CMPS 2200 Introduction to Algorithms – Fall 15

10/7/15

6. Homework

Due **10/19/15** at the beginning of class

1. **DP in less space (6 points)**

   (a) (2 points) The bottom-up dynamic programming algorithm computing the n-th Fibonacci number $F(n)$ takes $O(n)$ time and uses $O(n)$ space. Show how to modify the algorithm to use only constant space. Give pseudo-code for your solution.

   (b) (4 points) Suppose we want to compute only the *length* of an LCS of two strings of length m and n. Describe how to alter the dynamic programming algorithm such that it only needs $O(\min(m, n))$ space. Give pseudo-code for your solution.
   (Hint: Try to first develop an algorithm that runs in either $O(m)$ or $O(n)$ space.)

2. **LCS traceback (3 points)**
   Give pseudocode that performs the traceback to construct an LCS from a filled dynamic programming table *without* using the "arrows", in $O(n+m)$ time. Justify shortly why your algorithm is correct.
   (Hint: You need to essentially "recompute" the information.)

3. **LCS memoization (4 points)**
   Compute the length of LCS("ABC'', "BAC'') using memoization. In which order do you fill the entries in the DP-table? Give the DP-table for this case and annotate each cell with a "time stamp" (i.e., with a number $1, 2, 3, ...$) when it was filled.

4. **Edit distance (8 points)**
   Let A and B be two strings of length m and n, respectively. The *edit distance* of A and B is the minimum number of *transformations* needed to transform A into B. Allowed transformations are: *insert* a character into A, *delete* a character from A, *replace* a character in A with another character.

   (a) (5 points) Develop a recurrence for the edit distance. This is similar to the LCS problem. Do not forget to state the base cases. You do not need to give a formal proof, but please justify the correctness shortly.

   (b) (3 points) Give pseudocode for a bottom-up dynamic programming algorithm that computes the edit distance for two strings of length m and n. What is its runtime?

FLIP OVER TO BACK PAGE ⟹

Practice Problems

(Not required for homework credit.)

(a) **Edit distance**

What is the edit distance for "BABCADB" and "ADCAB"? For visualization purposes, align both strings on top of each other such that spaces are inserted into the strings for insertions/deletions, and such that characters on top of each other either mismatch (= replacement operation), match (= no operation), or line up with a space (= insertion or deletion).

(b) **Binomial coefficient**

Given n and k with $n \geq k \geq 0$, we want to compute the binomial coefficient $\binom{n}{k}$.

i. Give pseudo-code for the bottom-up dynamic programming algorithm to compute $\binom{n}{k}$ using the recurrence

$$\begin{aligned} \binom{n}{k} &= \binom{n-1}{k-1} + \binom{n-1}{k}, \text{ for } n > k > 0 \\ \binom{n}{0} &= \binom{n}{n} = 1, \text{ for } n \geq 0 \end{aligned}$$

ii. What are the runtime and the space complexity of your algorithm, in terms of n and k?

iii. Now assume you use memoization to compute $\binom{4}{3}$ using the above recurrence. In which order do you fill the entries in the DP-table? Give the DP-table for this case and annotate each cell with a "time stamp" (i.e., with a number $1, 2, 3, ...$) when it was filled. *(Hint: Draw the recursion tree and fill the table in the order determined by the recursive evaluation.)*

(c) **Checkerboard**

Suppose that you are given an $n \times n$ checkerboard. A checker is allowed to move from its current square to (1) the square immediately above, (2) the square that is one up and one to the left, and (3) the square that is one up and one to the right, as long as it does not leave the checkerboard. You are also given a cost function c which specifies for every valid move from square (i_1, j_1) to square (i_2, j_2) a possibly negative amount of $c((i_1, j_1), (i_2, j_2))$ dollars.

Your task is to design a dynamic programming algorithm that finds a path from some position on the bottom edge to some position on the top edge of the checkerboard, such that the total dollar amount that is generated by the moves on the path is maximized. *(Hint: The first part of the sentence specifies base cases, the second the recursive case.)*

Define a recurrence relation for the total cost of a path to a given square:

$$total(i, j) = \text{ total cost of a path to square } (i, j),$$

for all i, j. Briefly justify the correctness. Then argue in words (no pseudo-code necessary) how the dynamic programming matrix would be filled and finally traced back to find an optimal path, and give the runtime of the resulting algorithm.