Name	Surti Keyur
Roll No.	25MCE026
Branch	M.Tech (CSE)

Experiment 8

Aim: To implement the Matrix Chain Multiplication (MCM) algorithm using Dynamic Programming, print the DP cost table, and calculate the minimum number of scalar multiplications required to multiply a sequence of matrices.

Problem Statement

Given a chain of matrices A1, A2, A3, ..., An where matrix Ai has dimensions p[i−1] × p[i], determine the most efficient order to multiply the matrices so that the total number of scalar multiplications is minimized.

Theory

Matrix multiplication is associative; however, different parenthesizations yield different multiplication costs. Dynamic Programming is used because the problem contains optimal substructure and overlapping subproblems.

Algorithm

1. Read number of matrices.

2. Store matrix dimensions in p[].

3. Create DP table dp[n+1][n+1].

4. Initialize dp[i][i] = 0.

5. For chain length L = 2 to n:

    For each valid i:

        Compute j = i + L − 1

        For each k from i to j−1:

            Compute cost = dp[i][k] + dp[k+1][j] + p[i−1] × p[k] × p[j]

Choose minimum.

6. Print DP table.

7. Final result = dp[1][n].

Source Code

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <climits>
using namespace std;

void matrixChainOrder(const vector<int>& p) {
    int n = p.size() - 1;
    vector<vector<int>> dp(n + 1, vector<int>(n + 1, 0));

    for (int L = 2; L <= n; L++) {
        for (int i = 1; i <= n - L + 1; i++) {
            int j = i + L - 1;
            dp[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {
                int cost = dp[i][k] + dp[k + 1][j] + p[i - 1] * p[k] * p[j];
                dp[i][j] = min(dp[i][j], cost);
            }
        }
    }

    // Print DP Table in matrix-style format
    cout << "\nMatrix Chain Multiplication DP Table:\n";
    cout << "---------------------------------------\n\n";

    // Print column headers
    cout << setw(6) << " ";
    for (int j = 1; j <= n; j++)
        cout << setw(6) << j;
    cout << "\n";

    // Print rows
    for (int i = 1; i <=n; i++) {
        cout << setw(3) << i << " "; // row index
        for (int j = 1; j <= n; j++) {
            if (i > j)
                cout << setw(6) << "-";
            else
                cout << setw(6) << dp[i][j];
        }
        cout << "\n";
    }
```

```
    cout << "\nMinimum number of multiplications is: " << dp[1][n] << endl;
}

int main() {
    // Example: 6 matrices
    // Dimensions: A1(30x35), A2(35x15), A3(15x5), A4(5x10), A5(10x20), A6(20x25)
    vector<int> p = {30, 35, 15, 5, 10, 20, 25};

    matrixChainOrder(p);

    return 0;
}
```

Output

```
PS E:\DSA\Experiments\EXP_8_MCM\output> & .\'MCM.exe'

 Matrix Chain Multiplication DP Table:
 --------------------------------------

           1     2     3     4     5     6
   1       0 15750  7875  9375 11875 15125
   2       -     0  2625  4375  7125 10500
   3       -     -     0   750  2500  5375
   4       -     -     -     0  1000  3500
   5       -     -     -     -     0  5000
   6       -     -     -     -     -     0

 Minimum number of multiplications is: 15125
PS E:\DSA\Experiments\EXP_8_MCM\output>
```

The minimum number of multiplications for the example input is 15125.

Time & Space Complexity

Time Complexity: $O(n^3)$

Space Complexity: $O(n^2)$

Conclusion

Matrix Chain Multiplication demonstrates how Dynamic Programming reduces computation by dividing the problem into smaller overlapping subproblems and combining their solutions efficiently.