1 Longest Huffman Tree

Under a Huffman encoding of $n$ symbols with frequencies $f_1, f_2, \ldots, f_n$, what is the longest a codeword could possibly be? Give an example set of frequencies that would produce this case, and argue that it is the longest possible.

2 Twenty Questions

Your friend challenges you to a variant of the guessing game 20 questions. First, they pick some word ($w_1, w_2, \ldots, w_n$) according to a known probability distribution ($p_1, p_2, \ldots, p_n$), i.e. word $w_i$ is chosen with probability $p_i$. Then, you ask yes/no questions until you are certain which word has been chosen. You can ask any yes/no question, meaning you can eliminate any subset $S$ of the possible words with the question “Is the word in $S$?”.

Define the cost of a guessing strategy as the expected number of queries it requires to determine the chosen word, and let an optimal strategy be one which minimizes cost. Design an $O(n \log n)$ algorithm to determine the cost of the optimal strategy.

**Note:** We are only considering deterministic guessing strategies in this question. Including randomized strategies doesn’t change the answer, but it makes the proof of correctness more difficult.
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.

Hint: Show that every activity in the optimal solution overlaps with at least 1 activity in Algorithm A’s solution, and any activity in Algorithm A’s solution can overlap with at most 2 activities in the optimal solution.
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?)
5 Union find with path compression

Recall the union find data structure which we implemented via a disjoint-set forest using path compression. Consider the following sequence of operations.

1. First, we call makeset($x$) once on each letter in $\mathcal{L} = \{A, B, C, D, E, F, G, H\}$.
2. Next, we perform a sequence of union($x, y$)'s, where at each step both $x$ and $y$ are the roots of their respective sets.
3. Finally, we perform a single find($x$), on a letter $x \in \mathcal{L}$ which may or may not be the root of its set.

Suppose after these operations are complete, the data structure looks as follows:

In this diagram, $x^i$ means that the node contains the letter $x$ and has rank $i$.

1. Draw what the directed tree looked like before the find($x$) in step 3.

2. For which letter $x \in \mathcal{L}$ was find($x$) called in step 3?
6 When the Levee Breaks

A levee broke and the water is flooding a group of villages. There are \( n \) villages and \( m \) roads between the villages. The water floods one road at a time. For each road \( e_i \), you’re given the unique integer timestep \( t_i \) of when this road is flooded. \( (t_i)_{i=1}^m \) is a permutation of \( \{1\ldots m\} \). Give an efficient algorithm that computes the array \( A \), where \( A_i \) denotes the number of connected components of villages at timestep \( i \in \{1\ldots m\} \) and then analyze its runtime. **Proof of correctness is not needed for this question.**

Example:

\[
\begin{align*}
n &= 4, m = 4. \\
\text{The roads } (e_i)_{i=1}^4 &= (1, 2); (2, 3); (3, 4); (4, 1) \\
\text{Timesteps } (t_i)_{i=1}^4 &= 3; 2; 4; 1
\end{align*}
\]

At timestep 1, \( e_4 = (1, 4) \) is flooded, there’s 1 connected component: \( \{1, 2, 3, 4\} \).
At timestep 2, \( e_4 = (1, 4) \) and \( e_2 = (2, 3) \) are flooded, there are 2 connected components: \( \{1, 2\}, \{3, 4\} \).
At timestep 3, \( e_4, e_2, \) and \( e_1 \) are flooded, there are 3 connected components: \( \{1\}, \{2\}, \{3, 4\} \).
At timestep 4, all roads are flooded, there are 4 connected components: \( \{1\}, \{2\}, \{3\}, \{4\} \).
Therefore \( A = (1, 2, 3, 4) \)

---

1 All villagers have evacuated so there are no casualties.
2 A group of villages in which any two villages are connected to each other by unflooded roads, and which is connected to no additional villages in the rest of the \( n \) total villages.