Foundations of Algorithms, Spring 2023: Homework 3

Due: Wednesday, March 1, 11:59pm

Homework will be graded out of 67; up to 71 points can be earned

Problem 1 (8 points)

An alphabet contains symbols A, B, C, D, E, F. The frequencies of the symbols are 35%, 20%, 15%, 15%, 8%, and 7%, respectively. We know that the Huffman algorithm always outputs an optimal prefix code. However, this code is not always unique (obviously we can, e.g., switch 0's and 1's and get a different code - but, for some inputs, there are two optimal prefix codes that are more substantially different). For the purposes of this exercise, we consider two Huffman codes to be different if there exists a symbol for which one of the codes assigns a shorter codeword than the other code.

- Trace the Huffman algorithm and construct two different Huffman codes for the above input.
- Compute the expected number of bits per symbol (i.e., the expected codeword length, appropriately weighted) for both codes.

Problem 2 (6 points)

Consider the following "minimize gaps" interval scheduling problem: Given is a global start time S and finish time F. Given also is a set of n intervals where the i-th interval starts at time s_i and ends at time f_i, $S \leq s_i \leq f_i \leq F$, for $i \in \{0, \ldots, n-1\}$. Find a set of non-overlapping intervals so that the overall time from S to F *not* covered by the selected intervals is as small as possible (we refer to this time as *gap time*).

Example: $S = 0$, $F = 10$, and the intervals are $(1, 4), (2, 7), (4, 5), (6, 10), (8, 9)$. If we select intervals $(1, 4), (4, 5), (6, 10)$, then the time between 0 and 10 that is not covered by the intervals is $(0, 1)$ and $(5, 6)$ – total time 2. This is the smallest possible overall gap time. Note that in the case when the end time of one interval equals the start time of a second interval, those two intervals are considered to be compatible (e.g. $(1, 4)$ and $(4, 5)$).

Consider the following greedy strategies for this problem:

1. Select the earliest finishing interval and discard overlapping intervals. Keep doing this until all intervals have been eliminated (either selected or discarded).
2. Select the earliest starting interval and discard overlapping intervals. Keep doing this until all intervals have been eliminated (either selected or discarded).
3. Select the pair of non-overlapping intervals that have the smallest gap between them: find a pair of intervals $i \neq j$ such that $s_j - f_i \geq 0$ is the smallest possible. Select both intervals and discard overlapping intervals. Recursively do the same selection process with intervals that finish at or before s_i: the recursive call will have $S_{\text{new}} = S$ and $F_{\text{new}} = s_i$. Similarly, recursively do the same selection process with intervals that

start at or after f_j: now $S_{\text{new}} = f_j$ and $F_{\text{new}} = F$. If there is no such pair of intervals, select a single interval that minimizes the gap between S and F (do not make any further recursive calls).

None of these strategies works all the time. Find a counterexample for each strategy. In other words, for each strategy,

- find a set of intervals and S and F so that the strategy does not produce an optimal solution,
- highlight intervals selected by the strategy and state the corresponding gap time, and
- highlight intervals in an optimal solution (one that minimizes the overall gap time) and state the corresponding gap time.

Problem 3 (18 points: 14 for implementation / 4 for writeup / 4 bonus possible)

Given are n distinct points in 2-dimensional space. Design an $O(n^2 \log n)$ algorithm that determines whether any three of these points are evenly spaced, co-linear points. A $\Theta(n^3)$ algorithm can earn at most 8 out of 18 marks.

BONUS: only can be earned after you successfully complete this problem in $O(n^2 \log n)$ complexity. Additionally, design a solution that runs in *expected* $O(n^2)$ time. You may use a hash-table data structure for this part of the problem, and only for this part of the problem. Document the performance of this approach compared to your $O(n^2 \log n)$ solution.

Problem 4 (20 points: 15 for implementation / 5 for writeup)

Don't rock the boat! The school kids, chaperoned by two teachers, have an exciting field trip planned on the open water. All of the kids would like to sit on the left side of the boat, because the view of the scenery will be better from that side. But to keep the boat afloat, it's necessary to distribute the weight evenly between the left and right side of the boat. The teachers decide to let the youngest kids get the preferred side of the boat. Of course, one teacher has to also be on each side of the boat to chaperone!

Design an $O(n)$ algorithm that determines the number of youngest kids that end up sitting on the left side while balancing the boat as much as possible.

Hint: start out by asking yourself how you would solve this problem if the children were already sorted by increasing age. You can't afford to sort the data, however, as none of the $O(n)$ sorts applies (you can't make any assumptions about the distribution of the age values).

Problem 5 (15 points: 10 for implementation / 5 for writeup)

Suppose that instead of encoding with a prefix code such as determined in Problem 1, the symbols are encoded using the codewords given in the table below:

Symbol	Codeword
A	0
B	1
C	10
D	01
E	111
F	011

Given an input encoding of size n bits, design an $O(n)$ algorithm that determines how many different decodings of those n bits are possible using the table above.

Note: Heart of the Algorithm

For any problem in which you develop a dynamic program (in this or any future homework!), to explain how your algorithm works, describe the "heart of the algorithm" (you do not need to include any other explanation). Recall that the heart of the algorithm consists of three parts:

- Precise verbal description of the meaning of your dynamic programming array; see the slides for examples.
- A mathematical formula that describes how to compute the value of each cell of the array (using previously computed array values).
- Return value of the algorithm.

This will satisfy the "correctness" argument for your algorithm. You will still need to provide some additional analysis for the running time.