Q1. (10 points)

Given a undirected, weighted graph G=(V,E). Assume that no two edges have the same weight. Prove that there is a unique minimum spanning tree (MST) for such a graph.

I will use a proof by contradiction. First assume that there is an MST constructed from the given graph that is not unique -- there is another MST of equal weight but a different structure.

Let e1 be an edge from vertices U to V that is in the first MST but not the second. Then the second MST must contain a path e2 from U to V which is different from that in the first MST, resulting in a cycle.

Assume that weight of e1 is less than that of e2. If we replace e2 with e1 in the second MST, it has a smaller weight compared to B which is a contradiction.

Q2. (10 points)

Consider the following algorithm for finding a MST. Given a graph G=(V,E), sort the edges in decreasing order of their weights. Pick each edge e in sorted order, and if e creates a cycle in G, remove it from G. Note that each time you remove an edge, you are modifying the graph, so that the next check for a cycle will be on the reduced graph. Prove that this method is correct, and analyze its running time.

First we'll prove that the algorithm doesn't modify the connectivity / spanning property of the graph. We know that the graph we are given initially has a at least one connected region. At each step we only remove cycles which by definition cannot disconnect the graph because there is another edge connecting the nodes involved. By induction, we can infer that the global connectivity of the graph is not changed.

Next I'll prove that the algorithm produces a tree structure. This is easy to prove. The algorithm touches every edge in the graph and removes the edge if it creates a cycle – eliminating all cycles and creating the tree.

Finally, I'll prove that the algorithm produces a minimum weighted spanning tree. We know that 1) the algorithm only deletes cycles / edges that would disconnect the graph and 2) that the algorithm deletes cycles in decreasing order of their weights. That means that the only edges that remain are those with the smallest weights that do not disconnect the graph.

All these properties define a MST, so we know the algorithm is correct.

As for runtime…

1) Sorting the edges takes ElogE time

2) We have a outer loop through the number of edges:

Checking for the presence of a cycle can be done by temporarily marking an edge as deleted then running a DFS to see if the two nodes connected by the edge are still reachable from one another O(V+E)

Thus our runtime comes out to be O(ElogE + E*(V+E)) → O(E(V+E))

Q3. (10 points)

Consider the problem of a constrained MST. Given a weighted, undirected graph G=(V,E), and given a subset of vertices C⊆V, describe a method to find a constrained MST such that vertices in C only appear as leaves in the constrained MST. What is the running time of your algorithm? Note that for some choice of C, it may be the case that the constraint cannot be satisfied, i.e., it may be the case that there is at least one vertex in C that cannot be made a leaf node. In these cases, you can output that the constraint cannot be satisfied.

First construct new sets of vertices and edges by considering all vertices and edges not in / connecting vertices in U.

Run Kruskal's algorithm on this new graph. If a minimum tree cannot be made, then the constraint cannot be satisfied.

Next for every node in C in increasing order of weight, greedily connect those nodes to the MST using a node already in the MST (i.e. not in C). If such an operation cannot be completed, then the constraint cannot be satisfied.

The algorithm uses Kruskal's algorithm as it's basis which subsumes all the other operations when considered under big oh analysis. Thus the runtime for the algorithm is still O(Elog E)

Q4a. (5 points)

Given an alphabet with n characters. Let the characters be numbered from 1 to n, and let the frequency of character i be $\boldsymbol{f_i = \frac{1}{2^i}}$ for i=1,2,...,n−1, with the last character n having frequency $\boldsymbol{f_n = \frac{1}{2^{n-1}}}$. For example, if n=6, then the frequencies of the first five characters are 1/2, 1/4, 1/8, 1/16, 1/32, respectively, and the frequency of the last character is 1/32. Answer the following questions:

Show that the frequencies of all characters sum to 1 (as they should) for any n.

The above series can be represented as:

$$(\sum_{i=1}^{n-1} \frac{1}{2^i}) + \frac{1}{2}^{n-1}$$

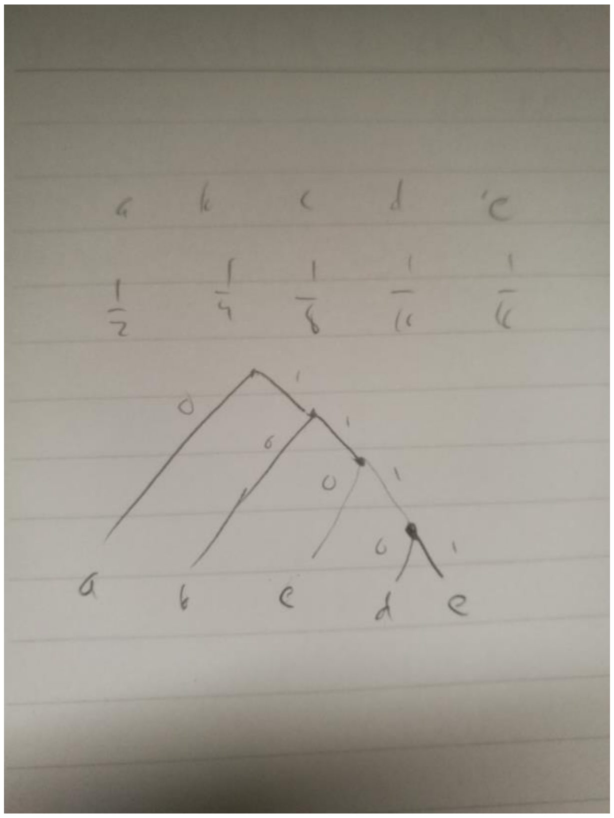
Using the series rules we have

$$\left(\sum_{i=1}^{n-1} \frac{1}{2^i}\right) + \frac{1}{2}^{n-1} = \frac{.5(1 -.5^{n-1}}{.5} + .5^{n-1} = 1 -.5^{n-1} +.5^{n-1} = 1$$

Q4b. (5 points)

Show what the huffman encoding is for each character. In building the encoding tree, the larger frequency child should be on the left, and if there are two groups of characters with the same frequency, the one with the smaller index should be on the left.

Take the set of characters a,b,c,d,e, as an example with the frequencies ½,1/4,1/8,1/16,1/6



The most appearing is always the leftmost and the least appearing is always rightmost. Thus by generalizing, we can do the code for any character as:

1*(i-1) + 0 where i is the number assigned to the character that determines it’s frequency.

Q4c. (5 points)

What is the expected number of bits per character? Let the encoding length of character i be denoted as li, then the expected or average number of bits for the encoding is computed as $\sum_{i=1}^{n} l_i * f_i$. Use this formula to derive an closed form expression (for any n).

$$\sum_{i=1}^{n} \left(1 * (i-1)\right) * \frac{1}{2^i} = expected\ number\ of\ bits$$