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 Horn Formula Practice

(a) Find the variable assignment that solves the following horn formula:

$$(x \wedge z) \Rightarrow y, z \Rightarrow w, (y \wedge z) \Rightarrow x, \Rightarrow z, (\bar{z} \vee \bar{x}), (\bar{w} \vee \bar{y} \vee \bar{z})$$

(b) Show that any implication clause of the form $(x_i \wedge x_j \wedge \cdots) \Rightarrow \mathsf{True}$ is always satisfiable.
Hint: what disjunction clause is this equivalent to?

(c) Show that any implication clause of the form $\mathsf{False} \Rightarrow x_k$ is always satisfiable.

Solution:

(a)
- z must be true since we have the statement $\Rightarrow z$.
- w must be true since $z \Rightarrow w$
- x and y need not be changed, as all our implications are satisfied.
- All negative clauses are now satisfied, so we've found our satisfying assignment.

(b) This clause is satisfied no matter the assigment to the variables because it is equivalent to $(\mathsf{True} \vee \overline{x_i} \vee \overline{x_j} \ldots)$.

(c) This clause is equivalent to $(x_k \vee \mathsf{True})$ so is always satisfied no matter what x_k is.

2 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.

(b) Prove that in the Huffman coding scheme, if all symbols occur with frequency less than $\frac{1}{3}$, then there is guaranteed to be no codeword of length 1.

(c) 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).

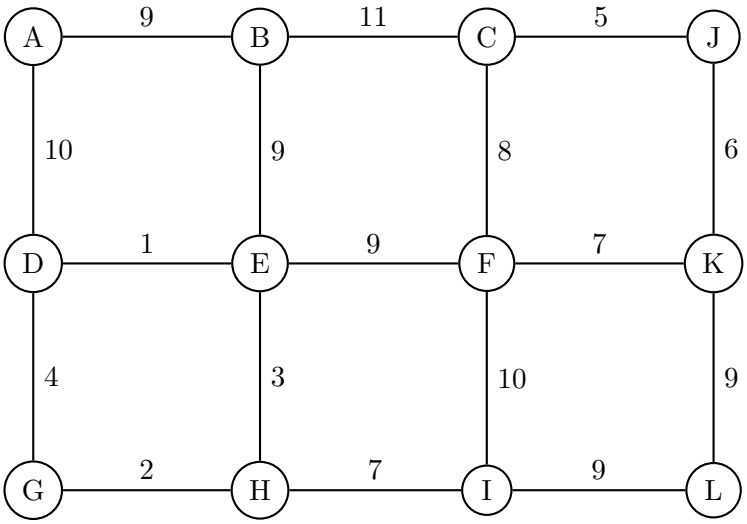
2. Suppose there is a codeword of length 1. We have 3 cases. Either the tree will consist of 1 single level-1 leaf node, 2 level-1 leaf nodes, or 1 level-1 leaf node, and 2 level-2 nodes with an arbitrary number of leaves below them in the tree. We will prove the contrapositive of the original statement, 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.

3. 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.

3 MST Tutorial



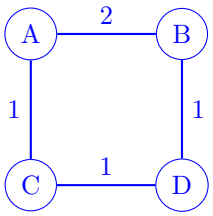
(a) List the first **six** edges added by Prim's algorithm in the order in which they are added. Assume that Prim's algorithm starts at vertex A and breaks ties lexicographically.

Solution: AB, BE, DE, EH, GH, HI

(b) List the first **seven** edges added by Kruskal's algorithm in the order in which they are added. You may break ties in any way.

Solution: (DE, GH, EH, CJ, JK, FK, HI) **or** (DE, GH, EH, CJ, JK, HI, FK)

(c) Prim's algorithm is very similar to Dijkstra's in that a vertex is processed at each step which minimizes some cost function. These algorithms also produce similar outputs: the union of all shortest paths produced by a run of Dijkstra's algorithm forms a tree. However, the trees they produce aren't optimizing for the same thing. To see this, give an example of a graph for which different trees are produced by running Prim's algorithm and Dijkstra's algorithm. In other words, give a graph where there is a shortest path from a start vertex A using at least one edge that doesn't appear in any MST.



Solution:

The MST is $\{(A, C), (C, D), (B, D)\}$, but the shortest path from A to B is through the weight 2 edge.

4 MST Potpourri

(a) Given an undirected graph $G = (V, E)$ and a set $E' \subset E$ briefly describe how to update Kruskal's algorithm to find the minimum spanning tree that includes all edges from E'.

(b) Suppose we want to find the minimum cost set of edges that suffices to connect a given weighted graph $G = (V, E)$; if the weights are non-negative then we know that the optimum will be a MST. What about the case when the weights are allowed to be negative? Does it have to be a tree if the weights are allowed to be negative? If not, how would you find this minimum-cost connected subgraph?

(c) Describe an algorithm to find a maximum spanning tree of a given graph.

Solution:

(a) Assuming E' doesn't have a cycle, add all edges from E' to the MST first, then sort $E \backslash E'$ and run Kruskal's as normal.

(b) In case the weights are allowed to be negative, the minimum-cost connected subgraph is not necessarily a tree; consider for example the cycle graph on n vertices with each edge having negative weight. The minimum-cost connected subgraph is the complete graph since removing any single edge only increases the cost of the selected component.
Let us denote this component by $G' = (V, E')$ where $E' = \phi$ is the set of empty edges intially. Add all the negative weight edges to E'. If the component is still not connected, run Kruskal's algorithm beginning from this set E' of the edges.

(c) Negate all edge weights and apply MST algorithm by scaling up all edge weights to ensure they are all positive.

5 Penguin Phone Plans

The PNPenguins' ice-cream selling business has really taken off, and they would really like to share the great news with all their n friends. The PNPenguins have friends who live internationally, so having to call each friend individually could be expensive, as the price of phone calls increases with distance. Luckily, some of their n friends also know each other, so they can call one another and share the news (gossip!). For two people i and j, the PNPenguins know if i and j are friends and know the distance $d(i, j)$ between them.

(a) Help the PNPenguins design an algorithm to determine how phone calls should be made to minimize the total phone bill when sharing the news with all their friends. Assume that the group of PNPenguins call their friends together.

Solution: MST!

Main Idea: Construct an undirected graph with $n + 1$ nodes. One node represents the PNPenguins themselves, and the remaining n nodes represent each of their friends. Add an edge with weight $d(i, j)$ for each pair of friends that knows each other. Run Prim's or Kruskal's to find the minimum spanning tree.

Proof of Correctness: Kruskal's algorithm iterates through edges in ascending order, including edges if they don't create a cycle. An MST on a connected graph (which it is connected here, since they are all PNPenguins' friends) minimizes the sum of edge weights while keeping the graph connected. Since the price of a phone call increases with distance, labeling edges with the distance does not change the ordering in which the edges are chosen by Kruskal's.

Runtime Analysis: Constructing the graph takes $O(|V| + |E|)$ time. MST finding takes $O(|E| \log |V|)$ time. So the total runtime is $O(|V| + |E| + |E| \log |V|)$.

(b) Seeing the pattern of phone calls, P-Mobile (a telecommunications company) has changed their phone plan so that the price of calls within a certain distance are halved, and calls above a certain distance are doubled. Does the solution returned from part (a) change? Briefly justify.

Solution: No. The relative ordering of the edges does not change with the new plan, so Kruskal's algorithm will choose the same edges and the solution from part (a) remains the same.

(c) Following their success in the Antarctic, PNPenguins have decided to expand their business into the Arctic, stationing at least one penguin in each of the m regions of the Arctic. Hearing this, P-Mobile has decided to expand into the Arctic as well. P-Mobile knows that the PNPenguins have to make phone calls to coordinate their ice-cream selling business across all regions of the Arctic. P-Mobile has decided that calls within a region should be free, and calls between

different regions should be priced proportionally to the distance between the regions. They have also decided to only allow calls between certain specific regions. Design an algorithm for P-Mobile to find a connection between regions that maximizes revenue from PNPenguin calls.

Solution: More MST!

Main Idea: Construct a graph with m nodes, each representing a region of the Arctic. Between each of the two nodes, add an edge with an edge weight representing the distance between them. Call this graph G. Negate all the edges to create G'. Run Kruskal's on Prim's on G_n to find its minimum spanning tree. Output all the edges in this tree.

Correctness: The maximum revenue is obtained from a maximum spanning tree of the graph. We can have negative edges and return a correct MST, as Kruskal's / Prim's do not rely on non-negative edge weights to find the MST. We correctly find the maximum spanning tree, since $\text{argmax}_G \sum_{e \in G} w_e = \text{argmin}_G - \sum_{e \in G} w_e = \text{argmin}_{G_n} \sum_{e \in G} w_e$.

Runtime: Negating all the edges of the graph takes $O(|V| + |E|)$ time. MST finding takes $O(|E| \log |V|)$ time. So the total runtime is $O(|V| + |E| + |E| \log |V|)$

6 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. For simplicity, you may assume that no two edges have the same weight (this applies to both w and $\hat{w}$).

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

Solution:

(a) **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.

(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 cannot be in 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 decrease), 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 not in the MST. Thus after this update, all edges outside of T cannot be in the MST, so T is 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:** 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 the 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 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.

(d) **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 the MST.

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