Design and Analysis of Algorithms

Practice-sheet 5 : **Dynamic Programming**

1. **Weighted job scheduling problem**

   There are n jobs : $\langle J_1, \ldots, J_n \rangle$ and a single server. Job J_i has a start time s_i and finish time f_i. The server can execute only one job at a time. Unfortunately, there is overlap between the time spans of various jobs, and so we can not execute all the jobs on the server. Each job has also a profit associated with it which is obtained if that job is executed on the server. Let p_i denote the profit associated with job J_i. Give a counter-example to show that the greedy strategy "select the earliest finish time" won't work here. Design a polynomial time algorithm to find a subset of jobs that can be executed on the server such that the profit obtained is maximum.

2. **Monotonically increasing subsequence**

   Given a sequence $A = a_1, \ldots, a_n$, a subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is said to be monotonically increasing if $a_{i_j} < a_{i_{j+1}}$ for all $1 \leq j < k$. Design an $O(n^2)$ time algorithm to compute the longest monotonically increasing subsequence of sequence A.

3. **Bellman Ford algorithm**

   Let $G = (V, E)$ be a directed graph on n vertices and m edges where each edge has a weight which is a real number. Show that there exists an order among the vertices such that if we process the vertices according to that order in the inner For loop of the Bellman-ford algorithm, then just after one iteration, D[v] will store the distance from s to v.

4. **Box stacking**

   Box Stacking. You are given a set of n types of rectangular 3-D boxes, where the ith box has height h(i), width w(i) and depth d(i) (all real numbers). You want to create a stack of boxes which is as tall as possible, but you can only stack a box on top of another box if the dimensions of the 2-D base of the lower box are each strictly larger than those of the 2-D base of the higher box. Of course, you can rotate a box so that any side functions as its base. It is also allowable to use multiple instances of the same type of box.

5. **Bridges across a river**

   Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates $a(1)...a(n)$ and n cities on the northern bank with x-coordinates $b(1)...b(n)$. Note that there is no order among $a(i)$'s. Similarly, you can't assume any order among $b(i)$'s. You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. When connecting cities, you can only connect city i on the northern bank to city i on the southern bank. Design a polynomial time algorithm to compute the maximum number of non-crossing bridges that can be built.

6. **Edit Distance**

   Given two text strings A of length n and B of length m, you want to transform A into B with a minimum number of operations of the following types: delete a character from A, insert a character into A, or change some character in A into a new character. The minimal number of such operations required to transform A into B is called the edit distance between A and B. Design a polynomial time algorithm to compute edit distance between A and B.

7. **Matrix chain multiplication**

   There is a sequence of matrices $M_1, M_2, \ldots, M_n$ storing numbers. For each $1 < i \leq n$, the number of rows of M_i is identical to the number of columns of M_{i-1}. So the product $M_1 \times M_2 \times \ldots M_n$ is well defined. We also know that matrix multiplication is associative. That is, $(M_1 \times M_2) \times M_3 = M_1 \times (M_2 \times M_3)$. However, the number of arithmetic operations required may vary in these two possible ways. For example, let M_1 be 10×100, M_2 be 100×5, and M_3 be 5×50.

   - If we multiply according to $((M_1 \times M_2) \times M_3)$, we perform $10 \cdot 100 \cdot 5 = 5000$ multiplication operations to compute the 10×5 matrix $(M_1 \times M_2)$ and then $10 \cdot 5 \cdot 50$ multiplication operations to multiply this matrix with M_3. So a total of 7500 multiplication operations are carried out.
   - If instead we multiply according to $(M_1 \times (M_2 \times M_3))$, we perform $100 \cdot 5 \cdot 50 = 25000$ scalar multiplications to compute 100×50 matrix $(M_2 \times M_3)$ and then another $10 \cdot 100 \cdot 50 = 50000$ multiplication operation to multiply M_1 by this matrix. Hence a total of 75000 multiplication operations are carried out.

   Our aim is to compute $M_1 \times M_2 \times \cdots \times M_n$ using least number of multiplication operations. Design an algorithm based on dynamic programming to solve this problem in polynomial time. Here is some additional explanation which should help you understand the problem better.

   The order in which we compute $M_1 \times M_2 \times \cdots \times M_n$ is defined by parenthesizing the expression of n matrices. For example, if $n = 4$, then there are the following 4 ways to compute the product:

   - $(((M_1 \times M_2) \times M_3) \times M_4)$
   - $((M_1 \times M_2) \times (M_3 \times M_4))$
   - $(M_1 \times ((M_2 \times M_3) \times M_4))$
   - $(M_1 \times (M_2 \times (M_3 \times M_4)))$

   The number of ways to parenthesize the expression is exponential (it is catalan number). Of course, we can not afford to try out each possible way to multiply the n matrices.
   Hint: Consider that parenthesized expression that corresponds to the least number of multiplication operations. If you multiply the matrices according to this parenthesization, the number of matrices will reduce by 1 each time. Finally we will be left with only 2 matrices. How do these last two matrices which are multiplied look like ? What can you infer from it ? Use it to come up with a recursive formulation.