Name: ______________________________

Entry number: ______________________________

There are 4 questions for a total of 75 points.

---

1. (20 points) Given a weighted, undirected graph G and a minimum spanning tree T of G. Suppose that we decrease the weight of one of the edges not in T. Design an algorithm for finding the minimum spanning tree in the modified graph. Give pseudocode, discuss running time, and give proof of correctness.

Solution: We will first prove the following claim.

Claim 1.1: Let G be a weighted graph and let T be a MST of G. There exists an ordering of the edges of G such that the weights of these edges are in increasing order and the Kruskal's algorithm, when run on this ordering of edges, outputs T.

Proof. If all the edge weights are distinct, then T is the unique MST of G and since the Kruskals algorithm outputs an MST, it outputs T . Now if all the edge weights are not distinct, then we can argue in the following manner:

We will slightly modify the edge weight of each edge so that all edge weights become distinct. Let $E_T = \{t_1, t_2, ..., t_{n-1}\}$ denote the edges of T such that they are sorted in increasing order of their weights. That is $W(t_1) \leq W(t_2) \leq ... \leq W(t_{n-1})$. Let $E'_T = \{s_1, ..., s_k\}$ denote the remaining edges in the graph. We construct the following new graph that has the same vertices and edges as G but the edge weights are slightly modified. Let W denote the weight function of G and W' be the weight function of G' that is defined in the following manner. Let β be the minimum value of the difference of weights of two edges of G and let $\alpha = \beta/n^3$. For edges in E_T, the weights are given by $W'(t_1) = W(t_1) - (n-1) \cdot \alpha$, $W'(t_2) = W(t_2) - (n-2) \cdot \alpha$, ..., $W'(t_{n-1}) = W(t_{n-1}) - 1 \cdot \alpha$. For edges in E'_T, the weight function is defined as $W'(s_1) = W(s_1) + 1 \cdot \alpha$, $W'(s_2) = W(s_2) + 2 \cdot \alpha$, ..., $W'(s_k) = W(s_k) + k \cdot \alpha$. Note that the way we have defined the weight function W', we have made sure that all the weights of edges are distinct. We now show the following two sub-claims:

Claim 1.1.1 The MST of G' is an MST of G.

Claim 1.1.2 T is the MST of G'.

We will defer the proof of the above sub-claims and first see how these imply Claim 1.1. Let $e_1, ..., e_m$ be an ordering of edges such that $W'(e_1) < W'(e_2) < ... < W'(e_m)$. Note that due to the way we have defined the weight function W', we get that $W(e_1) \leq W(e_2) \leq ... \leq W(e_m)$. This is because for any two edges e_i, e_j such that $W(e_i) < W(e_j)$, we have

$$W'(e_i) \leq W(e_i) + (W(e_j) - E(e_i)) \cdot (n^2/n^3) < W(e_j) - (W(e_j) - E(e_i)) \cdot (n/n^3) \leq W'(e_j).$$

Note that the Kruskal's algorithm, when executed using the ordering $e_1, ..., e_m$, returns T . This proves the Claim 1.1.

Let us now prove sub-claims 1.1.1 and 1.1.2.

Proof of claim 1.1.1. Let V be the total weight of any MST of G. Note that $V = W(t_1) + W(t_2) + ... + W(t_{n-1})$. Consider the weight V' of T in G'. $V' = V - \frac{n(n-1)}{2} \cdot \alpha$. The proof follows from the fact that any other spanning tree will have weight more than V' in G'. ∎

Proof of claim 1.1.2. Consider any edge e in T. When e is removed from T, it gets disconnected into two subsets of vertices, say V_1 and V_2. Note that the weight of e is one of the smallest among the edges going across the cut (V_1, V_2) in G. This implies that e is the smallest weighted edge that goes across the cut (V_1, V_2) in G'. This implies that all MST's of G' should contain e. Similarly we can argue for all edges in T. $\square$

This concludes the proof of Claim 1.1. $\square$

We will use the above claim to prove the correctness of our algorithm that is described as follows:

The algorithm: Given G, T, e, if e is already in T, then the algorithm returns the same T. Otherwise, the algorithm adds this edge to the MST T creating a cycle. The algorithm then goes around the cycle (using DFS) and throws out the edge with maximum weight.

Proof of correctness: In Claim 1.1, we have shown that given any weighted graph G and an MST T of G, there is an ordering of edges (in increasing order of weight) of G such that the Kruskal's algorithm when executed using this ordering returns T . Let $e_1, ..., e_m$ be this an ordering. Suppose that we have decreased the weight of edge e_i. Let us re-insert this edge e_i into this ordering such that the as per the new ordering, the edge weights are again sorted. This ordering can be denoted as $e_1, ..., e_j, e_i, e_{j+1}, ..., e_{i-1}, e_{i+1}, ..., e_m$. We now examine the MST returned by the Kruskal's algorithm on this ordering. If the algorithm does not pick e_i, then the MST returned is exactly T. If the algorithm picks e_i, then the only edge from T that it does not pick is the one that completes a cycle involving the edge e_i. This is precisely the maximum weight edge along the cycle that is created when e_i is added to T.

Running time: The time to go around the cycle involves simply using DFS on the tree T. This will cost $O(n)$ time.

2. (15 points) Let T be a minimum spanning tree of a weighted, undirected graph G. Given a connected subgraph H of G, show that $T \cap H$ is contained in some minimum spanning tree of H.

Solution: For any graph $G = (V, E)$ and its MST T, we have shown in Claim 1.1 that there is an ordering of edges of the graph such that the edges are in increasing order of their weights and when Kruskal's algorithm is run on this ordering, the minimum spanning tree obtained is T. Let $e_1, e_2, ..., e_m$ be this ordering of edges. Let $T = \{e_{i_1}, e_{i_2}, ..., e_{i_{n-1}}\}$ such that $i_1 < i_2 < ... < i_{n-1}$. Let us use H to denote the edges of the subgraph H. Let $H = \{e_{j_1}, e_{j_2}, ..., e_{j_l}\}$ such that $j_1 < j_2 < ... < j_l$. The following claim proves the result.

Claim 2.1: When Kruskal's algorithm is executed on the graph H where the edges are considered in the order $e_{j_1}, ..., e_{j_l}$, all edges in $T \cap H$ is picked by the algorithm and hence in the MST produced by the algorithm.

Proof. Let $T \cap H = \{e_{k_1}, e_{k_2}, ..., e_{k_r}\}$ such that $k_1 < ... < k_r$. For the sake of contradiction assume that the Kruskal's algorithm when executed on $e_{j_1}, ..., e_{j_l}$ does not pick one or more edges in the set $T \cap H$. Let e_{k_s} be the first such edge (as per the sequence $e_{k_1}, ..., e_{k_r}$). Let $k_s = i_p = j_q$ and consider the forest $F = (V, \{e_{i_1}, e_{i_2}, ..., e_{i_{p-1}}\})$. We show the following claim:

Claim 2.1.1: Every edge in $\{e_{j_1}, ..., e_{j_q}\}$ is either (i) a tree-edge in F, or (ii) between two nodes in the same tree in F.

Proof. The proof follows by the fact that if an edge in the set $\{e_{j_1}, ..., e_{j_q}\}$ is not in the set T, then that means that it was not picked by the Kruskal algorithm when executed on the sequence $e_1, ..., e_m$. This means that it formed a cycle which in turn means that this edge is between two vertices in the same tree in F. □

The above claim means that the edge e_{k_s} does not form a cycle with respect to the edges picked by the Kruskal algorithm when executed on $e_{j_1}, ..., e_{j_{q-1}}$. So the edge e_{k_s} will be picked by the algorithm which is a contradiction. □

3. There is a currency system that has coins of value $v_1, v_2, ..., v_k$ for some integer $k > 1$ such that $v_1 = 1$. You have to pay a person V units of money using this currency. Answer the following:

   (a) (16 points) Let $v_2 = c^1, v_3 = c^2, ..., v_k = c^{k-1}$ for some fixed integer constant $c > 1$. Design a greedy algorithm that minimises the total number of coins needed to pay V units of money for any given V. Give pseudocode, discuss running time, and give proof of correctness.

Solution: Here is a greedy algorithm that we will analyze:

`GreedyCoinSelect`(c, V)
- For $i = k$ to 1
  - $n \leftarrow \lfloor \frac{V}{c^{i-1}} \rfloor$
  - $g[i] \leftarrow n$
  - $V \leftarrow V - n \cdot c^{i-1}$
- return$(g[1], g[2], ..., g[k])$

Correctness proof: Let a solution be represented as an n-tuple $(m_1, ..., m_k)$ which denotes that the number of coins of value v_1 is m_1, coins of value v_2 is m_2 and so on. Let $(g_1, ..., g_k)$ be a greedy solution and $(o_1, ..., o_k)$ be any optimal solution. We first show that for all $1 \leq i < k, o_i \leq (c-1)$.

Claim 3.1: For all $1 \leq i < k$, $o_i \leq c-1$.

Proof. For the sake of contradiction, assume that there is an index $1 \leq i < k$ such that $o_i \geq c$. In this case, the solution $(o_1, ..., o_{i-1}, o_i - c, o_{i+1} + 1, ..., o_k)$ is also a valid solution but with smaller number of coins which is a contradiction. □

We are now ready to prove that $(o_1, ..., o_k) = (g_1, ..., g_k)$.

Claim 3.2 $(o_1, ..., o_k) = (g_1, ..., g_k)$.

Proof. For the sake of contradiction, assume that there is an index j such that $g_j \neq o_j$. Let j be the largest such index. That is, $g_{j+1} = o_{j+1}, ..., g_k = o_k$. This means that $o_j < g_j$ (if $o_j > g_j$, then the total value is exceeded). Now, we show a contradiction from the following sequence of equations.

$$
\begin{aligned}
\sum_{i=1}^{k} o_i \cdot c^{i-1} &= \sum_{i=1}^{j-1} o_i \cdot c^{i-1} + \sum_{i=j}^{k} o_i \cdot c^{i-1} \\
&\leq \sum_{i=1}^{j-1} (c-1) \cdot c^{i-1} + \sum_{i=j}^{k} o_i \cdot c^{i-1} \quad \text{(using Claim 3.1)} \\
&= c^{j-1} - 1 + \sum_{i=j}^{k} o_i \cdot c^{i-1} \\
&< (o_j + 1) \cdot c^{j-1} + \sum_{i=j+1}^{k} o_i \cdot c^{i-1} \\
&\leq g_j \cdot c^{j-1} + \sum_{i=j+1}^{k} o_i \cdot c^{i-1} \quad \text{(since } g_j > o_j\text{)} \\
&= g_j \cdot c^{j-1} + \sum_{i=j+1}^{k} g_i \cdot c^{i-1} \quad \text{(since } g_{j+1} = o_{j+1}, ..., g_k = o_k\text{)} \\
&\leq \sum_{i=1}^{k} g_i \cdot c^{i-1}
\end{aligned}
$$

This is a contradiction since the value of both solutions should be the same (equal to V). □

Running time: The number of arithmetic operations involved in the algorithm is $O(k)$.

(b) (4 points) Let $c > 1$ be any fixed integer constant. Does your greedy algorithm above also work when for all $1 \leq i < k$, $\frac{v_{i+1}}{v_i} \geq c$? Give reason for your answer.

Solution: For any integer constant $c > 1$, consider the coins $v_1 = 1, v_2 = 5c, v_3 = 10c^2 - 4c$. We have $v_2/v_1 \geq c$ and $v_3/v_2 \geq c$. Let $V = 10c^2$. The optimal solution uses $2c$ coins. However, the greedy algorithm uses $4c + 1$ coins.

4. (20 points) Given a list of n natural numbers $d_1, d_2, ..., d_n$, design an algorithm that determines whether there exists an undirected graph $G = (V, E)$ whose vertex degrees are precisely $d_1, ..., d_n$. That is, if $V = \{v_1, ..., v_n\}$, then degree of v_i should be exactly d_i. G should not contain multiple edges between the same pair of nodes or "loop" edges. Give pseudocode, discuss running time, and give proof of correctness.

Solution: Let $G(d_1, ..., d_n)$ denote the proposition that there exists an undirected graph where v_1 has degree d_1, v_2 has degree d_2 and so on. Let us assume that $d_1 \geq d_2 \geq ... \geq d_n$. We will prove the following claims.

Claim 4.1: $G(d_1, ..., d_n) \Leftarrow G(d_2 - 1, d_3 - 1, ..., d_{d_1+1} - 1, d_{d_1+2}, ..., d_n)$.

Proof. Let G be the graph such that v_1 has degree $d_2 - 1$, v_2 has degree $d_3 - 1$ and so on. Then we can construct a graph with degrees $d_1, ..., d_n$ by adding a vertex to G and adding edges from this vertex to $v_1, v_2, ..., v_{d_1}$. □

Claim 4.2: $G(d_1, ..., d_n) \Rightarrow G(d_2 - 1, d_3 - 1, ..., d_{d_1+1} - 1, d_{d_1+2}, ..., d_n)$.

Proof. Let G be the graph with vertex v_1 with degree d_1, v_2 with degree d_2 and so on. Suppose v_1 has edges to vertices $v_2, v_3, ..., v_{d_1+1}$ in G. Then removing v_1 and its edges gives the lemma. Suppose this is not the case and let v_j for $2 \leq j \leq d_1 + 1$ be the first vertex to which v_1 does not have an edge in G. We will construct another graph G_1, in which there is an edge from v_1 to vertices $v_2, v_3, ..., v_j$. We construct G_1 from G in the following manner: v_k be a vertex with maximum value of k such that v_1 has an edge to v_k in G. Since the degree of v_j is at most the degree of v_j, there exists a vertex $v_i \neq v_k$ to which v_j has an edge in G but v_k does not. We construct G_1 by removing the edges $(v_1, v_k), (v_j, v_i)$ and adding the edges $(v_1, v_j), (v_i, v_k)$.

Now suppose, v_1 has edges to $v_2, ..., v_{d_1+1}$ in G_1, then again removing v_1 gives the lemma. Otherwise, we repeat the previous argument to create a sequence of graphs with the same degree sequence such that in the final graph in the sequence v_1 has edges to $v_2, ..., v_{d_1+1}$. Now removing v_1 in this graph gives the result. □

The above two claims suggest the following simple recursive algorithm for the problem. The input is n and a degree matrix $D = (D[1], D[2], ..., D[n])$.

`GraphCheckSort`(n, D)

- Sort elements in the matrix D in increasing order
- If $(D[n] = 0)$return(1)
- If $(n - 1 < D[n])$return(0)
- For $i = (n - 1)$ to $(n - D[n])$
  - $D[i] \leftarrow D[i] - 1$
- return(`GraphCheckSort`$(n - 1, (D[1], ..., D[n - 1]))$)

The proof of correctness of the above algorithm follows from Claims 4.1 and 4.2. The recurrence relation for the running time is $T(n) = T(n - 1) + O(n \log n); T(1) = O(1)$ which gives $T(n) = O(n^2 \log n)$. We can improve the running time by taking the sorting out of the recursion. Suppose, in the above program we also maintain an array A and B such that the $A[i]$ stores the index in the array at which i starts and $B[i]$ store the index at which i ends. For example, if $D = (1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5)$, then $A[1] = 1, B[1] = 3, A[2] = 4, B[2] = 5, A[3] = 6, B[3] = 8$ etc. Creating sorted array D and A, B before running the program will cost only $O(n \log n)$ time.

```
GraphCheck(n, D)
  - If (D[n] = 0)return(1)
  - If (n − 1 < D[n])return(0)
  - p ← n − D[n]
  - i ← A[D[p]]; j ← B[D[p]]
  - For i = (n − 1) to p
      - D[i] ← D[i] − 1
  - While (D[i] > D[j])
      - Swap(D[i], D[j])
      - i ← i + 1; j ← j − 1
  - Update arrays A and B
  - return(GraphCheck(n − 1, (D[1], ..., D[n − 1])))
```

Running time: Every time the n^{th} degree is removed the amount of work done is $O(n)$. The recurrence relation for the running time becomes $T(n) = T(n-1) + O(n); T(1) = O(1)$. The solution is $O(n^2)$.