Homework 03

CS 624, 2024 Spring

1. Problem 15.4-5 on p397, modified.

   > Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

   Do not write out the algorithm. Instead, do the following:

   (a) We will start with a related problem: Given a sequence $X[1 \mathrel{..} n]$, what is the longest increasing subsequence of $X[1 \mathrel{..} n]$ that ends with $X[n]$? I'll call this the LISE problem.

   State the *optimal substructure property* for the LISE problem. You are not required to prove it.

   (b) Define a recursive function $LISE(k)$ that returns the *length* of the longest monotonically increasing subsequence for $X[1 \mathrel{..} k]$ that ends in $X[k]$. Assume that the original sequence of numbers X is fixed/global.

   (c) Briefly explain how to solve the original problem by adapting the LISE-solving algorithm.

   (d) Briefly explain the running time of the algorithm, assuming memoization.

   Note that there are other ways of solving the original problem. You should think about that, but for the homework you are required to use the structure above.

2. Problem 15-2 on p405, modified.

   > A *palindrome* is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).
   >
   > Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`. What is the running time of your algorithm?

   Do not write out the algorithm. Instead, do the following:

   (a) State the *optimal substructure property* for the longest palindrome subsequence problem. You are not required to prove it.

   (b) Define a recursive function $LPS(??)$ which returns the *length* of the longest palindrome subsequence for a given subproblem. Assume the original string is fixed/global. You must decide what arguments identify each subproblem. Briefly explain the meaning of the arguments.

   (c) Briefly explain the running time of your algorithm, assuming memoization.

3. Problem 16.2-2 on p427, modified.

   > Give a dynamic-programming solution to the 0-1 knapsack problem that runs in $O(nW)$ time, where n is the number of items and W is the maximum weight of items that the thief can put in his knapsack.

   Do not write out the algorithm. Instead, do the following:

   (a) State the *optimal substructure property* for the 0-1 knapsack problem. You are not required to prove it.

(b) Define a recursive function $KS(??)$ which returns the maximum *total value* of items in the knapsack for a given subproblem. Briefly explain what parts of the original problem are fixed/global and what parts are subproblem-specific.

(c) Briefly explain the running time of your algorithm, assuming memoization.

4. The Fibonacci numbers are defined by the following recurrence:

$$F_0 = 0 \qquad F_1 = 1 \qquad F_{n+2} = F_{n+1} + F_n$$

(a) Write pseudocode for a *naive* recursive function that computes the n^{th} Fibonacci number.

(b) Write pseudocode for a *top-down memoized* version of the same function. Your main function should still take one integer argument and produce one integer result; it should use a memoized helper function.

(c) Write pseudocode for a *bottom-up memoized* version of the Fibonacci function. Your main function should still take one integer argument and produce one integer result.

If you are using LaTeX, the `verbatim` or `alltt` environment might be helpful for writing the pseudocode.