# CS302 - Problem Set 7

1. Finish the proof of Lemma 1 from the proof of Huffman's algorithm from class:

   **Lemma:** Given an alphabet $\Sigma$ and probabilities $p(i)$ for each $i \in \Sigma$, let $a$ and $b$ be the letters with the smallest probabilities. Then there is an optimal tree (a tree corresponding to a code with smallest average letter length) where $a$ and $b$ are siblings.

   **Proof:** Suppose for contradiction that there are no optimal trees where $a$ and $b$ are siblings. Let $T'$ be an optimal tree where $a$, $b$ are not siblings. Let $x, y \in \Sigma$ be letters that are sibling leaves in $T'$ that also have maximum depth. (Note that at least one of $x$ or $y$ must not equal $a$ or $b$, because otherwise $a$ and $b$ would be siblings. Also there must be a pair of siblings at maximum depth because if there is only one leaf at maximum depth, then you can relocate that symbol to be at its parent node while maintaining a prefix free tree, which would then result in a tree with smaller average length than $T'$, contradicting the fact that $T'$ is optimal.)

   Let $d$ be the depth of $x$ and $y$ in $T'$, and let $d(i)$ be the depth of any other letter $i$ in $T'$.

   Consider the tree $T'_{ex}$ that we get by exchanging the positions of $a$ and $x$ and exchanging the positions of $b$ and $y$, while leaving all other letters in the same positions. Then ...

2. Suppose you run a company with two offices, one in Washington, D.C. and the other in San Francisco, California. You always spend the whole week in one location, but each weekend, you decide whether to fly to the other office. In week $i$, you can make $W_i$ dollars if you are in Washington, and $C_i$ dollars if you are in California. Each flight from one office to the other costs \$1000. Suppose you are initially in D.C., and you are given the lists of potential profits for each location: $W_1, W_2, \ldots, W_n$ and $C_1, C_2, \ldots, C_n$. Additionally, you are given $L_f \in \{C, W\}$, your final location, which is the location that you must be in after the $n$th week. This means if you spent the $n$th week in California, and $L_f = W$ you must spend a final \$1000 to fly back at the end of the $n$th week. In this problem, you will come up with an algorithm to maximize your profit.

   (a) Comment on societal/ethical impacts of this algorithm (think environmental, personal, etc.)

   (b) A greedy algorithm would look at how much money you would make in each week, and work in the place that will earn the most that week (after travel costs).

Give a counter example to show why this greedy algorithm is not always optimal. (Your example should have $n \leq 3$, otherwise a later part of this problem will be more challenging.)

(c) Create a dynamic programming algorithm (follow our usual steps, as described below, or for more challenge try to do as much on your own without my guidance/hints.)

i. Identify the likely subproblems and recurrence objects. (Pre-Hint Hint: you need two sequences of subproblems, depending on whether you have to end up in DC or California)

ii. Determine the relevant final options for the recurrence object and write a recurrence relation for your recurrence object for each option. Also determine base case(s). Briefly explain. (Partial solution in hints if stuck.)

iii. Turn your recurrence relation for your recurrence object into a recurrence relation for the objective function value.

iv. Fill in pseudocode for an algorithm to determine the optimal schedule:

**Algorithm 1:** `CommuteSchedule`$(M, L_f)$

**Input** : Array $M$ of size $n \times 2$, where $M[i, 0]$ is the money to be earned in $W$ in week $i$, $M[i, 1]$ is the money to be earned in $C$ in week $i$, and $L_f \in \{0, 1\}$ telling you where to be at the end of week $n$, where 0 corresponds to $W$ and 1 corresponds to $C$.

**Output**: Array $D$ such that $D[i] = 0$ if you should work week $i$ in $W$ and $D[i] = 1$ if you should work week $i$ in $C$.

(d) What is the runtime of your algorithm?

(e) Apply your algorithm to your example from part (b) and show that it outputs the correct schedule.

3. Suppose I would like to give you more flexibility on your exams, so I give you some large number (let's call it $M$) of problems, where the $i$th problem is worth $p_i$ points. The time I give you to take the exam is not sufficient to solve all of the problems, so

each student might solve a different subset of problems and might get different grades on each problem. I would like to give a good grade to a student who does sufficiently well on a sufficient number of questions, and give a worse grade to a student who just gets a few points correct on a lot of problems.

(a) How could I reduce this problem of determining grades to the knapsack problem?

(b) Is your reduction a polynomial reduction? Explain.

(c) Do you feel this is an equitable form of grading? Do you feel it is a form of grading that accurately captures the amount of learning a student accomplishes? Would you like to have professors use this grading scheme to determine your grade? Why or why not?