# DAA Experiment-4

## (Batch-A/A1)

| | |
|---|---|
| **Name** | Ansari Mohammed Shanouf Valijan |
| **UID Number** | 2021300004 |
| **Class** | SY B.Tech Computer Engineering(Div-A) |
| **Experiment Number** | 4 |
| **Date of Performance** | 16-03-23 |
| **Date of Submission** | 22-03-23 |

**Aim:**
To implement dynamic algorithm for Matrix Chain Multiplication(MCM).

**Problem Definition and Assumptions:**
Consider the optimization problem of efficiently multiplying a randomly generated sequence of 10 matrices $M_1$, $M_2$, $M_3$, $M_4$, ..., $M_{10}$ using Dynamic programming approach. The dimensions of these matrices are stored in an array p[i] for i = 0 to 10, where the dimension of the matrix $M_i$ is p[i-1] x p[i]. All p[i] are randomly generated and they are between 15 and 46. For example, p[0...10] = {23, 20, 25, 45, 30, 35, 40, 22, 15, 29, 21}.

Determine following values of Matrix Chain Multiplication(MCM) using Dynamic Programming:

I. m[1..10][1..10] = Two-dimensional matrix of optimal solutions(No. of multiplications) of all possible matrices $M_1$ to $M_{10}$.
II. c[1..9][2..10] = Two-dimensional matrix of optimal solutions (parenthesizations) of all combinations of matrices $M_1$ to $M_{10}$.
III. The optimal solution(parenthesization) for the multiplication of all ten matrices $M_1$ to $M_{10}$.

**Theory:**
Dynamic programming is a method for solving optimization problems by breaking them down into smaller subproblems and solving each subproblem only once. This approach is particularly useful when the subproblems overlap or share sub-solutions, as it allows for efficient computation and avoids redundant

calculations. The key idea behind dynamic programming is to store the solutions to the subproblems in a table, so that they can be reused when needed. This is known as memoization, and it can significantly reduce the time complexity of an algorithm.

Dynamic programming can be applied to a wide range of problems, including optimization, sequencing, shortest path, and scheduling problems. Some well-known examples of problems that can be solved using dynamic programming include the Matrix Chain Multiplication problem, the Traveling Salesman problem, and the Longest Common Subsequence problem.

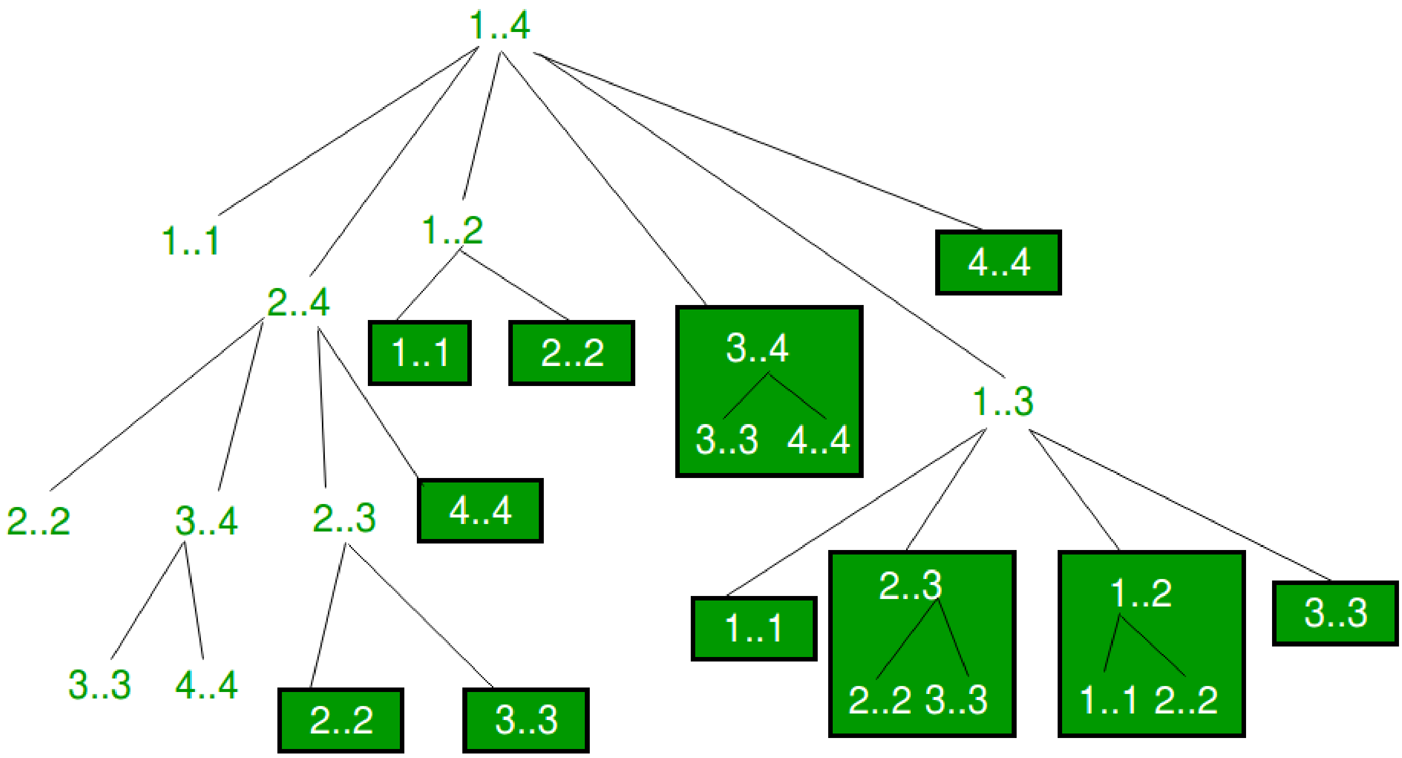
In this experiment, I have implemented the dynamic programming approach towards finding the optimal order of multiplying a chain of matrices in an attempt to minimize the amount of time taken for the same.

Matrix Chain Multiplication(MCM) is a problem in computer science that involves finding the most efficient way to multiply a series of matrices. The objective is to minimize the total number of scalar multiplications required to multiply the matrices together. The problem can be stated as follows: given a sequence of matrices $A_1$, $A_2$, ..., $A_n$, where the dimensions of matrix $A_i$ are p[i-1] x p[i], find the order in which to multiply the matrices that minimizes the total number of scalar multiplications.

For example, if we have matrices $A_1$ with dimensions 10 x 20, $A_2$ with dimensions 20 x 30, and $A_3$ with dimensions 30 x 40, there are two possible ways to multiply them: ($A_1$ x $A_2$) x $A_3$ or $A_1$ x ($A_2$ x $A_3$). The number of scalar multiplications required for each option is different, and the objective is to find the order that minimizes the total number of scalar multiplications. MCM can be solved efficiently using dynamic programming. The key idea is to break the problem down into smaller subproblems and build up a table of solutions that can be used to solve larger subproblems. The time complexity of the dynamic programming solution is O($n^3$), where n is the number of matrices in the sequence.

Overall, MCM is an important problem in computer science that has practical applications in areas such as computer graphics, data compression, and optimization.

Given below is an image that conceptualizes how a solution is generated using dynamic programming for matrix chain multiplication-




Here, the optimal parts are marked in green which can later be used for parenthesization.

**Algorithm:**

[A] For Matrix Chain Multiplication-

I. Start.
II. Take the range of matrices from the user, say from $A_i$ to $A_j$.
III. For i<=k<j, divide the provided range of matrices into two parts having matrices $A_i$ to $A_k$ and $A_{k+1}$ to $A_j$ respectively.
IV. For each set of divisions, calculate the number of scalar products required using dynamic programming approach.
V. Store the minimum scalar products required in a table. Also, store the k value at which the minimum was obtained in a separate table.
VI. End.

[B] For parenthesization-

I. Start.
II. Iterate over the k value table and recursively store the number of opening and closing brackets for each matrix.

III. Use the stored information while printing the final parenthesized expression.
IV. End.

**Program:**

```
//header files
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

//function for creation and destruction of 2D arrays
int** createArr(int row, int column){
    int** arr=(int**)calloc(row,sizeof(int*));
    for(int i=0; i<row; i++)
        arr[i]=(int*)calloc(column,sizeof(int));
    return arr;
}

void destroyArr(int** arr, int row){
    for(int i=0; i<row; i++)
        free(arr[i]);
    free(arr);
}

//function for randomly populating the dimension array
int* generateDimensions(int size, int startVal, int endVal){
    int* dim=(int*)malloc(size*sizeof(int));
    for(int i=0; i<size; i++)
        dim[i]=startVal+rand()%(endVal-startVal+1);
    return dim;
}

//functions for finding optimal number of scalar products and corresponsing k values
void matrixChainMul(int* dim, int** optimalVal, int** kVal, int i, int j){
    if(i!=j && optimalVal[i][j]==0){
        int tempVal=0, optimal, kOpt=i, k=i;
        optimal=optimalVal[i][k]+optimalVal[k+1][j]+dim[i-1]*dim[k]*dim[j];
        k++;
        while(k<j){
            tempVal=optimalVal[i][k]+optimalVal[k+1][j]+dim[i-1]*dim[k]*dim[j];
            if(tempVal<optimal){
                optimal=tempVal;
                kOpt=k;
            }
            k++;
        }
        optimalVal[i][j]=optimal;
        kVal[i][j]=kOpt;
    }
}
```

```c
void fillOptimalSolution(int* dim, int** optimalVal, int** kVal, int numOfMat){
    int offset;
    for(int d=numOfMat-1; d>0; d--){
        offset=numOfMat-d;
        for(int i=1; i<=d; i++){
            matrixChainMul(dim, optimalVal, kVal, i, i+offset);
        }
    }
}

//function for printing the required tables
void printTab(int** table, int size){
    printf("\t");
    for(int i=1; i<size; i++){
        printf("%d\t",i);
    }
    printf("\n");
    for(int i=0; i<size; i++){
        printf("--------");
    }
    printf("\n");
    for(int i=1; i<size; i++){
        printf("%d\t",i);
        for(int j=1; j<size; j++){
            if(table[i][j]==0)
                printf("-\t");
            else
                printf("%d\t",table[i][j]);
        }
        printf("\n");
    }
}

//functions for determining parenthesization
void findParenthesisInfo(int** parenthesis, int** kVal, int i, int j){
    int k=kVal[i][j];
    if(j-i+1>2){
        if(k-i+1>1){
            parenthesis[i][0]++;
            parenthesis[k][1]++;
            findParenthesisInfo(parenthesis,kVal,i,k);
        }
        if(j-k>1){
            parenthesis[k+1][0]++;
            parenthesis[j][1]++;
            findParenthesisInfo(parenthesis,kVal,k+1,j);
        }
    }
}

void printMatMulExp(int** parenthesis, int numOfMat){
    for(int i=1; i<=numOfMat; i++){
        for(int j=0; j<parenthesis[i][0]; j++){
            printf("(");
```

```c
        }
        printf("M%d",i);
        for(int j=0; j<parenthesis[i][1]; j++){
            printf(")");
        }
    }
}

//function to calculate the number of scalar products under trivial matrix multiplication
int trivialMatMul(int* dim, int numOfMat){
    int sum=0;
    for(int i=1; i<=numOfMat-1; i++)
        sum+=dim[0]*dim[i]*dim[i+1];
    return sum;
}

//main function
void main(){
    srand(time(0));
    //taking user input
    int num;
    printf("\nEnter the number of matrices that you want to multiply -----> ");
    scanf("%d",&num);

    //displaying the input configuration the program will be dealing with
    int* dim=generateDimensions(num+1,15,46);
    printf("\nThe following dimension matrix was randomly generated having values between 
15 and 46 -\n");
    for(int i=0; i<=num; i++)
        printf("%d\t",dim[i]);
    printf("\n\nThat is, the following matrices are taken into consideration-\n\n");
    for(int i=1; i<=num; i++)
        printf("M%d - order(%dx%d)\n",i,dim[i-1],dim[i]);
    printf("\n");

    //calculating the optimal multiplication order using dynamic programming approach
    int** optimalVal=createArr(num+1,num+1);
    int** kVal=createArr(num+1,num+1);
    fillOptimalSolution(dim,optimalVal,kVal,num);

    //displaying the results
    printf("Following tabular data was obtained-\n\n");
    printf("I. Table showing the optimal number of multiplications required at each step-
\n\n");
    printTab(optimalVal,num+1);
    printf("\nII. Table showing the k values at which optimal solution was obtained at 
each step-\n\n");
    printTab(kVal,num+1);
    printf("\nOptimal Parenthesization is as follows-\n\n");
    int** parenthesis=createArr(num+1,2);
    findParenthesisInfo(parenthesis,kVal,1,num);
    printMatMulExp(parenthesis,num);
    printf("\n\n");
    printf("Summary-\n\n");
```

```
    int sum=trivialMatMul(dim,num);
    printf("Number of scalar products required under trivial matrix chain multiplication: %d\n",sum);
    printf("Number of scalar products required under optimal matrix chain multiplication: %d\n",optimalVal[1][num]);
    printf("Hence, optimal solution is %.2lf times faster than the trivial solution\n\n",(double)sum/optimalVal[1][num]);

    //de-allocating all the used locations
    destroyArr(optimalVal,num+1);
    destroyArr(kVal,num+1);
    destroyArr(parenthesis,num+1);
    free(dim);
}
```

## Implementation:

```
Enter the number of matrices that you want to multiply -----> 10

The following dimension matrix was randomly generated having values between 15 and 46 -
45      38      32      18      41      19      24      43      27      34      28

That is, the following matrices are taken into consideration-

M1 - order(45x38)
M2 - order(38x32)
M3 - order(32x18)
M4 - order(18x41)
M5 - order(41x19)
M6 - order(19x24)
M7 - order(24x43)
M8 - order(43x27)
M9 - order(27x34)
M10 - order(34x28)

Following tabular data was obtained-

I. Table showing the optimal number of multiplications required at each step-

        1       2       3       4       5       6       7       8       9       10
----------------------------------------------------------------------------------------
1       -       54720   52668   85878   80560   94338   128304  136242  158436  170712
2       -       -       21888   49932   48070   60534   92106   102060  123372  136404
3       -       -       -       23616   24966   36054   65574   77256   97812   111492
4       -       -       -       -       14022   22230   40806   61704   78228   95364
5       -       -       -       -       -       18696   53105   61209   84104   97518
6       -       -       -       -       -       -       19608   40176   57618   75706
7       -       -       -       -       -       -       -       27864   49896   71712
8       -       -       -       -       -       -       -       -       39474   58212
9       -       -       -       -       -       -       -       -       -       25704
10      -       -       -       -       -       -       -       -       -       -
```

```
II. Table showing the k values at which optimal solution was obtained at each step-

        1       2       3       4       5       6       7       8       9       10
----------------------------------------------------------------------------------------
1       -       1       1       3       1       3       3       3       3       3
2       -       -       2       3       2       3       3       3       3       3
3       -       -       -       3       3       3       3       3       3       3
4       -       -       -       -       4       5       6       7       8       9
5       -       -       -       -       -       5       5       5       5       5
6       -       -       -       -       -       -       6       6       8       9
7       -       -       -       -       -       -       -       7       8       8
8       -       -       -       -       -       -       -       -       8       8
9       -       -       -       -       -       -       -       -       -       9
10      -       -       -       -       -       -       -       -       -       -

Optimal Parenthesization is as follows-

(M1(M2M3))((((((M4M5)M6)M7)M8)M9)M10)

Summary-

Number of scalar products required under trivial matrix chain multiplication: 352260
Number of scalar products required under optimal matrix chain multiplication: 170712
Hence, optimal solution is 2.06 times faster than the trivial solution
```

```
Enter the number of matrices that you want to multiply -----> 20

The following dimension matrix was randomly generated having values between 15 and 46 -
37      37      29      31      38      25      28      15      16      20      35      18      36      32      37      25      32      22      37      17      46

That is, the following matrices are taken into consideration-

M1 - order(37x37)
M2 - order(37x29)
M3 - order(29x31)
M4 - order(31x38)
M5 - order(38x25)
M6 - order(25x28)
M7 - order(28x15)
M8 - order(15x16)
M9 - order(16x20)
M10 - order(20x35)
M11 - order(35x18)
M12 - order(18x36)
M13 - order(36x32)
M14 - order(32x37)
M15 - order(37x25)
M16 - order(25x32)
M17 - order(32x22)
M18 - order(22x37)
M19 - order(37x17)
M20 - order(17x46)
```

Following tabular data was obtained-

I. Table showing the optimal number of multiplications required at each step-

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 39701 | 72964 | 114637 | 112975 | 138875 | 92535 | 101415 | 108435 | 127260 | 125205 | 144915 | 159975 | 180510 | 187725 | 203610 | 208620 | 229155 | 225293 | 253118 |
| 2 | - | - | 33263 | 74936 | 78750 | 102269 | 72000 | 80033 | 87900 | 106725 | 104670 | 124380 | 139440 | 159975 | 167190 | 183075 | 188085 | 208620 | 204758 | 232583 |
| 3 | - | - | - | 34162 | 51925 | 72225 | 55905 | 62865 | 69405 | 86430 | 86415 | 103965 | 119505 | 139440 | 148095 | 163140 | 169350 | 188085 | 186623 | 209301 |
| 4 | - | - | - | - | 29450 | 51150 | 42420 | 49860 | 56520 | 73995 | 73470 | 91560 | 106980 | 127065 | 135360 | 150615 | 156525 | 175710 | 173648 | 197890 |
| 5 | - | - | - | - | - | 26600 | 24750 | 31700 | 40950 | 60000 | 57030 | 77670 | 92670 | 113280 | 120315 | 136305 | 141165 | 161925 | 156348 | 186023 |
| 6 | - | - | - | - | - | - | 10500 | 16500 | 22800 | 38925 | 39930 | 56130 | 72180 | 91815 | 101190 | 115815 | 122625 | 140460 | 140198 | 159748 |
| 7 | - | - | - | - | - | - | - | 6720 | 13200 | 30000 | 30240 | 47520 | 63120 | 82980 | 91815 | 106755 | 113115 | 131625 | 130463 | 152359 |
| 8 | - | - | - | - | - | - | - | - | 4800 | 15300 | 22680 | 32400 | 49680 | 67440 | 81315 | 93315 | 103875 | 116085 | 123323 | 135053 |
| 9 | - | - | - | - | - | - | - | - | - | 11200 | 18360 | 28728 | 47160 | 66104 | 80904 | 93704 | 104968 | 117992 | 124790 | 137302 |
| 10 | - | - | - | - | - | - | - | - | - | - | 12600 | 25560 | 44856 | 67968 | 80298 | 96298 | 106290 | 122570 | 124474 | 140114 |
| 11 | - | - | - | - | - | - | - | - | - | - | - | 22680 | 40896 | 65358 | 74448 | 93258 | 99630 | 123732 | 116464 | 143834 |
| 12 | - | - | - | - | - | - | - | - | - | - | - | - | 20736 | 42048 | 58698 | 73098 | 85770 | 100422 | 105754 | 119830 |
| 13 | - | - | - | - | - | - | - | - | - | - | - | - | - | 42624 | 58400 | 87200 | 89342 | 118646 | 94843 | 122995 |
| 14 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 29600 | 55200 | 63998 | 90046 | 75259 | 100283 |
| 15 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 29600 | 37950 | 68068 | 55131 | 84065 |
| 16 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 17600 | 37950 | 39406 | 58956 |
| 17 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 26048 | 25806 | 50830 |
| 18 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 13838 | 31042 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 28934 |
| 20 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

II. Table showing the k values at which optimal solution was obtained at each step-

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 1 | 2 | 2 | 1 | 5 | 1 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 2 | - | - | 2 | 2 | 2 | 2 | 2 | 2 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 3 | - | - | - | 3 | 3 | 5 | 3 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 19 |
| 4 | - | - | - | - | 4 | 5 | 4 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 19 |
| 5 | - | - | - | - | - | 5 | 5 | 5 | 7 | 7 | 5 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 7 |
| 6 | - | - | - | - | - | - | 6 | 7 | 7 | 7 | 7 | 11 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 19 |
| 7 | - | - | - | - | - | - | - | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 19 |
| 8 | - | - | - | - | - | - | - | - | 8 | 9 | 8 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 17 | 19 |
| 9 | - | - | - | - | - | - | - | - | - | 9 | 9 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 17 | 19 |
| 10 | - | - | - | - | - | - | - | - | - | - | 10 | 11 | 11 | 11 | 11 | 15 | 11 | 17 | 11 | 19 |
| 11 | - | - | - | - | - | - | - | - | - | - | - | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 19 |
| 12 | - | - | - | - | - | - | - | - | - | - | - | - | 12 | 13 | 14 | 15 | 16 | 17 | 15 | 19 |
| 13 | - | - | - | - | - | - | - | - | - | - | - | - | - | 13 | 13 | 15 | 13 | 17 | 13 | 19 |
| 14 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 14 | 15 | 14 | 17 | 14 | 19 |
| 15 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 15 | 15 | 17 | 15 | 19 |
| 16 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 16 | 17 | 16 | 19 |
| 17 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 17 | 17 | 19 |
| 18 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 18 | 19 |
| 19 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | 19 |
| 20 | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

Optimal Parenthesization is as follows-

(M1(M2(M3(M4(M5(M6M7))))))((((((((M8(M9(M10M11)))M12)M13)M14)M15)M16)M17)(M18M19))M20)

Summary-

Number of scalar products required under trivial matrix chain multiplication: 545676
Number of scalar products required under optimal matrix chain multiplication: 253118
Hence, optimal solution is 2.16 times faster than the trivial solution

```
Enter the number of matrices that you want to multiply -----> 5

The following dimension matrix was randomly generated having values between 15 and 46 -
27      31      21      32      44      46

That is, the following matrices are taken into consideration-

M1 - order(27x31)
M2 - order(31x21)
M3 - order(21x32)
M4 - order(32x44)
M5 - order(44x46)

Following tabular data was obtained-

I. Table showing the optimal number of multiplications required at each step-

        1       2       3       4       5
------------------------------------------------
1       -       17577   35721   72093   115731
2       -       -       20832   58212   102018
3       -       -       -       29568   72072
4       -       -       -       -       64768
5       -       -       -       -       -

II. Table showing the k values at which optimal solution was obtained at each step-

        1       2       3       4       5
------------------------------------------------
1       -       1       2       2       2
2       -       -       2       2       2
3       -       -       -       3       4
4       -       -       -       -       4
5       -       -       -       -       -

Optimal Parenthesization is as follows-

(M1M2)((M3M4)M5)

Summary-

Number of scalar products required under trivial matrix chain multiplication: 128385
Number of scalar products required under optimal matrix chain multiplication: 115731
Hence, optimal solution is 1.11 times faster than the trivial solution
```

**Inference:**

From above implementations, I observed that optimal order of multiplying a chain of matrices can be a crucial factor in reducing the time an algorithm takes to multiply matrices. I also noticed that the effect of optimal multiplication grows as a function of number of matrices participating in multiplication. For example, when the number of matrices considered was 5, optimal multiplication was about 1.11 times faster than the trivial multiplication; however, when 20 matrices were considered, it was 2.16 times faster. This clearly shows that optimal multiplication order is a major aiding factor in improvising the efficiency of algorithms that deal with matrix multiplication.

**Conclusion:**

By performing this experiment, I was able to understand how one can optimize matrix chain multiplication using dynamic programming approach. I was also able to implement the same.