CS 4150: Homework 3
Divide & conquer, Memoization & dynamic programming

Submission date: Friday, Oct 4, 2024 (11:59 PM)

This assignment has 4 questions, for a total of 40 points. Unless otherwise specified, complete and reasoned arguments will be expected for all answers.

Note. *When asked to describe and analyze an algorithm, you need to first write the pseudocode, provide a running time analysis by going over all the steps (writing recurrences if necessary), and provide a reasoning for why the algorithm is correct.* Skipping or having incorrect reasoning will lead to a partial credit, even if the pseudocode itself is OK.

Question 1: Selection Revisited .... **[8]**

The *Order Selection* algorithm we saw in class can be a powerful subroutine. Recall that `Select`(A, k) allows one to find the kth smallest element of an *unsorted* array A (with n elements) in time $O(n)$.

Given an (unsorted) array A with n elements, a *frequent element* is defined as some x that appears at least $\lceil n/4 \rceil$ times in the array. Describe an algorithm that runs in $O(n)$ time and finds all the frequent elements (if they exist). For convenience, you may assume that n is a multiple of 4. Along with pseudocode, **provide reasoning** showing that your algorithm returns the correct answer and has $O(n)$ running time.

[*Hint:* Suppose you find the kth smallest element of A for $k = \frac{n}{4}, \frac{n}{2}, \frac{3n}{4}, n$. Argue that all "potential" frequent elements must be one of the elements you found.]

Question 2: Faster Fibonacci .... **[11]**

In the previous Homework, we saw how to implement exponentiation, i.e., finding a^n, using $O(\log n)$ (multiplication) operations. Now suppose we want to compute powers of 2×2 matrices. Assume that arithmetic operations (addition & multiplication) between two integers takes constant time.

(a) [**4**] Suppose A is a 2×2 matrix, and let $n \geq 1$ be an integer. Describe an algorithm to compute A^n (the matrix A multiplied n times) using $O(\log n)$ operations. [Just the pseudocode plus a line or two of explanation suffices. You may refer to HW 2.]

(b) [**2**] Let M be the matrix below:

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

Let f_n denote the nth Fibonacci number ($f_1 = f_2 = 1$ and $f_n = f_{n-1} + f_{n-2}$ for all $n > 2$). Evaluate the expression: (It is M times the 2×1 vector with f_{n-1} and f_n as entries.)

$$M \cdot \begin{bmatrix} f_{n-1} \\ f_n \end{bmatrix}$$

(c) [**5**] Using parts (a) and (b), give an algorithm for computing f_n that uses only $O(\log n)$ arithmetic operations.

Question 3: Broken Space Bar . . . . **[13]**

You just received an email from a friend who had a broken space bar, so the message is just one long string with all the words concatenated. You want to find a way to split the string into valid words, but as a first step, you want to make sure your friend had no typos – you want to find out if it is *possible* to split the string into valid words.

Given a dictionary, i.e., a set of "valid words" given as a collection of strings $w_1, w_2, \ldots, w_N$, and an input string S, the goal is to split S into a combination of valid words. E.g., if the set of valid words is {`ate, bar, bard, cats, dog, man, rant`} and you are given the string `baraterant`, you need to output the split: `bar, ate, rant`.

Suppose we have access to a procedure called isValidWord(w) that takes a query word w and returns true/false depending on whether w is a valid word, and suppose all such look-ups take time L. We will also denote by $S[i : j]$ the substring of S starting at position i and ending in position j.

(a) **[4]** Consider the following simple procedure Parse(S): find the smallest i such that $S[0 : i]$ is a valid word, cut it off, and recurse on the remaining string (i.e., return Parse($S[i + 1, \text{len}(S) - 1]$)). Does this always produce a valid split? [*Hint:* consider the example dictionary above...]

(b) **[3]** Consider a more brute force approach that searches over multiple splits (not just removing the first valid word). Suppose m is the maximum word length in the dictionary.

Algorithm 1 Procedure `isValidString`(S)

If S is empty, return `True`
for $i = 0, 1, \ldots, m - 1$ **do**
  if isValidWord($S[0, i]$) && isValidString($S[i + 1, \text{len}(S) - 1]$) **then**
    return `True`
  end if
end for
return `False`

What do you expect will be the running time of the procedure `isValidString` (without memoization, as a function of the length of the string)? You can give an informal answer.

(c) **[6]** Suppose you want to do memoization (i.e., store the answers to all potential recursive calls). (i) How many distinct recursive calls are possible and why? (ii) Derive a bound on the running time after memoization.

Question 4: Counting Change . . . . **[8]**

Recall the CountChange procedure that we saw in class. In this problem, we are given coins of n denominations, $d_1, d_2, \ldots, d_n$ (in cents), and the goal is to find the number of ways of making change for N cents.

We denoted by CountChange(r, j) the number of ways of making change for r cents using coins of denominations $d_j, d_{j+1}, \ldots, d_n$. For this function, we wrote a recursive formulation:

$$\text{CountChange}(r, j) = \begin{cases} 1 \text{ if } j = n+1 \text{ and } r = 0, \\ 0 \text{ if } j = n+1 \text{ and } r \neq 0, \\ \sum_{i=0}^{\lfloor r/d_j \rfloor} \text{CountChange}(r - i \cdot d_j, j+1). \end{cases}$$

We observed that the total number of distinct recursive calls (when starting with $r = N$ and $j = 1$) is $O(Nn)$, which led to a memory requirement of $O(Nn)$ and a time complexity of $O(N^2 n)$. Suppose now that we want to improve the running time of this procedure. One simple idea is to observe that if we were given a j, and we wanted to compute the values of CountChange$(0, j)$, CountChange$(1, j)$, $\ldots$, CountChange(N, j), we could do this given the values CountChange$(0, j+1)$, CountChange$(1, j+1)$, $\ldots$, CountChange$(N, j+1)$.

Using this observation, show how one can compute CountChange$(N, 1)$ using a space complexity only $O(N)$.