1 Huffman Proofs

(a) Prove that in the Huffman coding scheme, if some symbol occurs with frequency more than $\frac{2}{3}$, 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 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?)
3 Planting Trees

This problem will guide you through the process of writing a dynamic programming algorithm.

You have a garden and want to plant some apple trees in your garden, so that they produce as many apples as possible. There are \( n \) adjacent spots numbered 1 to \( n \) in your garden where you can place a tree. Based on the quality of the soil in each spot, you know that if you plant a tree in the \( i \)th spot, it will produce exactly \( x_i \) apples. However, each tree needs space to grow, so if you place a tree in the \( i \)th spot, you can’t place a tree in spots \( i - 1 \) or \( i + 1 \). What is the maximum number of apples you can produce in your garden?

(a) Give an example of an input for which:

- Starting from either the first or second spot and then picking every other spot (e.g. either planting the trees in spots 1, 3, 5... or in spots 2, 4, 6...) does not produce an optimal solution.
- The following algorithm does not produce an optimal solution: While it is possible to plant another tree, plant a tree in the spot where we are allowed to plant a tree with the largest \( x_i \) value.

(b) To solve this problem, we’ll thinking about solving the following, more general problem:

“What is the maximum number of apples that can be produced using only spots 1 to \( i \)?” Let \( f(i) \) denote the answer to this question for any \( i \). Define \( f(0) = 0 \), as when we have no spots, we can’t plant any trees. What is \( f(1) \)? What is \( f(2) \)?

(c) Suppose you know that the best way to plant trees using only spots 1 to \( i \) does not place a tree in spot \( i \). In this case, express \( f(i) \) in terms of \( x_i \) and \( f(j) \) for \( j < i \). (Hint: What spots are we left with? What is the best way to plant trees in these spots?)

(d) Suppose you know that the best way to plant trees using only spots 1 to \( i \) places a tree in spot \( i \). In this case, express \( f(i) \) in terms of \( x_i \) and \( f(j) \) for \( j < i \).
(e) Describe a linear-time algorithm to compute the maximum number of apples you can produce. (Hint: Compute $f(i)$ for every $i$. You should be able to combine your results from the previous two parts to perform each computation in $O(1)$ time).

4 Change making

You are given an unlimited supply of coins of denominations $v_1, \ldots, v_n \in N$ and a value $W \in N$. Your goal is to make change for $W$ using the minimum number of coins, that is, find a smallest set of coins whose total value is $W$.

1. Design a dynamic programming algorithm for solving the change making problem. What is its running time?

2. You now have the additional constraint that there is only one coin per denomination. Does your previous algorithm still work? If not, design a new one.