Bharatiya Vidya Bhavan's
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Autonomous Institute Affiliated to University of Mumbai)
Munshi Nagar, Andheri (W), Mumbai – 400 058.
Department of Master of Computer Application

Experiment	3
Aim	To understand and implement Dynamic Programming Approach
Objective	1) Write Pseudocode for given problems and understanding the implementation of Dynamic Programming 2) Solve Matrix Multiplication Problem using Dynamic Programming 3) Calculating time complexity of the given problems
Name	Durgesh Dilip Mandge
UCID	2023510032
Class	FYMCA
Batch	B
Date of Submission	17-02-2023

Algorithm and Explanation of the technique used

```
// mat = Matrix chain of length n
// low = 1, j = n-1 initially

MatrixChainMultiplication(mat[], low , high):
   // 0 steps are required if low equals high
   If(low=high):
      return 0

   // Initialize minCost to a very big number.
   minCost = Infinity

   // Iterate from k = low to k = high-1
   For(k=low to high-1):
      /*
       Cost = Cost of Multiplying chain on left side +
            Cost of Multiplying chain on right side +
            Cost of Multiplying matrix obtained from left
            and right side.
      */
      cost=MatrixChainMultiplication(mat, low, k)+
         MatrixChainMultiplication(mat, low+1, high)+
         mat[low-1]*mat[k]*mat[high]

      // Update the minCost if cost<minCost.
      If(cost<minCost):
         minCost=cost

   return minCost
```

	Explaination: 1. We want to find the most efficient way to multiply these matrices together. The order in which we perform the multiplications affect the total number of scalar multiplications needed. 2. Recursive memoization is a technique of storing the results of intermediate computations to avoid redundant calculations. (Dynamic Programming) 3. We will make all possible valid combinations of matrix multiplications between the set of matrices provided to us and decide which combination gives us the smallest computation.
Program(Code)	

```java
import java.util.Arrays;

public class MCM{
    // Function to find the minimum number of Multiplication
    //  steps required in multiplying chain of n matrices.
    private static int MatrixChainMultiplication(int mat[], int low, int
high){
        // If we are left with one matrix then
        // we don't need any multiplication steps.
        if(low==high)
            return 0;

        // Initializing minCost with very
        // large value.
        int minCost=Integer.MAX_VALUE;

        // Iterating from low to high - 1
        for(int k=low;k<high;k++){
            /*
             Cost = Cost of Multiplying chain on left side +
                    Cost of Multiplying chain on right side +
                    Cost of Multiplying matrix obtained from left
                    and right side.
            */
            int cost=MatrixChainMultiplication(mat, low, k)+
                MatrixChainMultiplication(mat, k+1, high)+
                mat[low-1]*mat[k]*mat[high];

            // If the above cost is less than
            // minCost find so far then update minCost.
            if(cost<minCost)
                minCost=cost;
        }
        // Returning the minCost
        return minCost;
    }
    // Main Function
    public static void main(String args[]){
        // This matrix chain of length 5 represents
        // 4 matrices of dimensions as follows -
```

```
        int mat[]={32, 65, 98, 131, 164, 197, 230, 263, 296, 329};
        int n=mat.length;

        System.out.println("Given input of row and column is "+
Arrays.toString(mat));
        System.out.println("Minimum number of steps are - "+
                        MatrixChainMultiplication(mat, 1, n-1));
    }
}
```

Output

```
PS C:\Users\smart\Documents\SPIT-lab\Sem 2\DAA\Lab3&4>  c:; cd 'c:\Users\smart\Documents\SPIT-lab\Sem 2\D
a.exe' '--enable-preview' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\smart\AppData\Roaming
20604e1a0\redhat.java\jdt_ws\Lab3&4_7b75640c\bin' 'MCM'
Given input of row and column is [32, 65, 98, 131, 164, 197, 230, 263, 296, 329]
Minimum number of steps are - 11329024
PS C:\Users\smart\Documents\SPIT-lab\Sem 2\DAA\Lab3&4>
```

Justification of the complexity calculated

Time Complexity:

- In the function we are iterating from 1 to n−1 which costs O(n) and in each iteration, it takes O(n^2) time to calculate the answer of left and right sub-problems. Hence, the overall time complexity is O(n^3).
- The number of function call made by subproblem is 2 hence Complexity gets O(n^2)
- In Every function call the complexity required for calculation is directly proportional to size of input hence it is O(n)
- Then the total complexity becomes O(n^3)

Space Complexity:

- As we are storing every answer so far in the cell of 2D matrix of size (n*n) where 'l' is the number of matrices-1
- So for the worst case the space complexity is O(n^2)

Conclusion

- Matrix chain multiplication can be broken down into smaller subproblems and the redundant function calls can be avoided by storing the outcome in matrix.
- Solving matrix chain multiplication by optimized way leads to seamless calculation with less processing ability
- Applications of MCM:
  1. Vector graphics- 3D designs needs to transform the 2D plane into 3D space works on matrix calculations.
  2. Mathematics – Many math problems can be easily solved using matrices theorems.
  3. Scientific Calculators – Scientific calculators give precise solution of the matrix chain multiplications by using suitable choices.
- Use of Dynamic Programming:
  1. Because of the observation, MCM problem has subproblem and hence can be solved recursively.
  2. In recursive calls, some calls are redundant hence here comes dynamic programming and we save the result each time we get it.