Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 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.

2 Finding Counterexamples

In this problem, we give example greedy algorithms for various problems, and your goal is to find an example where they are not optimal.

(a) In the travelling salesman problem, we have a weighted undirected graph $G(V, E)$ with all possible edges. Our goal is to find the cycle that visits all the vertices exactly once with minimum length.

One greedy algorithm is: Build the cycle starting from an arbitrary start point s, and initialize the set of visited vertices to just s. At each step, if we are currently at vertex u and our cycle has not visited all the vertices yet, add the shortest edge from u to an unvisited vertex v to the cycle, and then move to v and mark v as visited. Otherwise, add an edge from the current vertex to s to the cycle, and return the now complete cycle.

(b) In the maximum matching problem, we have an undirected graph $G(V, E)$ and our goal is to find the largest matching E' in E, i.e. the largest subset E' of E such that no two edges in E' share an endpoint.

One greedy algorithm is: While there is an edge $e = (u, v)$ in E such that neither u or v is already an endpoint of an edge in E', add any such edge to E'. (Can you prove that this algorithm still finds a solution whose size is at least half the size of the best solution?)

Solution:
Note: For each part, there are many counterexamples.

(a) One counterexample is to have a four vertex graph with vertices a, b, c, d, where the edges $(a, b), (b, c), (c, d)$ cost 1, the edge (a, d) costs 100, and all other edges cost 2. If the greedy algorithm starts at vertex a, it will add the edges $(a, b), (b, c), (c, d)$ to the cycle, and then be forced to add the very expensive edge (a, d) at the end to find a cycle of cost 103. The optimal cycle is $(a, b), (b, d), (d, c), (c, a)$ which costs 6. The key idea here is that by using a path of low-weight edges, we forced the algorithm into a position where it had to pick a high-weight edge to complete its solution.

(b) The simplest example is a path graph with three edges. The greedy algorithm will pick the middle edge, the optimal solution is to pick the two outer edges.

To show this algorithm always finds a matching at least half the size of the best solution, let the size of the solution found by the algorithm be m. Any edge in the best solution shares an endpoint with one of the edges in the algorithm's solution (otherwise, the greedy algorithm could have added it). None of the edges in the best solution can share an endpoint, and the m edges in the algorithm's solution have $2m$ endpoints, so the best solution must have at most $2m$ edges.

3 Service scheduling

A server has n customers waiting to be served. Customer i requires t_i minutes to be served. If, for example, the customers were served in the order $t_1, t_2, t_3, \ldots, t_n$, then the i-th customer would wait for $t_1 + t_2 + \cdots + t_i$ minutes.

We want to minimize the total waiting time

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)\,.$$

Given the list of the t_i's, give an efficient algorithm for computing the optimal order in which to serve the customers.

Solution: We use a greedy strategy, by sorting the customers in increasing order of service times and serving them in this order. The running time is $O(n \log n)$.
To prove correctness, for any ordering of the customers, let $s(j)$ denote the j-th customer in the ordering. Then

$$T = \sum_{i=1}^{n} \sum_{j=1}^{i-1} t_{s(j)} = \sum_{i=1}^{n} (n - i) t_{s(i)}\,.$$

For any ordering, if $t_{s(i)} > t_{s(j)}$ for $i < j$, then swapping the positions of the two customers gives a better ordering. Since we can generate all possible orderings by swaps, an ordering which has the property that $t_{s(1)} \leq \ldots \leq t_{s(n)}$ must be the global optimum. This is exactly the ordering that we output.

4 Finding MSTs by Deleting Edges

Consider the following algorithm to find the minimum spanning tree of an undirected, weighted graph $G(V, E)$. For simplicity, you may assume that no two edges in G have the same weight.

```
procedure FindMST($G(V, E)$)
    $E' \leftarrow E$
    for Each edge $e$ in $E$ in decreasing weight order do
        if $G(V, E' - e)$ is connected then
            $E' \leftarrow E' - e$
    return $E'$
```

Show that this algorithm outputs a minimum spanning tree of G.

Solution:

There are several solutions. One is to note that, anytime the algorithm chooses not to delete an edge e, that edge must be a lightest edge across some cut. In particular, the cut is the two components of the disconnected graph $E' - e$ (using the value of E' at the start of the iteration where the algorithm looks at e). The algorithm is also guaranteed to output E' containing no cycles, so by applying the cut property we get that it outputs a minimum spanning tree.

Other proofs include the use of the cycle property. As a review, the cycle property claims that the heaviest edge in any cycle in G cannot appear in the (unique) minimum spanning tree. To prove the cycle property, suppose e is the heaviest edge in some cycle C and is in the minimum spanning tree T^*. Consider deleting e from the minimum spanning tree to get $T^* - e$. $T^* - e$ has two components, and some edge e' in C other than e must connect the two components. So $T^* - e + e'$ is a spanning tree, and costs less than T^* since e' costs less than e, a contradiction.

The algorithm is guaranteed to output a spanning tree T. Suppose that the MST $T*$ is not T, and let $e \in T * -T$. Then $T \cup e$ contains a cycle; denote by e' its heaviest edge, and note that $e \neq e'$ by the cycle property. When we considered e', all edges in $T \cup e$ were still there, and so we should have deleted e' since removing it would leave the graph connected.

An alternative proof is to show that every edge we delete is the heaviest edge in some cycle. This is because whenever we delete an edge e, it is part of some cycle in the remaining edges in E' since E' remains connected after deleting e. No other edge in this cycle can be heavier than e, otherwise we would have deleted that edge first. So by the cycle property we have only deleted edges that do not appear in the minimum spanning tree. Furthermore, note that this algorithm will eliminate all cycles from T. So we know the final solution is a tree, and thus must be the minimum spanning tree.