Dynamic Programming Practice Problems with Solutions

Question 1: Matrix Chain Multiplication (Book page # 370)

Question 2:

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.

Solution:

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 $1 \leq i < j \leq n$).

Question 3:

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 $n = |Y|$.

Solution:

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.