Batch: A3	**Roll No.: 16010121045**
Experiment No : 8	
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.

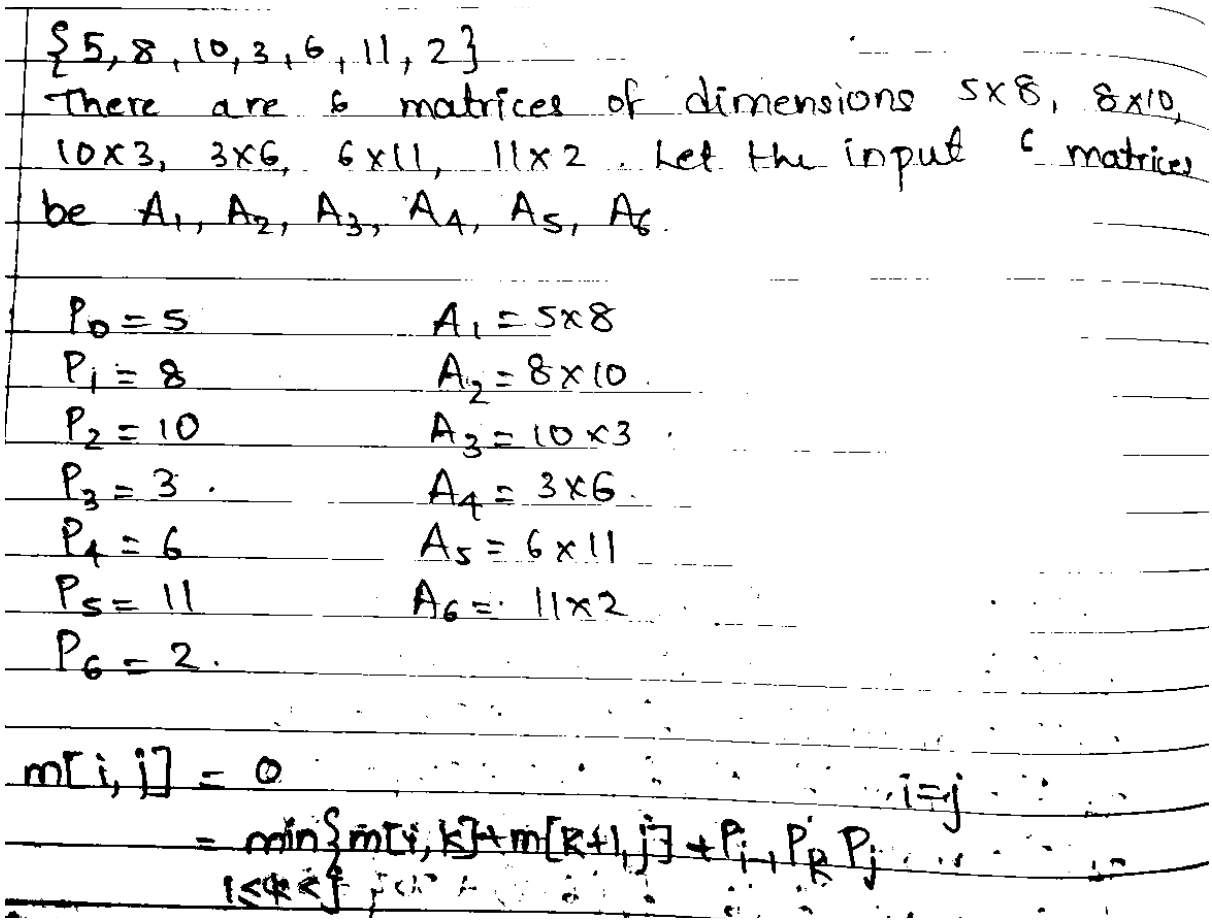
Recursive Formula:

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

Algorithm:

```
1. n   length[p]-1
2. for i ← 1 to n
3. do m [i, i] ← 0
4. for l ← 2 to n    // l is the chain length
5. do for i ← 1 to n-l + 1
6. do j ← i+ l -1
7. m[i,j] ←  ∞
8. for k  ← i to j-1
9. do q  ← m [i, k] + m [k + 1, j] + pi-1 pk pj
10. If q < m [i,j]
11. then m [i,j] ← q
12. s [i,j] ← k
13. return m and s.
```

Example:



Solution for the example:

```
#include <bits/stdc++.h>
using namespace std;
void printParenthesis(int i, int j, int n, int *bracket,
                      char &name)
{
    if (i == j)
    {
        cout << name++;
        return;
    }
    cout << "(";
    printParenthesis(i, *((bracket + i * n) + j), n,
                     bracket, name);
    printParenthesis(*((bracket + i * n) + j) + 1, j, n,
                     bracket, name);
    cout << ")";
```

```
}

void matrixChainOrder(int p[], int n)
{
    int m[n][n];
    int bracket[n][n];
    for (int i = 1; i < n; i++)
        m[i][i] = 0;

    for (int L = 2; L < n; L++)
    {
        for (int i = 1; i < n - L + 1; i++)
        {
            int j = i + L - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++)
            {
                int q = m[i][k] + m[k + 1][j] + p[i - 1] * p[k] *
p[j];
                if (q < m[i][j])
                {
                    m[i][j] = q;
                    bracket[i][j] = k;
                }
            }
        }
    }

    char name = 'A';

    cout << "Optimal Parenthesization is : ";
    printParenthesis(1, n - 1, n, (int *)bracket, name);
    cout << "\nOptimal Cost is : " << m[1][n - 1] << endl;
}

int main()
{
    int arr[] = {40, 20, 30, 10, 30};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    matrixChainOrder(arr, n);
    return 0;
}
```

Ans:

```
❯ cd "/Users/pargatsinghdhanjal/Desktop/Data-Struc
s/"matrix
Optimal Parenthesization is : (A(B(C(D(EF)))))
Optimal Cost is : 468
```

Analysis of algorithm:

Time Complexity is: O (n3)
There are three nested loops. Each loop executes a maximum n times.

Space Complexity is O(n*n) where n is the number present in the chain of the matrices. We create a DP matrix that stores the results after each operation.

CONCLUSION:

In this experiment, we have learnt Implementation of Matrix Chain Multiplication by dynamic programming approach.