<h1 style="text-align:center">Assignment 3: Dynamic Programming</h1>

1. Determine the optimal order and cost for evaluating the product of matrices
   A1×A2×A3×A4, where:

   A1 has dimensions 5×2,

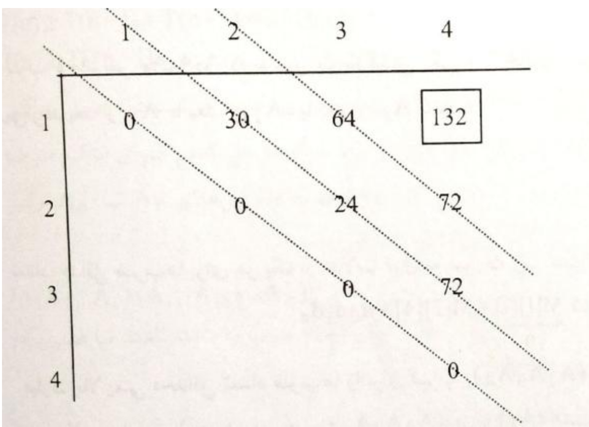
   A2 has dimensions 2×3,

   A3 has dimensions 3×4,

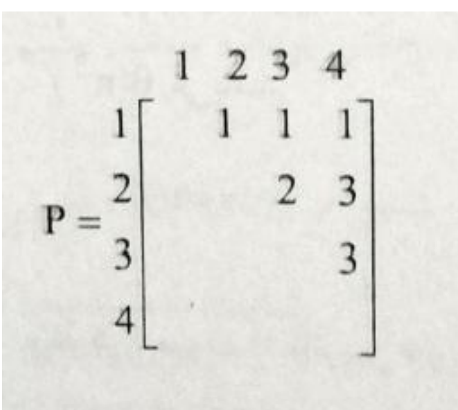
   A4 has dimensions 4×6.

   Construct and display the Cost table (used for optimization) and the K table (used for storing the optimal multiplication order). Show all calculations and steps used in the algorithm. (30 points)

   Answer:



	1	2	3	4
1	0	30	64	132
2		0	24	72
3			0	72
4				0



$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} & 1 & 1 & 1 \\ & & 2 & 3 \\ & & & 3 \\ & & & \end{bmatrix} \end{array}$$

A1((A2 A3) A4)

Activity 3

Cost (c)

$i \downarrow$ \ $j \rightarrow$	1	2	3	4
1	0	30	64	132
2		0	24	72
3			0	72
4				0

K value:

	1	2	3	4
1		1	1	1
2			2	3
3				3
4				

$d_0 = 5,\ d_1 = 2,\ d_2 = 3,\ d_3 = 4,\ d_4 = 6$

$c[1,1] = 0$

$$c[1,2] = \min_{1 \leq k < 2} \left\{ k=1:\ \underbrace{c[1,1]}_{0} + \underbrace{c[2,2]}_{0} + \underbrace{d_0 \times d_1 \times d_2}_{5 \times 2 \times 3} \right\} = 30$$

$$c[2,3] = \min_{2 \leq k < 3} \left\{ k=2:\ \underbrace{c[2,2]}_{0} + \underbrace{c[3,3]}_{0} + \underbrace{d_1 \times d_2 \times d_3}_{2 \times 3 \times 4} \right\} = 24$$

$$c[3,4] = \min_{3 \leq k < 4} \left\{ k=3:\ \underbrace{c[3,3]}_{0} + \underbrace{c[4,4]}_{0} + \underbrace{d_2 \times d_3 \times d_4}_{3 \times 4 \times 6} \right\} = 72$$

$$c[1,3] = \min_{1 \leq k < 3} \begin{cases} k=1:\ \underbrace{c[1,1]}_{0} + \overbrace{c[2,3]}^{24} + \underbrace{d_0 \times d_1 \times d_3}_{5 \times 2 \times 4} = 64 \\ k=2:\ \underbrace{c[1,2]}_{30} + \underbrace{c[3,3]}_{0} + \underbrace{d_0 \times d_2 \times d_3}_{5 \times 3 \times 4} = 90 \end{cases}$$

$$c[2,4] = \min_{2 \le k < 4} \begin{cases} k=2: & \underbrace{c[2,2]}_{0} + \underbrace{c[3,4]}_{72} + \underbrace{d_1 \times d_2 \times d_4}_{2 \times 3 \times 6} = 106 \\ k=3: & \underbrace{c[2,3]}_{24} + \underbrace{c[4,4]}_{0} + \underbrace{d_1 \times d_3 \times d_4}_{2 \times 4 \times 6} = \boxed{72} \end{cases}$$

$$c[1,4] = \min_{1 \le k < 4} \begin{cases} k=1: & \underbrace{c[1,1]}_{0} + \overbrace{c[2,4]}^{72} + \underbrace{d_0 \times d_1 \times d_4}_{5 \times 2 \times 6} = \boxed{132} \\ k=2: & \overbrace{c[1,2]}^{30} + \overbrace{c[3,4]}^{72} + \underbrace{d_0 \times d_2 \times d_4}_{5 \times 3 \times 6} = 162 \\ k=3: & \underbrace{c[1,3]}_{64} + \underbrace{c[4,4]}_{0} + \underbrace{d_0 \times d_3 \times d_4}_{5 \times 4 \times 6} = 184 \end{cases}$$

$$A_1 \cdot ((A_2 \cdot A_3) \cdot A_4)$$

2. As a skilled house robber, you aim to steal from houses along a street, each containing a certain amount of money. However, robbing two adjacent houses will trigger a security alarm.
   Given an array where each element represents the money stored in a house, formulate the dynamic programming recurrence relation to determine the maximum amount you can steal without setting off the alarm and explain your answer. (50 points)
   Answer:

Dynamic Programming Recurrence Relation

Define the State:

Let **dp[i]** represent the **maximum money** that can be stolen from the first `i` houses.

Base Cases:

1. **If there are no houses** (`n = 0`), the maximum money stolen is `0`:

$$dp[0] = 0$$

2. **If there is only one house** (`n = 1`), the maximum money stolen is the money in that house:

$$dp[1] = nums[0]$$

Recurrence Relation:

For each house **i**, you have **two choices**:

- **Skip the current house** → The maximum money is the same as `dp[i-1]`.
- **Rob the current house** → The maximum money is `nums[i-1] + dp[i-2]` (since you must skip the adjacent house).

Thus, the recurrence relation is:

$$dp[i] = \max(dp[i-1], nums[i-1] + dp[i-2])$$

where:

- `dp[i-1]` → Skipping the current house.
- `nums[i-1] + dp[i-2]` → Robbing the current house and adding the best amount from `i-2`.

Example Walkthrough

Example Input:

```plaintext
nums = [2, 7, 9, 3, 1]
```

Step-by-Step Computation (Using DP Table)

House	Money (`nums[i]`)	Rob (`nums[i] + dp[i-2]`)	Skip (`dp[i-1]`)	`dp[i]` (Max)
1	2	2	0	**2**
2	7	7	2	**7**
3	9	9 + 2 = 11	7	**11**
4	3	3 + 7 = 10	11	**11**
5	1	1 + 11 = 12	11	**12**

Final Output:

```plaintext
Maximum money stolen = 12
```