

Practice Problems 4

Problem 1. This problem is about demonstrating the limitations of the greedy algorithm for interval scheduling. In this problem, suppose you are optimizing over total time of scheduled intervals, rather than number of intervals.

1. Devise a counterexample where the “earliest finish first” greedy algorithm fails to find the optimal value.
2. Suppose instead that at every step you schedule the job that takes the longest time. Show how this greedy algorithm fails to find an optimal value with a specific counterexample.
3. Show that scheduling the job that takes the longest time at every step can be no more than 3 times worse than the ideal solution.

Problem 2. This problem concerns a greedy approximation algorithm for the load balancing problem. In this problem, you are given m machines and k jobs, and each job takes t_i time. You may schedule any job on any machine, and your goal is to minimize the total makespan, or the longest time taken by any machine for its jobs. You will be analyzing a greedy load-balancing algorithm which runs over the jobs in any order, and always adds the next job to the machine with the shortest individual makespan.

1. Prove that an optimal solution to the load balancing problem has makespan at least $\frac{1}{m} \sum_{i=1}^k t_i$
2. Prove that the greedy load balancing algorithm produces a solution which has makespan at most $2 \cdot O$, where O is the optimal makespan. (Hint: at the moment the last job is inserted, consider what the state of the algorithm must be.)

Problem 3. In this problem, you will prove the relationship between Huffman Coding and Stone Merging. For reference, in Stone Merging, you are given n piles of stones with sizes $a_1, \dots, a_n$, and must merge them into a single pile by merging two piles at a time. The cost of merging two piles is equal to the sum of the sizes of those piles, and you seek to merge them all with the lowest total cost.

A *merge tree* for a series of merges is a rooted binary tree built as follows: Each initial pile is a leaf of the tree, and each merge creates a new internal node which has two edges to the piles it is merging.

1. Draw the merge tree for the piles 2, 3, 5, 7, 11 using the “always merge the two smallest piles” strategy, recording each move and the total cost.
2. A **full binary tree** is a binary tree such that every internal node has exactly two children. Show that every full binary tree with n leaves corresponds to a merge sequence, and that every merge sequence produces a full binary tree.

3. For a tree T and a leaf v in T , let $d_T(v)$ be the depth of leaf v in T , where the root has depth 0. Prove that the total cost of a merge sequence is equal to $\sum_v a_v \cdot d_T(v)$, summed over the leaves v of the corresponding merge tree.
4. Let $x_1, \dots, x_n$ be the symbols of an alphabet, where symbol x_i occurs with frequency $a_i > 0$ in a text to be compressed. A **binary code** assigns to each symbol x_i a binary string $c(x_i)$, called its *codeword*; a text is encoded by concatenating the codewords of its symbols. The code is a **prefix code** if no codeword is a prefix of another codeword. For example, $\{0, 10, 11\}$ is a prefix code, but $\{0, 01, 10\}$ is not, since 0 is a prefix of 01. The *cost* of a prefix code is the total length of the encoded text, $\sum_{i=1}^n a_i \cdot |c(x_i)|$, and an **optimal prefix code** is one of minimum cost.
 - (i) Suppose the code is not a prefix code, say $c(x_1) = 0$, $c(x_2) = 01$, and $c(x_3) = 10$. Give an encoded string that can be decoded in more than one way. Then explain why, for a prefix code, an encoded string can always be decoded unambiguously by reading it from left to right.
 - (ii) Consider four symbols x_1, x_2, x_3, x_4 with frequencies $a_1 = 45$, $a_2 = 13$, $a_3 = 12$, $a_4 = 30$, and the prefix code $c(x_1) = 0$, $c(x_2) = 110$, $c(x_3) = 111$, $c(x_4) = 10$. Compute the cost of this code. How does it compare to the cost of the fixed-length code that assigns every symbol a two-bit codeword?

A prefix code can be represented by a binary tree whose leaves are the symbols: each left edge is labeled 0 and each right edge 1, and the codeword of a symbol is the sequence of labels on the path from the root to its leaf. For example, a leaf reached by going left, then right, then left has codeword 010. Since every symbol is a leaf, no codeword is a prefix of another, and the length of a codeword equals the depth of its leaf.

 - (iii) Using the previous parts, devise an algorithm that computes an optimal prefix code for given frequencies $a_1, \dots, a_n$, and justify its correctness. Check that it produces a code of the same cost as the one in part (ii).