CS 473 ✧ Fall 2024

Homework 2

Due Tuesday, September 17, 2024 at 9pm Central Time

Full credit for any dynamic programming problem requires the following ingredients:

- A clear English specification of the underlying recursive function. In particular, this description should name the input parameters and describe how the function value depends on those parameters. This is the most important part of the problem!
- A recurrence or recursive backtracking algorithm that evaluates this recursive function.
- Iterative details: A description of an appropriate data structure (which is *never* a hash table), an evaluation order, and the resulting running time.

Complete iterative pseudocode is *not* required for full credit. If you provide iterative pseudocode, you do *not* need to separately provide a recurrence, a memoization structure, and an evaluation order, but you do still need an English specification and (as always) the running time.

---

1. For each of the following problems, the input consists of two arrays $P[1..k]$ (the "pattern") and $T[1..n]$ (the "text") where $k \le n$.

   (a) Describe and analyze an algorithm to determine whether P occurs as two ***disjoint*** subsequences of T, where "disjoint" means the two subsequences have no indices in common. For example, the string `PPAP` appears as two disjoint subsequences in the string `PENPINEAPPLEAPPLEPEN`, but the strings `PEEPLE` and `PIKOTARO` do not.

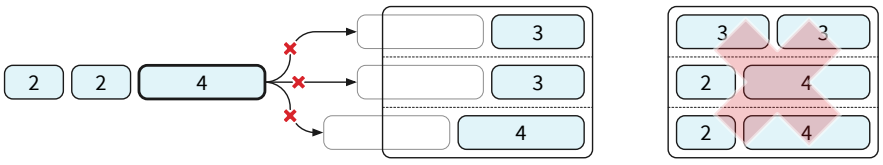
   (b) Describe and analyze an algorithm to compute the number of occurrences of P as a subsequence of T. For example, the string `PPAP` appears exactly 23 times as a subsequence of the string `PENPINEAPPLEAPPLEPEN`. If all characters in P and T are equal, your algorithm should return $\binom{n}{k}$. For purposes of analysis, assume that each arithmetic operation takes $O(1)$ time.

2. Suppose you are running a ferry across Lake Michigan.[1] The vehicle hold in your ferry is L meters long and three lanes wide. As each vehicle drives up to your ferry, you direct it to one of the three lanes; the vehicle then parks as far forward in that lane as possible. Vehicles must enter the ferry in the order they arrived; if the vehicle at the front of the queue doesn't fit into any of the lanes, then no more vehicles are allowed to board.

   Because your uncle runs the concession stand at the ferry terminal, you want to load as ***few*** vehicles onto your ferry as possible for each trip. But you don't want to be *obvious* about it, if the vehicle at the front of the queue fits anywhere, you must assign it to a lane where it fits. You can see the lengths of all vehicles in the queue on your security camera.

   Describe and analyze an algorithm to load the ferry. The input to your algorithm is the integer L and an array $len[1..n]$ containing the (integer) lengths of all vehicles in the queue. (You can assume that $1 \le len[i] \le L$ for all i.) Your output should be the ***smallest*** integer k such that you can put vehicles 1 through k onto the ferry, in such a way that vehicle $k+1$ does not fit. Express the running time of your algorithm as a function of both n (the number of vehicles) and L (the length of the ferry).

   For example, suppose $L = 6$, and the first six vehicles in the queue have lengths 3, 3, 4, 4, 2, and 2. Your algorithm should return the integer 3, because if you assign the first three vehicles to three different lanes, the fourth vehicle won't fit. (A different lane assignment gets all six vehicles on board, but that would rob your uncle of three customers.)



[1] Welcome aboard the Recursion Ferry! How we get across the water is none of your business!

3. Consider the following variant of the *coin-changing* problem. You are running a shop that still accepts cash—How quaint!. Whenever you give one of your customers change, you want to give them as few coins as possible. Your shop is located in a country whose coins have n different values; you have an unlimited supply of each type of coin.

   Fix a sorted array $Coin[1..n]$ of distinct positive integer coin values. For any integers k and T, let $Change(T,k) = \textsc{True}$ if some collection of exactly k coins (possibly including multiple coins with the same value) has total value T, and $\textsc{False}$ otherwise.

   For example, if $Coin = [1,5,10,25,50,100]$, then $Change(30,2) = Change(30,3) = Change(30,8) = \textsc{True}$, but $Change(30,1) = Change(30,7) = \textsc{False}$.

   (a) Describe a recurrence for $Change(T,k)$ in terms of $Change(\cdot, k/2)$, possibly invoking the latter function multiple times, with different first arguments. Assume k is a power of 2.

   (b) Describe an efficient algorithm to compute $Change(T,k)$, given the array $Coin[1..n]$, the target total T, and the allowed number of coins k as input. Analyze your algorithm as a function of the parameters n, T, and k. (You can safely assume that $n \le T$ and $k \le T$, and that k is a power of 2.)

   (c) **Extra credit:** Describe an efficient algorithm to compute the smallest number of coins with total value T. For example, with American coins, given $T = 30$, your algorithm should return the integer 2.[2]

   You may assume that there is at least one collection of coins with total value T. Ideally, your algorithm should have exactly the same big-Oh running time as your algorithm for part (b), where now k is the output value (which is *not* necessarily a power of 2).

[2]With with most real-world currency systems, including the American system, a simple greedy algorithm always yields the minimum number of coins for a given value T: If $T = 0$, return nothing; otherwise, use one coin with the largest value $c \le T$ and recursively find coins with value $T - c$. This greedy algorithm does ***not*** work for all coin systems, however—consider the coin values $[1,3,4]$ and the target value 6, for example—so you should assume that it does ***not*** work for yours.