CS 5633 Analysis of Algorithms – Spring 04

3/24/04

# 8. Homework

Due **3/31/04** before class

1. **LCS reconstruction (3 points)**
   Can you reconstruct an LCS from the filled dynamic programming table *without* using the "arrows", in $O(n+m)$ time? Justify your answer.

2. **Saving space (5 points)**
   Suppose we only want to compute the *length* of an LCS of two strings of length $m$ and $n$. Show how to alter the dynamic programming algorithm such that it only needs $\min(m,n)+O(1)$ space. (Notice that it is *not* $O(\min(m,n))$, but plain $\min(m,n)$.)

3. **Saving space – again? (3 points)**
   Suppose we only want to compute the *minimum number* of scalar multiplications to multiply a chain of $n$ matrices. Can we save space in a similar manner to problem 2 ? If yes argue how we can save space, and how much. If not, argue why.

4. **Don't be greedy (3 points)**
   Consider the following *greedy* approach to solve the matrix chain multiplication problem for $A_1 \cdot \ldots \cdot A_n$ with the sequence of matrix dimensions $p_0, p_1, \ldots, p_n$: Let $k^*$ be that value of $k$ that minimizes $p_0 p_k, p_n$ for all $1 \le k < n$. Split the matrix chain at index $k^*$, i.e., parenthesize as follows: $(A_1 \cdot \ldots \cdot A_{k^*}) \cdot (A_{k^*+1} \cdot \ldots \cdot A_n)$. The two subproblems are being solved recursively in a similar way.

   Give an example matrix chain (i.e., a sequence of dimensions is enough), which shows that this greedy approach does not yield an optimal parenthesization.

5. **Binomial coefficient (8 points)**
   Given $n$ and $k$ with $n \ge k \ge 0$, we want to compute the binomial coefficient $\binom{n}{k}$. However, we are only allowed to use additions, and no multiplications.

   **a) (3 points)** Give a bottom-up dynamic programming algorithm to compute $\binom{n}{k}$ using the recurrence

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

   .

   **b) (1 point)** How fast does your algorithm run, expressed in $n$ *and* $k$?

   **c) (1 point)** What is the minimum amount of space you need?

   **d) (1 point)** What mathematical "entity" does your dynamic programming table store?

   **e) (2 point)** Now assume you use memoization to compute $\binom{4}{2}$ 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" when it was filled.

FLIP OVER TO BACK PAGE $\Longrightarrow$

6. **Change (8 points)**
Vending machines need to be able to give change, preferrably by breaking the owed amount into the minimum number of coins. Since vending machines exist all over the world, we don't want to be restricted to cents, nickels, dimes, and quarters, but we want to be able to perform this task for all kinds of different coin denominations.

Let $d_1, \ldots, d_k$ be the given coin denominations, assume $d_1$ always equals 1, and let $n$ be the amount to break into change. Assume you have an inifinite amount of coins for every denomination. The task is to find the minimum number of coins whose values sum up to $n$.

Example: Assume we have $n = 8$ cents, and we have 1-cent coins, 4-cent coins, and 5-cent coins, so $k = 3$ and $d_1 = 1, d_2 = 4, d_3 = 5$. The best way to make change for 8 cents is to use two 4-cent coins.

Devise a dynamic programming algorithm to solve the task. Proceed in the following steps:

a) Come up with a recurrence relation for the minimum *number* of coins needed.

b) Use dynamic programming to compute this number.

c) Finally, extract an actual optimal set of coins from the DP-table.

What is the runtime of your algorithm, expressed in $n$ and $k$?