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 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?

2 Doctor

A doctor's office has n customers, labeled $1, 2, \ldots, n$, waiting to be seen. They are all present right now and will wait until the doctor can see them. The doctor can see one customer at a time, and we can predict exactly how much time each customer will need with the doctor: customer i will take $t(i)$ minutes.

(a) We want to minimize the average waiting time (the average of the amount of time each customer waits before they are seen, not counting the time they spend with the doctor). What order should we use? You do not need to justify your answer for this part. (Hint: sort the customers by ____)

(b) Let $x_1, x_2, \ldots, x_n$ denote an ordering of the customers (so we see customer x_1 first, then customer x_2, and so on). Prove that the following modification, if applied to any order, will never increase the average waiting time:

- If $i < j$ and $t(x_i) \geq t(x_j)$, swap customer i with customer j.

(For example, if the order of customers is $3, 1, 4, 2$ and $t(3) \geq t(4)$, then applying this rule with $i = 1$ and $j = 3$ gives us the new order $4, 1, 3, 2$.)

(c) Let u be the ordering of customers you selected in part (a), and x be any other ordering. Prove that the average waiting time of u is no larger than the average waiting time of x—and therefore your answer in part (a) is optimal.

Hint: Let i be the smallest index such that $u_i \neq x_i$. Use what you learned in part (b). Then, use proof by induction (maybe backwards, in the order $i = n, n-1, n-2, \ldots, 1$, or in some other way).

3 Activity Selection

Assume there are n activities each with its own start time a_i and end time b_i such that $a_i < b_i$. All these activities share a common resource (think computers trying to use the same printer). A feasible schedule of the activities is one such that no two activities are using the common resource simultaneously. Mathematically, the time intervals are disjoint: $(a_i, b_i) \cap (a_j, b_j) = \emptyset$. The goal is to find a feasible schedule that maximizes the number of activities k.

Here are two potential greedy algorithms for the problem.

Algorithm A: Select the shortest-duration activity that doesn't conflict with those already selected until no more can be selected.

Algorithm B: Select the earliest-ending activity that doesn't conflict with those already selected until no more can be selected.

(a) Show that Algorithm A can fail to produce an optimal output.

(b) Show that Algorithm B will always produce an optimal output. (Hint: To prove correctness, show how to take any other solution S and repeatedly swap one of the activities used by Algorithm B into S while maintaining that S has no overlaps)

(c) **Challenge Problem:** Show that Algorithm A will always produce an output at least half as large as the optimal output.

4 Finding Counterexamples

In this problem, we give example greedy algorithms for various problems, and your goal is to find a counterexample where they do not find the best solution.

(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'. (Challenge: Can you prove that this algorithm still finds a solution whose size is at least half the size of the best solution?)