Dynammic Programming

Main Focus: Knapsack

Problem Statement for Chain Matrix Multiplication:

Given a series of matrices: $A_1 \times A_2 \times A_3 \times \ldots \times A_n$ with their dimensions, what is the best way to parenthesize them so that it produces the minimum number of total multiplications.

Solution:

Input: Sequence of matrices $A_1 \times A_2 \times A_3 \times \ldots \times A_n$ where A_i is a $P_{i-1} \times P_i$. The array P contains the dimensions.

For this problem, there are many possibilities for multiplication. This is because multiplication is associative. It does not matter how we paranthesize the product, the result will be the same. For example, consider matrices A, B, C, D, the possibilities are:
$(ABC)D = (AB)(CD) = A(BCD) = A(BC)D = \ldots$

Multiplying the $(p \times q)$ matrix with the $(q \times r)$ matrix requires pqr multiplications. Each of the above possible multiplications produces a different number of products. To select the best one, we can go through possible parenthesization (brute force), but this requires $O(2^n)$ time and is very slow.

GREEDY SOLUTION: In this solution, we would always do the cheapest multiplication first. Example of this backfiring on us is:

Consider $A_1 \times A_2 \times A_3$ with dimensions $3 \times 100, 100 \times 2 and 2 \times 2$. Based on greedy approach, we get:
$A_1 \times (A_2 \times A_3) = 1000$ multiplications. But, the optimal solution to this problem is:
$(A_1 \times A_2) \times A_3 = 612$ multiplications. Thus, we CANNOT use greedy for solving this problem.

DP SOLUTION: Assume that, $M[i, j]$ represents the least number of multiplications needed to multiply $A_i, \ldots, A_j$.

$$M[i,j] = \begin{cases} 0, & if \quad i = j \\ Min\{M[i,k] + M[k+1,j] + P_{i-1}P_kP_j\}, & if \quad i < j \end{cases}$$

The above recursive problem requires us to find point k such that it produces the minimum number os multiplications. After computing all the possible values for k, we have to select the k values which gives minimum value.

Coded Algorithm:

We use one more table $S[i, j]$ to reconstruct the optimal parenthesizations. We compute the $M[i, j]$ and $S[i, j]$ in a bottom-up fashion.

> /*P is the size of the matrices. Matrix i has the dimension $P[i-1] \times P[i]$.
>
> $M[i, j]$ is the best cost of multiplying the matrices i through j

> $S[i, j]$ saves the multiplication point and we use this for back tracing. */

```
void MatrixChainOrder(int P[], int length){
    int n = length-1, M[n][n], S[n][n];
    for(int i = 1; i<=n; i++)
        M[i][i] = 0;
    //Filling matrix by diagonals
    for(int l=2; l<=n; l++){ //l is chain length
        for(int i=1; i <= n-l+1; i++){
            int j = i+l-1;
            M[i][j] = MAX_VALUE;
            //Try all possible division points i...k and k...j
            for(int k=i; k<=j-1; k++){
                int thisCost = M[i][k] + M[k+1][j] + P[i-1]*P[k]*P[j];
                if(thisCost < M[i][j]){
                    M[i][j] = thisCost;
                    S[i][j] = k;
                }
            }
        }
    }
}
```

Analysis:

1. Time Complexity: $O(n^3)$
2. Space Complexity: $O(n^2)$

Problem Statement for KnapSack:

Duplicate Items Permitted (Integer Knapsack Problem):

Given n types of items, where the i^{th} item type has integer size s_i and a value v_i. We need to fill a knapsack of total capacity C with items of maximum value. We can add multiple items of same type to the knapsack.

NOTE: It is not compulsory to fill the knapsack completely. If we fill the knapsack of size C completely and we get a value V and without filling the knapsack completely (let us take $C-1$), with value U and if $V < U$ then we consider the second one.

Let $M(j)$ denote the maximum value we can pack into a j size knapsack. We can express $M(j)$ recursively in terms of solutions to subproblems as follows:

$$M(j) = \{maxM(j-1), max_{i=1ton}(M(j-s_i)) + v_i, if \quad j \geq 1$$
$$0, if \quad j \leq 0\}$$

Based on the solution above, the decision depends on whether we select a particular i^{th} item or not for a knapsack of size j.

1. If we select i^{th} item, then we add its value v_i to the optimal solution and decrease the size of the knapsack to be solved to $j - s_i$.
2. If we do not select the item then check whether we can get a better solution for the knapsack of size $j-1$.

The value of $M(C)$ will contain the value of the optimal solution. We can find the list of items in the optimal solution by maintaining and following "back pointers".

Time Complexity: $O(nC)$

Space Complexity: $O(C)$

Duplicate Items Restricted (0-1 Knapsack Problem):
We do not have infinite number of items. Each item is allowed to be used 0 ot 1 time ONLY.

Same constraints as above with the minor change noted. We try to find the recursive solution to this using DP. Let $M(i, j)$ represent the optimal value we can get for filling up a knapsack of size j with items $1 \ldots i$. This gives the recursive formula as:
$M(i, j) = Max\{M(i-1, j), M(i-1, j-s_i) + v_i\}$

Time Complexity: $O(nC)$

Space Complexity: $O(nC)$

Now, we intutively see the matrix which helps us create the optimal solution. Lets assume the size of the matrix to be M.

Since, i takes values from $1 \ldots n$ and j takes value from $1 \ldots C$, there are a total of nC subproblems. Now let us see what the above formula says:

1. $M(i-1, j)$: Indicates the case of not selecting the i^{th} item. In this case, since we are not adding any size to the knapsack we have to use the same knapsack size for subproblems but excluding the i^{th} item. The remaining items are $i-1$.
2. $M(i-1, j-s_i) + v_i$ indicates the case where we select the i^{th} item. If we add that item, then we have to reduce the subproblem knapsack size to $j - s_i$ and at the same time we need to add the value v_i to optimal solution. The remaining items are $i-1$.

After finding all the $M(i, j)$ values, the optimal objective value can be obtained as:
$Max_j\{M(n, j)\}$.
This is because we do not know what amount of capacity gives the best solution.
