K. J. Somaiya College of Engineering, Mumbai-77
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: C2	**Roll No.:16010122257**
Experiment No.__6____	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Implementation Matrix Chain Multiplication of Dynamic Programming

Objective: To learn Matrix chain multiplication using Dynamic Programming Approach

CO to be achieved:

CO 2 Describe various algorithm design strategies to solve different problems and analyse Complexity.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **http://www.lsi.upc.edu/~mjserna/docencia/algofib/P07/dynprog.pdf**
4. **http://www.geeksforgeeks.org/travelling-salesman-problem-set-1/**
5. **http://www.mafy.lut.fi/study/DiscreteOpt/tspdp.pdf**
6. **https://class.coursera.org/algo2-2012-001/lecture/181**
7. **http://www.quora.com/Algorithms/How-do-I-solve-the-travelling-salesman-problem-using-Dynamic-programming**
8. **www.cse.hcmut.edu.vn/~dtanh/download/Appendix_B_2.ppt**
9. **www.ms.unimelb.edu.au/~s620261/powerpoint/chapter9_4.ppt**

Pre Lab/ Prior Concepts:

Data structures, Concepts of algorithm analysis

Historical Profile:

Dynamic Programming (DP) is used heavily in optimization problems (finding the maximum and the minimum of something). Applications range from financial models and operation research to biology and basic algorithm research. So the good news is that understanding DP is profitable. However, the bad news is that DP is not an algorithm or a data structure that you can memorize. It is a powerful algorithmic design technique.

New Concepts to be learned:

Application of algorithmic design strategy to any problem, dynamic Programming method of problem solving Vs other methods of problem solving, optimality of the solution, Optimal Binary Search Tree Problems and their applications

Theory:

Problem definition:

Given a sequence of N matrices, the matrix chain multiplication problem is to find the most efficient way to multiply these matrices by minimizing the number of computations involved during multiplications.

Optimal Substructure: parameterization/ select the subgroup of matrices that will result in least number of computations.

For multiplication of matrix series Ai to Aj, choose Ak such that multiplication of matrices through Ai..k and Ak+1…j will incur least number of computations for any k such that i<=k<j.

Recursive Formula:

$$m[i,j] = \begin{cases} 0 & i = j, \\ \min_{i \le k < j} (m[i,k] + m[k+1,j] + p_{i-1}p_k p_j) & i < j \end{cases}$$

Algorithm:

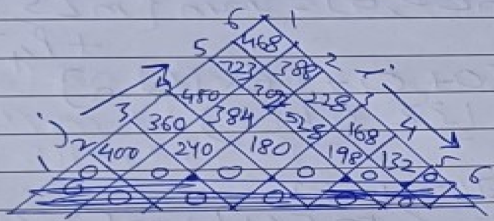
```
function MatrixChainOrder(dimensions):
    n = length(dimensions) - 1
    create an n x n table M


    for i = 1 to n:
        M[i][i] = 0


    for length = 2 to n:
        for i = 1 to n - length + 1:
            j = i + length - 1
            M[i][j] = infinity
```

```
        for k = i to j - 1:
            cost = M[i][k] + M[k+1][j] + dimensions[i-1] * dimensions[k] * dimensions[j]
            if cost < M[i][j]:
                M[i][j] = cost
return M[1][n]
```

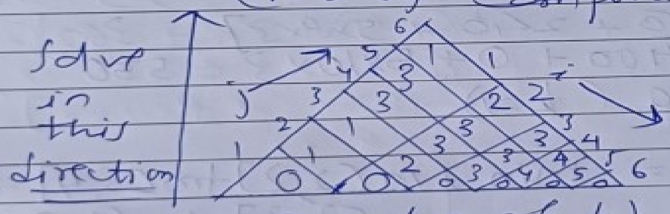
Example and solution for the example:

I/P: < 5, 8, 10, 3, 6, 11, 2 >

$A_1 = 5 \times 8$ $\quad P_0 = 5$
$A_2 = 8 \times 10$ $\quad P_1 = 8$
$A_3 = 10 \times 3$ $\quad P_2 = 10$
$A_4 = 3 \times 6$ $\quad P_3 = 3$
$A_5 = 6 \times 11$ $\quad P_4 = 6$
$A_6 = 11 \times 12$ $\quad P_5 = 11$
$\quad P_6 = 2$



(No. of computations) M[i, j]



(value of k)

<u>Iteration 1:</u>

$i = 1, j = 2;$ $\quad m[1,2] = \min\{m[1,1] + m[2,2] + P_0 P_1 P_2$
$i \le k \le j \quad k = 1$
$= [0 + 0 + 5 \times 8 \times 10] = \underline{400}$

$i = 2, j = 3 \quad m[2,3] = \min\{m[2,2] + m[3,3] + P_1 P_2 P_3 \quad (k = 1)$
$i \le k \le j \quad k = 2$
$= [0 + 0 + 8 \times 10 \times 3]$
$= \underline{240} \quad (k = 2)$

$i=3,\ j=4 \quad m[3,4] = \min\{m[3,3] + m[4,4] + P_2 P_3 P_4$

$i \le k \le j \quad \therefore k = 3$

$= [0 + 0 + 10 \times 3 \times 6]$

$= \underline{180} \quad (k=3)$

$i=4,\ j=5 \quad m[4,5] = \min\{m[4,4] + m[5,5] + P_3 P_4 P_5\}$

$k=4 \quad = [0 + 0 + 3 \times 6 \times 11]$

$= \underline{198} \quad (k=4)$

$i=5,\ j=6 \quad m[5,6] = \min\{m[5,5] + m[6,6] + P_4 P_5 P_6\}$

$k=5 \quad = [0 + 0 + 6 \times 11 \times 2]$

$= \underline{132} \quad (k=5)$

Iteration II :- $k=1,2$

$i=1,\ j=3 \quad m[1,3] = \min\begin{cases} m[1,1] + m[2,3] + P_0 P_3 P_1 \\ m[1,2] + m[3,3] + P_0 P_2 P_3 \end{cases}$

$= \min\begin{cases} [0 + 240 + 5 \times 8 \times 3] = 360 \\ [400 + 0 + 5 \times 10 \times 3] = 550 \end{cases}$

$= \underline{360} \quad (k=1)$

$i=2,\ j=4 \quad m[2,4] = \min\begin{cases} m[2,2] + m[3,4] + P_1 P_2 P_4 \\ m[2,3] + m[4,4] + P_1 P_3 P_4 \end{cases}$

$= \min\begin{cases} [0 + 180 + 8 \times 10 \times 6] = 660 \\ [240 + 0 + 8 \times 3 \times 6] = 384 \end{cases}$

$= \underline{384} \quad (k=3)$

$i=3,\ j=5 \quad m[3,5] = \min\begin{cases} m[3,3] + m[4,5] + P_2 P_3 P_5 \\ m[3,4] + m[5,5] + P_2 P_4 P_5 \end{cases}$

$= \min\begin{cases} [0 + 198 + 10 \times 3 \times 11] = 528 \\ [180 + 0 + 10 \times 6 \times 11] = 840 \end{cases}$

$= \underline{528} \quad (k=3)$

$i=4, j=6 \quad m[4,6] = \min \begin{cases} m[4,4] + m[5,6] + P_3 \times P_4 \times P_6 \\ m[4,5] + m[6,6] + P_3 \times P_5 \times P_6 \end{cases}$

$= \min \begin{cases} [0 + 132 + 36] = 168 \\ [198 + 0 + 66] = 264 \end{cases}$

$= \underline{168} \ (k=4)$

<u>Iteration III :-</u>

$i=1, j=4 \quad m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + P_0 P_1 P_4 \\ m[1,2] + m[3,4] + P_0 P_2 P_4 \\ m[1,3] + m[4,4] + P_0 P_3 P_4 \end{cases}$

$k = 1, 2, 3$

$= \min \begin{cases} [0 + 384 + 5 \times 8 \times 6] = 624 \\ [400 + 180 + 5 \times 10 \times 6] = 880 \\ [360 + 0 + 5 \times 3 \times 6] = 450 \end{cases}$

$= 450 \ (k=3)$

$i=2, j=5 \quad m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + P_1 P_2 P_5 \\ m[2,3] + m[4,5] + P_1 P_3 P_5 \\ m[2,4] + m[5,5] + P_1 P_4 P_5 \end{cases}$

$k = 2, 3, 4$

$= \min \begin{cases} [0 + 528 + 8 \times 10 \times 11] = 1408 \\ [240 + 198 + 8 \times 3 \times 11] = 702 \\ [384 + 0 + 8 \times 6 \times 11] = 912 \end{cases}$

$= 702 \ (k=3)$

$i=3, j=6 \quad m[3,6] = \min \begin{cases} m[3,3] + m[4,6] + P_2 P_3 P_6 \\ m[3,4] + m[5,6] + P_2 P_4 P_6 \\ m[3,5] + m[6,6] + P_2 P_5 P_6 \end{cases}$

$k = 3, 4, 5$

$= \min \begin{cases} [0 + 168 + 10 \times 3 \times 2] = 228 \\ [180 + 132 + 10 \times 6 \times 2] = 432 \\ [528 + 0 + 10 \cdot 11 \cdot 2] = 748 \end{cases}$

$= \underline{228} \ (k=3)$

Itr. IV :

$i=1, j=5$ $\quad m[1,5] = \min \begin{cases} m[1,1] + m[2,5] + P_0 P_1 P_5 \\ m[1,2] + m[3,5] + P_0 P_2 P_5 \\ m[1,3] + m[4,5] + P_0 P_3 P_5 \\ m[1,4] + m[5,5] + P_0 P_4 P_5 \end{cases}$

$k = 1,2,3,4$

$= \min \begin{cases} [0 + 702 + 5\times8\times11] = 1142 \\ [400 + 528 + 5\times10\times11] = 1478 \\ [360 + 198 + 5\times3\times11] = 723 \\ [4540 + 5\times6\times11] = 780 \end{cases}$

$= \underline{723}\ (k=3)$

$i=2, j=6$ $\quad m[2,6] = \min \begin{cases} m[2,2] + m[3,6] + P_1 P_2 P_6 \\ m[2,3] + m[4,6] + P_1 P_3 P_6 \\ m[2,4] + m[5,6] + P_1 P_4 P_6 \\ m[2,5] + m[6,6] + P_1 P_5 P_6 \end{cases}$

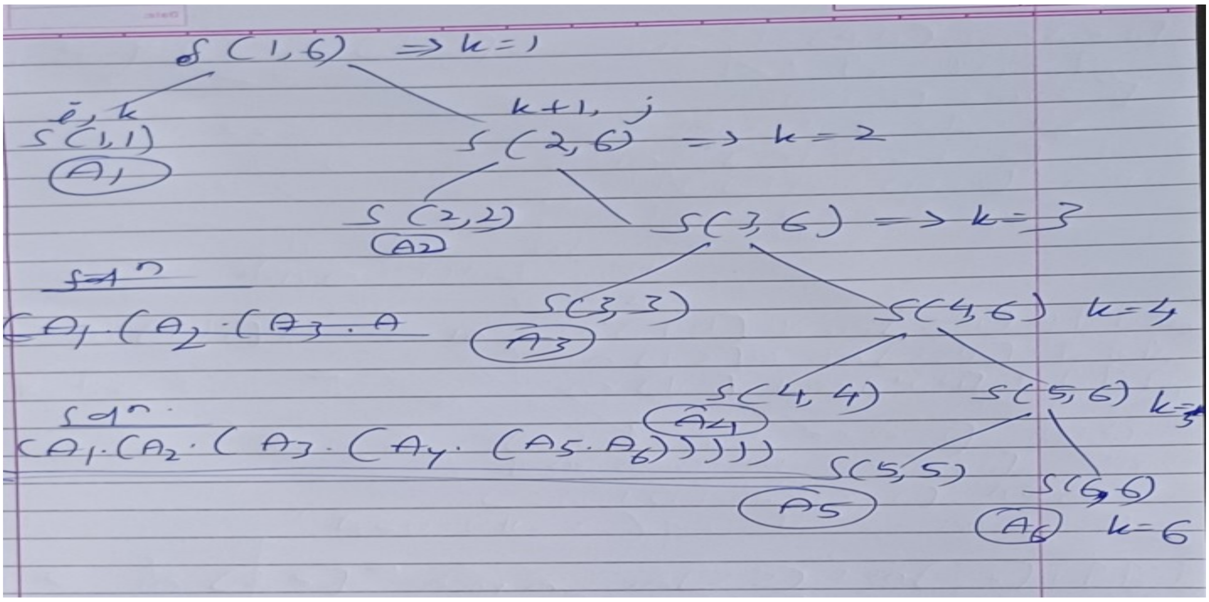
$= \min \begin{cases} [0 + 228 + 8\times10\times2] = 388 \\ [240 + 168 + 8\times3\times2] = \cancel{x} \\ [384 + 132 + 8\times6\times2] = \\ [702 + 0 + 8\times11\times2] = \end{cases}$

$= \underline{388}\ (k=2)$

Iter. V :

$i=1, j=6$ $\quad m[1,6] = \min \begin{cases} m[1,1] + m[2,6] + P_0 P_1 P_6 \\ m[1,2] + m[3,6] + P_0 P_2 P_6 \\ m[1,3] + m[4,6] + P_0 P_3 P_6 \\ m[1,4] + m[5,6] + P_0 P_4 P_6 \\ m[1,5] + m[6,6] + P_0 P_5 P_6 \end{cases}$

$= \underline{468}\ (k=1)$



Analysis of algorithm:

$= O(n) \times O(n)$

loop1 → $(n-1) \times 1$

Imp.

loop2 → $(n-2) \times 2$ $\quad O(n) \times O(n)$

loop 3 → $(n-3) \times 3$ $\quad O(n) \times O(n)$

⋮

loop n → $(n-(n-1)) \times n-1$ $\quad O(n) \times O(n)$

n times

$= O(n^3)$

Space complexity

$\frac{n}{2} \times \frac{n}{2} = O(n^2)$ matrix is maintained

matrix chain theory

matrix chain multplc^n is a dynamic programming algorithm which is used to find the most efficient method for multiplication of matrix.

Cost of multiplication

$$ {}^{n}[\quad]_{m} * {}^{m}[\quad]_{p} = {}^{n}[\quad]_{p} $$

Cost $= n \times m \times p$

CODE WITH OUTPUT:

Start here X Expt6.cpp X

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int MatrixChainOrder(vector<vector<int>>& matrixData, vector<vector<int>>& jaggArr) {
    int num = matrixData.size();
    for (int difference = 1; difference < num; difference++) {
        for (int j = 0; j < num - difference; j++) {
            int i = j + difference;
            jaggArr[j][i] = INT_MAX;
            for (int k = j; k < i; k++) {
                int temp = jaggArr[j][k] + jaggArr[k + 1][i] + matrixData[j][0] * matrixData[k][1] * matrixData[i][1];
                jaggArr[j][i] = min(jaggArr[j][i], temp);
            }
        }
    }
    return jaggArr[0][num - 1];
}

int main() {
    int num;
    cout << "Enter no of matrices: ";
    cin >> num;

    vector<vector<int>> matrixData(num, vector<int>(2));
    cout << "Enter dimensions of each matrix (rows & columns):\n";
    for (int i = 0; i < num; i++) {
        cout << "Matrix " << i + 1 << ": ";
        cin >> matrixData[i][0] >> matrixData[i][1];
    }

    vector<vector<int>> jaggArr(num, vector<int>(num, 0));


    MatrixChainOrder(matrixData, jaggArr);


    for (int i = 0; i < num; i++) {
        for (int j = 0; j < num; j++) {
            cout << jaggArr[i][j] << ",";
        }
        cout << endl;
    }
    return 0;
}
```

OUTPUT:

```
C:\Users\DELL\Desktop\Expt6.exe
Enter the number of matrices: 4
Enter the dimensions of each matrix (rows and columns):
Matrix 1: 4 8
Matrix 2: 8 6
Matrix 3: 6 18
Matrix 4: 18 36
0,192,624,3216,
0,0,864,5616,
0,0,0,3888,
0,0,0,0,


Process returned 0 (0x0)   execution time : 17.194 s
Press any key to continue.
```

CONCLUSION: We've implemented the dynamic programming approach for matrix chain multiplication, which operates with a time complexity of O(n^3) and a space complexity of O(n^2). Its primary application lies in determining the optimal sequence for multiplying a series of matrices efficiently.