CSE 441T/541T: Advanced Algorithms **2008**

Practice Problems on Dynamic Programming

Below are four practice problems on designing and proving the correctness of dynamic programming algorithms. For those of you who feel like you need us to guide you through some additional problems (that you first try to solve on your own), these problems will serve that purpose.

The solutions are available on the course web page (under homeworks). You can use these solutions as a guide as to how you should write-up your solutions. These problems will be MUCH more valuable to you if you first solve them and then check the solutions. Hints to get you going are also available on the web page if you want a hint without seeing the whole solution. If you need further guidance, let us know.

Practice Problems

1. Suppose we want to make change for n cents, using the least number of coins of denominations $1, 10$, and 25 cents.

   Describe an $O(n)$ dynamic programming algorithm to find an optimal solution. (There is also an easy $O(1)$ algorithm but the idea here is to illustrate dynamic programming.)

2. Here we look at a problem from computational biology. You can think of a DNA sequence as sequence of the characters "a","c","g","t". Suppose you are given DNA sequences D_1 of n_1 characters and DNA sequence D_2 of n_2 characters. You might want to know if these sequences appear to be from the same object. However, in obtaining the sequences, laboratory errors could cause reversed, repeated or missing characters. This leads to the following sequence alignment problem.

   An alignment is defined by inserting any number of spaces in D_1 and D_2 so that the resulting strings D_1' and D_2' both have the same length (with the spaces included as part of the sequence). Each character of D_1' (including each space as a character) has a corresponding character (matching or non-matching) in the same position in D_2'. For a particular alignment A we say $cost(A)$ is the number of mismatches (where you can think of a space as just another character and hence a space matches a space but does not match one of the other 4 characters).

   To be sure this problem is clear suppose that D_1 is `ctatg` and D_2 is `ttaagc`. One possible alignment is given by:

   ```
   ct at g
    tta agc
   ```

   In the above both D_1' and D_2' have length 8. The cost is 5. (There are mismatches in position 1, 3, 5, 6 and 8).

   Give the most efficient algorithm you can (analyzed as a function of n_1 and n_2) to compute the alignment of minimum cost.

3. You are traveling by a canoe down a river and there are n trading posts along the way. Before starting your journey, you are given for each $1 \leq i < j \leq n$, the fee $f_{i,j}$ for renting a canoe from post i to post j. These fees are arbitrary. For example it is possible that $f_{1,3} = 10$ and $f_{1,4} = 5$. You begin at trading post 1 and must end at trading post n (using rented canoes). Your goal is to minimize the rental cost. Give the most efficient algorithm you can for this problem. Be sure to prove that your algorithm yields an optimal solution and analyze the time complexity.

4. For bit strings $X = x_1 \ldots x_m$, $Y = y_1 \ldots y_n$ and $Z = z_1 \ldots z_{m+n}$, we say that Z is an *interleaving* of X and Y if it can be obtained by interleaving the bits in X and Y in a way that maintains the left-to-right order of the bits in X and Y. For example if $X = 101$ and $Y = 01$ then $x_1 x_2 y_1 x_3 y_2 = 10011$ is an interleaving of X and Y, whereas 11010 is not. Give the most efficient algorithm you can to determine if Z is an interleaving of X and Y. Prove your algorithm is correct and analyze its time complexity as a function $m = |X|$ and $n = |Y|$.

CSE 441T/541T: Advanced Algorithms **2008**

HW 2 Practice Problems Solutions

Here are the solutions for the practice problems on dynamic programming (homework 2). **However**, reading these is far less useful than solving these yourself, writing up your solution and then either comparing your solution to these or showing your solution to one of us at our office hours to look over. Also, if you just need help in figuring out the general subproblem form, you can find hints on the course web page.

1. Below is a dynamic programming solution for this problem to illustrate how it can be used. There is a very straightforward $O(1)$ time solution. It can be shown that if $n >= 50$ then any solution will include a set of coins that adds to exactly 50 cents. Hence it can be shown that an optimal solution uses $2 \cdot \lfloor n/50 \rfloor$ quarters along with an optimal solution for making $n/50 - \lfloor n/50 \rfloor$ cents which can be looked up in a table of size 50.

   Here's the dynamic programming solution for this problem. (It does not use the fact that an optimal solution can be proven to use $2 \cdot \lfloor n/50 \rfloor$ quarters and hence is not as efficient.) The general subproblem will be to make change for i cents for $1 \leq i \leq n$. Let $c[i]$ denote the fewest coins needed to make i cents. Then we can define $c[i]$ recursively by:

$$c[i] = \begin{cases} \text{use } i \text{ pennies} & \text{if } 1 \leq i < 9 \\ c[i-10]+1 & \text{if } 10 \leq i < 24 \\ \min(c[i-10]+1, c[i-25]+1) & \text{if } i \geq 25 \end{cases}$$

   Note that $c[n]$ is the optimal number of coins needed for the original problem.

   Clearly when $i < 10$ the optimal solution can use only pennies (since that is the only coin available). Otherwise, the optimal solution either uses a dime or a quarter and both of these are considered. Finally, the optimal substructure property clearly holds since the final solution just adds one coin to the subproblem solution. There are n subproblems each of which takes $O(1)$ time to solve and hence the overall time complexity is $O(n)$.

2. Recall that D_1 is a DNA sequence with n_1 characters and D_1 is a DNA sequence with n_2 characters. The general form of the subproblem we solve will be: Find the best alignment for the first i characters of D_1 and the first j characters of D_2 for $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$. Let $D(i)$ be the ith character in string D. Let $c[i,j]$ be the cost of an optimal alignment for $D_1(1), \ldots, D_1(i)$ and $D_2(1), \ldots, D_2(j)$. We can define $c[i,j]$ recursively as shown (where $c[n_1, n_2]$ gives the optimal cost to the original problem):

$$c[i,j] = \begin{cases} i & \text{if } j = 0 \\ j & \text{if } i = 0 \\ c[i-1, j-1] & \text{if } D_1(i) = D_2(j) \\ \min\{c[i-1,j-1], c[i-1,j], c[i,j-1]\} + 1 & \text{otherwise} \end{cases}$$

We now argue this recursive definition is correct. You can form D_1' and D_2' (and hence the alignment) for the subproblem from the right to left as follows. In an optimal alignment either the last character of D_1' is a space or it is the last character (character i) of D_1 and the last character of D_2' is a space or it is the last character (character j) of D_2. If $D_1(i) = D_2(j)$ then clearly it is best to align them (so add a space to neither). However, if $D_1(i) \neq D_2(j)$ then a space could be added to neither or just one. In all three cases one mismatch is caused by the last characters. Notice that there would never be any benefit in ending both D_1 and D_2 with a space. Hence the above recursive definition considers all possible cases that the optimal alignment could have. Since the solution to the original problem is either the value of the subproblem solution (if $D_1(i) = D_2(j)$) or otherwise one plus the subproblem solution, the optimal substructure property clearly holds. Thus the solution output is correct.

For the time complexity it is clearly $O(n_1 \cdot n_2)$ since there are $n_1 \cdot n_2$ subproblems each of which is solved in constant time. Finally, the $c[i, j]$ matrix can be computed in row major order and just as in the LCS problem a second matrix that contains which of the above 4 cases was applied can also be stored and then used to construct an optimal alignment.

3. Let $m[i]$ be the rental cost for the best solution to go from post i to post n for $1 \leq i \leq n$. The final answer is in $m[1]$. We can recursively, define $m[i]$ as follows:

$$m[i] = \begin{cases} 0 & \text{if } i = n \\ \min\limits_{i<j\leq n} \ (f_{i,j} + m[j]) & \text{otherwise} \end{cases}$$

We now prove this is correct. The canoe must be rented starting at post i (the starting location) and then returned next at a station among $i+1, \ldots, n$. In the recurrence we try all possibilities (with j being the station where the canoe is next returned). Furthermore, since $f_{i,j}$ is independent from how the subproblem of going from post $j, \ldots, n$ is solved, we have the optimal substructure property.

For the time complexity there are n subproblems to be solved each of which takes $O(n)$ time. These subproblems can be computed in the order $m[n], m[n-1], \ldots, m[1]$. Hence the overall time complexity is $O(n^2)$.

NOTE: One (of several) alternate general subproblem form that also leads to an $O(n^2)$ algorithm is to find the best solution to get from post i to post n where the canoe cannot be exchanged until post j is reached (for $1 \leq i < j \leq n$).

4. The general form of the subproblem we solve will be: determine if $z_1, \ldots, z_{i+j}$ is an interleaving of $x_1, \ldots, x_i$ and $y_1, \ldots y_j$ for $0 \leq i \leq m$ and $0 \leq j \leq n$. Let $c[i, j]$ be true if and only if $z_1 \ldots, z_{i+j}$ is an interleaving of $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$. We use the convention that if $i = 0$ then $x_i = \lambda$ (the empty string) and if $j = 0$ then $y_j = \lambda$. The subproblem $c[i, j]$ can be recursively defined as shown (where $c[m, n]$ gives the answer

to the original problem):

$$c[i,j] = \begin{cases} \text{true} & \text{if } i = j = 0 \\ \text{false} & \text{if } x_i \neq z_{i+j} \text{ and } y_j \neq z_{i+j} \\ c[i-1,j] & \text{if } x_i = z_{i+j} \text{ and } y_j \neq z_{i+j} \\ c[i,j-1] & \text{if } x_i \neq z_{i+j} \text{ and } y_j = z_{i+j} \\ c[i-1,j] \vee c[i,j-1] & \text{if } x_i = y_j = z_{i+j} \end{cases}$$

We now argue this recursive definition is correct. First the case where $i = j = 0$ is when both X and Y are empty and then by definition Z (which is also empty) is a valid interleaving of X and Y. If $x_i \neq z_{i+j}$ and $y_j = z_{i+j}$ then there could only be a valid interleaving in which x_i appears last in the interleaving, and hence $c[i,j]$ is true exactly when $z_1, \ldots, z_{i+j-1}$ is a valid interleaving of $x_1, \ldots, x_{i-1}$ and $y_1, \ldots y_j$ which is given by $c[i-1,j]$. Similarly, when $x_i \neq z_{i+j}$ and $y_j = z_{i+j}$ then $c[i,j] = c[i-1,j]$. Finally, consider when $x_i = y_j = z_{i+j}$. In this case the interleaving (if it exists) must either end with x_i (in which case $c[i-1,j]$ is true) or must end with y_i (in which case $c[i,j-1]$ is true). Thus returning $c[i-1,j] \vee c[i,j-1]$ gives the correct answer. Finally, since in all cases the value of $c[i,j]$ comes directly from the answer to one of the subproblems, we have the optimal substructure property.

The time complexity is clearly $O(nm)$ since there are $n \cdot m$ subproblems each of which is solved in constant time. Finally, the $c[i,j]$ matrix can be computed in row major order.