is Bob's turn in the game (please consider that the v=current vertex is fixed), Bob will select a *unique* edge $e\ \epsilon Z$ for which $v\ \epsilon e$.
Claim If perfect matching is present in the Graph, then Bob will play the strategy mentioned above. If he does so then it will result in Bob = winner of the game.
Proof Let us make an assumption that Alice will pick up some vertex v in the beginning. And Graph has a perfect matching thus v must be matching with some edge $e = \{v.v^{'}\}\epsilon Z$. Bob, selects e as mentioned in the plan above and then Alice has a turn. Thus, we have 2 different cases:
1. There will be no edge in between $v^{'}$ and any other unvisited vertex. In this case Bob wins acording to given rules in the question and the game will end.
2. In the second case, Alice will be selecting some edge $e^{'} = \{v^{'}, w\}$ adjacent to the vertex $v^{'}$. but we know $e^{'}$ is not part of Z as $v^{'}$ has a connection matched with e and e is not same as $e^{'}$ as v is already visited. Next, it will be Bob's turn in the game. We currently are at some other vertex w. As we say in the above argument, the edge choosen by Alice is not a part of Z, Bob will again follow the above stated strategy and choose *unique* edge in Z that will contain w. Consider here that the stated edge must exist, as this w is not visited previously and Graph has a perfect matching.
Game will end as soon as player has no option to make any legal moves in the game. And as we argued above, Bob always has an option to make a next move and thus game will lastly end with Case 1 scenario. Thus, Bob will win the game, if perfect matching exists in Graph.

BONUS Now we will see if no perfect matching exists in Graph, then Alice has a winning strategy, in which case Bob will lose the game. In this case Alice will fix some max. matching Z. Alice will start the game by choosing an unmatched vertex under Z. In the game whenever it is Alice's turn at any vertex v, Alice will choose an *unique* edge $e \epsilon Z$ with $v \epsilon e$.
Claim If no perfect matching exist in Graph, then Alice can play the above plan/strategy and win the game.
Proof Alice will follow the Strategy and pick up a vertex v which is not matched under Z . As there exist no perfect matching, such a vertex v much be present. After the intial turn of Alice, it is Bob's turn.
1. First case, there exist no edge between v and any unvisited vertex. Thus , Alice will win in this case and game will end.
2. In the second case, Bob will select some any edge $e = \{v, v^{'}\}$. As v is not matched in Z, it shows that e is not a part of Z.
Now, we can Claim that $v^{'}$ is matched in Z. To extend this further, suppose this was not the case. Then we see that v and $v^{'}$ will both be left unmatched, and thus the new formed set $Z \cup \{e\}$ will be a matching. But, $\|Z \cup \{e\}\|$ is greater than $\|Z\|$, which is a contradiction to *maximality* of Z. As $v^{'}$ is matched in Z but the edge e is not present in Z (and as mentioned in the game $v^{'}$ has not been previously visited), Alice will now be choosing the *unique* edge $e^{'} \epsilon Z$ such that $v^{'} \epsilon e^{'}$. As this strategy can be used after every move that Bob makes (and each vertex will be visited atleast once), Alice will play the strategy.
To ensure that Alice wins, let's assume that contradiction occurs and Bob wins the game. Let us assume that Alice chooses A set of edges and Bob chooses B set of edges. As we have claimed that Bob wins, Bob would have chosen the first and last edge, hence $\|B\| = \|A\| + 1$. If we use the same agrument or strategy as mentioned above for each turn, we can say that all edges present in B are not part of Z but all edges in A are part of Z. Thus, B Intersection Z is NULL. adn A intersection Z is A. As in the above strategy, game starts at unmatched vertex so it must have also ended in unmatched vertex:
1. If previous or last vertex was matched with an unvisited vertex, then Alice could have played another move and game would not have ended that way.
2. Previous or last vertex can alos not be matched to any visited vertex, because as per the argument above, those vertex are all already matched by edges chosen previously in the game. Thus, we can say that, the path created by Alice and Bob, is augmenting path for Z. This implies that $Z^{'} = (Z/A) \cup B$ is a matching. Additionally, $\|Z^{'}\| = \|Z\| + 1$, that contradicts the maximality of Z.
(Reference: Notes of Dr.J Lengler.)

4: Unexpected Reductions to Flow/Matchings

(a) The Goal is to find a subset S of users, such that the number of edges in G both of whose end points lie in S is at least $|\lambda|S$

Lets assume that there exist a cut 's' in given graph G(V,E).
As given, m is the total number of edges in G.
As given, d_i refers to degree of vertex i.
As we can see according to question, the mathematical derivation of given case will be:
So now accordingly the flow is:

$m|V - s| + |E(s, \overline{s}) + | \sum_{i \in s}(2\lambda + m - d_i)$ (Equation 1)
here, $m|V - s|$ = combined weight of all the vertices that are not present in set 's'.
and we can see, $|E(s, \overline{s})|$ is the out going edges.

Accordingly we can observe that using the Equation 1:
$| \sum_{i \in s}(2\lambda + m - d_i)$ may be formulated as $(2\lambda + m)|s| - \sum_{i \in s}(d_i)$ (Equation 2)

Combining from equation 1 and 2:
$\sum_{i \in s}(d_i)$ may be formulated as$|E(s, \overline{s})| + 2|E(s, s)|$ (Equation 3)
here, $|E(s, \overline{s})|$ = the particcular edges going in the cut 's'.

Thus summing the equation, we can conclude:
$2|E(s, s)|$ =the edges present inside the cut 's'.

Hence the combined equation can be writtten as:
$m|V - s| + |E(s, \overline{s})|) + (2\lambda + m)|s| - |E(s, \overline{s})| - 2|E(s, s)|$

Computing and reducing the above equation, we get:
$m|V| + 2\lambda|s| - 2|E(s, s)|$ (Equation 4) = This will be the flow when 's' is a cut.

On using the Ford-Fulkerson algorithm,we will get the same results for flow.

Now, let us consider that, $E(s, s) \geq \lambda|a|$

$\implies 0 \geq 2\lambda|s| - 2E(s, s)$ which is less than '0'

Hence if equation 4, $m|V| \geq m|V| + 2\lambda|s| - 2|E(s, s)|$ which is also the Ford fulkerson output is satisfied, it implies that a cut 's' is present else a cut 's' is not present.

Correctness: Applying Ford fulkerson algo, given that any cut 's' is present, one can find the set 's'.

Running time: Runtime = runtime of Ford Fulkerson algorithm, that is $O(Ef)$, where E is no. of edges and f is max flow in the graph.

(b) The required algorithm can be designed as:
1. Lets assume that a bipartite graph of tiles exist. The total number of vertices in graph will be 2n as n rectangular tiles are present. An edge will exist between any two pair of vertices iff the given statement in question is satisfied which states "stacking a tile on each other". We will assume that the weight of the edge is least or min. area of the 2 vertices that are present.
2. On other case, lets take into account the case when tiles wont stack on one another. For this particular case, we can consider the max area that is equivalent to the sum of the area of the tiles. In this situation if we can place one tile over the other, hence the full area will be decremented by the area of particular tile that is placed on top (tile with low area). Therefore, we have saved the area, that is equal to the area of the small tile
3. Thus, the max area that can be possible is the space that is utilised and space that is saved. The computation that we need to do is for minimizing the space utilised and maximizing the space that is saved. Hence, we initialize the weight of the edge in particular bipartite graph, which is denoted as the min. area of vertices or tiles.
4. The Graph G is now sent as an input, to the given blackbox that will output the max bipartite

matching, as for the given graph. This above computation will be done in polynomial time.
5. Further, we will decrement the balckbox output from the max area, which is equal to the summation of all areas. Here we are taking into account, summation of all area, and no tile is stacked on top of each other.

Correctness: Blackbox shows us the max bipartite matching. It lets us know, the max amount of space saved.
Worst case scenario - No tile stacked on top, this is equal to the summation of independent area of tiles. This will be a constant value.
This worst case will be constant = summation of the space saved and the space utilised. As we max out the output of blackbox (maximize the space saved), automatically space utilised will be least in this case. Hence, we will always get min space utilised. Correctness is 100%

Running time: : Blackbox runtime is polynomial. Other than this time is used for construction of bipartite graph, and in computation of difference b/w worst case and blackbox o/p which will take poly time and constant time. Thus total process takes polynomial time.