CMPS 2200 Introduction to Algorithms – Fall 17

10/18/17

6. Homework

Due **10/25/17** at the beginning of class

Remember, you are allowed to turn in homeworks in groups of two. One writeup, with two names.

1. **Master theorem (12 points)**
   Use the master theorem to solve the following recurrences. Justify your results. Assume that $T(1) = 1$.

   (a) $T(n) = 9T(\frac{n}{3}) + n^4$
   (b) $T(n) = 9T(\frac{n}{3}) + n^2 \log n$
   (c) $T(n) = T(\frac{n}{2}) + \sqrt{n}$
   (d) $T(n) = 125T(\frac{n}{5}) + n \log n$

2. **LCS in Less Space (5 points)**
   Suppose we want to compute only the *length* of an LCS of two strings of length m and n. Describe in words how to alter the dynamic programming algorithm such that it only needs $O(\min(m, n))$ space.
   (Hint: Try to first develop an algorithm that runs in either $O(m)$ or $O(n)$ space.)

3. **Longest Common Substring (12 points)**

   A *substring* is a contiguous subsequence. For example, $ACADA$ is a substring of $ABACADABRA$, while $CABRA$ is a subsequence but not a substring, and $DCAR$ is neither.

   Now, let $X = X[1..m]$ and $Y = Y[1..n]$ be two strings of length m and n, respectively. The task is to compute a longest common substring of X and Y.

   (a) (4 points) Develop a recurrence for the length of a longest common substring. 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) (4 points) Give pseudocode for a bottom-up dynamic programming algorithm that computes the length of a longest common substring for two strings of length m and n. What is its runtime?
   (c) (4 points) In words, describe how you can compute an actual longest common substring from the dynamic programming table.
   (Hint: Where is the maximum located?)

Flip over to back page $\Longrightarrow$

Practice Problems

(Not required for homework credit.)

1. **Edit distance**
   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) What is the edit distance for "BACADBA" and "ADCBAB"? 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) 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.

   (c) 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?

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

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

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

   (b) What are the runtime and the space complexity of your algorithm, in terms of n and k?