Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 MST Basics

For each of the following statements, either prove or give a counterexample. Always assume $G = (V, E)$ is undirected and connected. Do not assume the edge weights are distinct unless specifically stated.

(a) Let e be any edge of minimum weight in G. Then e must be part of some MST.

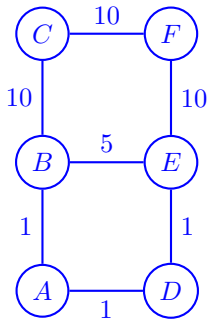
(b) If e is part of some MST of G, then it must be a lightest edge across some cut of G.

(c) If G has a cycle with a unique lightest edge e, then e must be part of every MST.

(d) For any $r > 0$, define an r-path to be a path whose edges all have weight less than r. If G contains an r-path from s to t, then every MST of G must also contain an r-path from s to t.

Solution:

1. True, e will belong to the MST produced by Kruskal.
2. True, suppose (u, v) is the edge. Let one side of the cut be everything reachable in the MST from u without using the edge (u, v). If this cut has an edge lighter than (u, v) then we could add this edge to the MST and remove (u, v). We know this edge is not already in the MST because otherwise both its endpoints would be reachable from u without using (u, v).
3. False. Let e be also the heaviest edge of a different cycle; then, we know that e can't be part of the MST. Concretely, in the following graph, edge (B, E) satisfies this condition, but will not be added to the graph.



4. True. Let $v_1, v_2, \ldots, v_n$ denote the r-path in G from $s = v_1$ to $t = v_n$. Consider the greatest i such that the MST has an r-path from s to v_i and assume for contradiction that $i < n$. Then the edge (v_i, v_{i+1}) is not in the MST. Adding this edge to the MST forms a cycle, which must have edges of length less than r since otherwise we could replace one of them with (v_i, v_{i+1}). Thus there is an r-path from v_i to v_{i+1} and hence from s to v_{i+1}, contradicting our initial assumption.

2 Updating a MST

You are given a graph $G = (V, E)$ with positive edge weights, and a minimum spanning tree $T = (V, E')$ with respect to these weights; you may assume G and T are given as adjacency lists. Now suppose the weight of a particular edge $e \in E$ is modified from $w(e)$ to a new value $\hat{w}(e)$. You wish to quickly

update the minimum spanning tree T to reflect this change, without recomputing the entire tree from scratch. There are four cases. In each, give a description of an algorithm for updating T, a proof of correctness, and a runtime analysis for the algorithm. Note that for some of the cases these may be quite brief.

(a) $e \notin E'$ and $\hat{w}(e) > w(e)$

(b) $e \notin E'$ and $\hat{w}(e) < w(e)$

(c) $e \in E'$ and $\hat{w}(e) < w(e)$

(d) $e \in E'$ and $\hat{w}(e) > w(e)$

Solution:

(a) **Main Idea:** Do nothing.

Correctness: T's weight does not increase, and any other spanning tree's weight either stays the same or increases, so T must still be an MST.

Runtime: Doing nothing takes $O(1)$ time.

(b) **Main Idea:** Add e to T. Use DFS to find the cycle that now exists in T. Remove the heaviest edge in the cycle from T.

Correctness: The heaviest edge in a cycle is safe to exclude from the MST (because if it is in the MST, you can remove it from the MST and add some other edge to the MST, and the MST's cost will not increase), and any edge not in an MST is the heavest edge in some cycle (in particular, the cycle formed by adding it to the MST). For any edge not in T except for e, decreasing e's weight does not change that it is the heaviest edge in the cycle, so it is safe to exclude from the MST. By adding e to T and then removing the heaviest edge in the cycle in T, we remove an edge that is also safe to exclude from the MST. Thus after this update, all edges outside of T are safe to exclude from the MST.

Runtime: This takes $O(|V|)$ time since T has $|V|$ edges after adding e, so the DFS runs in $O(|V|)$ time.

(c) **Main Idea:** Do nothing.

Correctness: T's weight decreases by $w(e) - \hat{w}(e)$, and any other spanning tree's weight either stays the same or also decreases by this much, so T must still be an MST.

Runtime: Doing nothing takes $O(1)$ time.

(d) **Main Idea:** Delete e from T. Now T has two components, A and B. Find the lightest edge with one endpoint in each of A and B, and add this edge to T.

Correctness: Every edge besides e in the MST is a lightest edge in some cut prior to changing e's weight, and increasing e's weight cannot affect this property. So all edges besides e are safe to keep in the MST. Then, whatever edge we add is also the lightest edge in the cut (A, B) and is thus also safe to include in the MST.

Runtime: This takes $O(|V| + |E|)$ time, since it might be the case that almost all edges in the graph might have one endpoint in both A and B and thus almost all edges will be looked at.

3 A Divide and Conquer Algorithm for MST

Is the following algorithm correct? If so, prove it. Otherwise, give a counterexample and **explain why it doesn't work**.

procedure FINDMST(G: graph on n vertices)
If $n = 1$ return the empty set
$T_1 \leftarrow$ FindMST(G_1: subgraph of G induced on vertices $\{1, \ldots, n/2\}$)
$T_2 \leftarrow$ FindMST(G_2: subgraph of G induced on vertices $\{n/2 + 1, \ldots, n\}$)
$e \leftarrow$ cheapest edge across the cut $\{1, \ldots, \frac{n}{2}\}$ and $\{\frac{n}{2} + 1, \ldots, n\}$.
return $T_1 \cup T_2 \cup \{e\}$.

Solution: This algorithm does not work; multiple edges of the MST could cross this particular cut. Another way to see this is that the MSTs of the subgraph needn't also be part of the MST of the whole graph.
As a concrete counterexample, consider a wide rectangle and the horizontal cut between the top two vertices and the bottom two. Both edges on this cut should be in the MST.

It is also possible that when you divide the graph it is not possible to construct a MST of the subsection of the graph.

4 Huffman Proofs

(a) Prove that in the Huffman coding scheme, if some symbol occurs with frequency more than $\frac{2}{5}$, then there is guaranteed to be a codeword of length 1. Also prove that if all symbols occur with frequency less than $\frac{1}{3}$, then there is guaranteed to be no codeword of length 1.

(b) Suppose that our alphabet consists of n symbols. What is the longest possible encoding of a single symbol under the Huffman code? What set of frequencies yields such an encoding?

Solution:

1. Suppose all codewords have length at least 2 – the tree must have at least 2 levels. Let the weight of a node be the sum of the frequencies of all leaves that can be reached from that node. Suppose the weights of the level-2 nodes are (from left to right for a tree rooted on top) a, b, c, and d. Without loss of generality, assume A and B are joined first, then C and D. So, $a, b \leq c, d \leq a+b$.

   If a or b is greater than $\frac{2}{5}$, then both c and d are greater than $\frac{2}{5}$, so $a + b + c + d > \frac{6}{5} > 1$ (impossible). Now suppose c is greater than $\frac{2}{5}$ (similar idea if d is greater than $\frac{2}{5}$). Then $a + b > \frac{2}{5}$, so either $a > \frac{1}{5}$ or $b > \frac{1}{5}$ which implies $d > \frac{1}{5}$. We obtain $a + b + c + d > 1$ (impossible).

   For the second part suppose there is a codeword of length 1. We have 3 cases. Either the tree will consist of 1 single level-1 leaf node, 2 level-1 leaf nodes, or 1 level-1 leaf node, and 2 level-2 nodes with an arbitrary number of leaves below them in the tree. We will prove the contrapositive of the original statement, that is, that if there is a codeword of length 1, there must be a node with frequency greater than $\frac{1}{3}$

   In the first case, our leaf must have frequency 1, so we've immediately found a leaf of frequency more than $\frac{1}{3}$. If the tree has two nodes, one of them has frequency at least $\frac{1}{2}$, so condition is again satsified. In the last case, the tree has one level-1 leaf (weight a), and two level-2 nodes (weights c and d). We have: $b, c \leq a$. So $1 = a + b + c \leq 3a$, or $a \geq \frac{1}{3}$. Again, our condition has been satisfied.

2. The longest codeword can be of length $n - 1$. An encoding of n symbols with $n - 2$ of them having probabilities $1/2, 1/4, \ldots, 1/2^{n-2}$ and two of them having probability $1/2^{n-1}$ achieves this value. No codeword can ever by longer than length $n - 1$. To see why, we consider a prefix tree of the code. If a codeword has length n or greater, then the prefix tree would have height n or greater, so it would have at least $n + 1$ leaves. Our alphabet is of size n, so the prefix tree has exactly n leaves.