Type = Partition DP - MCM

```
static int matrixMultiplication(int arr[]) {
    int n = arr.length;

    int[][] dp = new int[n][n];

    for(int len=2; len<n; len++) {          // consecutive subarray
        // Means taking len=2 (subsequence) in the given array of matrix
        // so after each iteration we increase the
        // length of the taken matrix at a time
        // and store the minm value in the dp array.

        for(int lowerRangeI=0; lowerRangeI<n-len; lowerRangeI++) {

            int upperRangeJ = lowerRangeI + len;
            // Like if we have
            //  P0  P1  P2  P3  P4  P5  P6
            //  i=0 <-len=2-> j=i+len=0+2=2
            //  i=0 <-len=2-> j=i+len=1+2=3
            // dp[i][j] = Integer.MAX
            // And so on we do for subsequence of
            // length 3, then 4, then 5, then 6 <= n(7)
            // Now between this range we can add
            // a middle mover k to give multiplication
            // result i.e. i -> k -> j  // k will partition it
            for(int k=i+1; k<j; k++) {
                ...
```

lowerRangeI upperRangeJ

// i → k → j e.g. P_1 P_2 P_3 P_4 P_5 (i, k, k, k, j)

K=2 ⇒ DP[i][k] + DP[k+1][j]

K=3 =

K=4

dp[i][j] = Math.min(

dp[i][j], (so k only & not k+1; this should be k+1)

dp[i][k] + dp[k][j]

+ $P_i * P_k * P_j$ → this means

// so dp[i][j] will change everytime k moves from i+1 to j-1 i.e. b/w i & j.

// so from them get the minm value.

// At the end dp[0][n-1] which means taking subsets away from 0 to n-1 index subsequence means taking all will be the ans.

// answer.

// Hence return dp[0][n-1]

So

$(M_{12} \times M_{23}) \times (M_{34} \times M_{45})$

$= (M_{13}) * (M_{35})$

$= P_0 * P_2 * P_5$

$= P_i * P_k * P_j$

M_{01} M_{12} M_{13} M_{34}

P_0 P_1 P_2 P_3 P_4 P_5

1 2 3 4 5

i k j

dp[1][3] → Mean this cell represents taken Subsequence P_1 P_2 P_3

dp[0][2] = $P_0 M_{12}$ = $P_0 \times P_1 \times P_2$ → Means this two Subsequence Matrix

dp[1][3] = $M_{12} M_{23}$ = $P_1 * P_2 * P_3$ → Means this

dp[i][j]

dp[1][0] This is going in reverse subsequence $P_1 P_0$ → does not make any sense so just 'gmax

dp[7][3] → reverse subsequence (P_7 P_6 P_5 P_4 P_3) taken toge does not make any sense.

only by this dp[i][i] because i ≥ j+2

For this case only we can have max in multiply so len = 2.

dp[5][6] repres single matrix dp[5][5] only so value =

dp[7][7] just dp[6][6] taki P_6 or P_7 alone does not even make a

$M_{2\times3}$ $M_{3\times4}$ $M_{4\times5}$ $M_{5\times6}$ $M_{6\times7}$ $M_{7\times8}$

M_{01} M_{12} M_{23} M_{34} M_{45} M_{56}

P_0 P_1 P_2 P_3 P_4 P_5 P_6

2 3 4 5 6 7 8

P_0 = Just nothing

$P_0 P_1$ = M_{01} Matrix together only then they make. = own multiply = 0

$P_0 P_1 P_2$ = $M_{01} \times M_{12}$ ⇒ Means 2 matrix size

⇒ so len = 2 taken.

Subarray (consecutive) Subsequence (They may or may not be consecutive)

DP Tough Questions

Write

Post Title

Matrix Chain Multiplication | DP-8

Given the dimension of a sequence of matrices in an array **arr[]**, where the dimension of the **i^th** matrix is **(arr[i-1] * arr[i])**, the task is to find the most efficient way to multiply these matrices together such that the total number of element multiplications is minimum.

i starts from 1 of course

Examples:

1 2 3 4 Matrices

Input: *arr[] = {40, 20, 30, 10, 30}*
Output: *26000*
Explanation: *There are 4 matrices of dimensions 40x20, 20x30, 30x10, 10x30.*
Let the input 4 matrices be A, B, C and D.
The minimum number of multiplications are obtained by putting parenthesis in following way (A(BC))D.
*The minimum is 20*30*10 + 40*20*10 + 40*10*30*

Input: *arr[] = {1, 2, 3, 4, 3}*
Output: *30*
Explanation: *There are 4 matrices of dimensions 1x2, 2x3, 3x4, 4x3.*
Let the input 4 matrices be A, B, C and D.
The minimum number of multiplications are obtained by putting parenthesis in following way ((AB)C)D.
*The minimum number is 1*2*3 + 1*3*4 + 1*4*3 = 30*

Input: *arr[] = {10, 20, 30}*
Output: *6000*
Explanation: *There are only two matrices of dimensions 10x20 and 20x30.*
*So there is only one way to multiply the matrices, cost of which is 10*20*30*

You can directly watch the YT. then check the Iterative Tabulation approach only. Then Next in medium note check for how 2DP graph used to solve this problem logically.

Matrix Chain Multiplication using Recursion:

We can solve the problem using recursion based on the following facts and observations:

*Two matrices of size **m*n** and **n*p** when multiplied, they generate a matrix of size **m*p** and the number of multiplications performed are **m*n*p**.*

*Now, for a given chain of **N** matrices, the first partition can be done in **N-1** ways. For example, sequence of matrices A, B, C and D can be grouped as (A)(BCD), (AB)(CD) or (ABC)(D) in these 3 ways.*

*So a range **[i, j]** can be broken into two groups like **{[i, i+1], [i+1, j]}, {[i, i+2], [i+2, j]}, . . . , {[i, j-1], [j-1, j]}**.*

- *Each of the groups can be further partitioned into smaller groups and we can find the total required multiplications by solving for each of the groups.*

- *The minimum number of multiplications among all the first partitions is the required answer.*

Follow the steps mentioned below to implement the approach:

- Create a recursive function that takes **i** and **j** as parameters that determines the range of a group.
  - Iterate from **k = i to j** to partition the given range into two groups.
  - Call the recursive function for these groups.
  - Return the minimum value among all the partitions as the required minimum number of multiplications to multiply all the matrices of this group.
- The minimum value returned for the range **0** to **N-1** is the required answer.

Below is the implementation of the above approach.

C++ C Java Python3 C# PHP Javascript

Approach is quite difficult to understand

```
// Java code to implement the
// matrix chain multiplication using recursion

class MatrixChainMultiplication {

    // Matrix Ai has dimension p[i-1] x p[i]
    // for i = 1 . . . n
    static int MatrixChainOrder(int p[], int i, int j)
    {
        if (i == j)
            return 0;

        int min = Integer.MAX_VALUE;

        // Place parenthesis at different places
        // between first and last matrix,
        // recursively calculate count of multiplications
        // for each parenthesis placement
        // and return the minimum count
        for (int k = i; k < j; k++) {
            int count = MatrixChainOrder(p, i, k)
                        + MatrixChainOrder(p, k + 1, j)
                        + p[i - 1] * p[k] * p[j];

            if (count < min)
                min = count;
        }

        // Return minimum count
        return min;
    }

    // Driver code
```

A recursion always stops at base case

Dividing the range in 2 groups in recursion

when it goes recursively to bottom then only it will come to this step & multiply.

k is mid element from where we are doing diversion into two and taking all possible recursion min count.

```
    public static void main(String args[])
    {
        int arr[] = new int[] { 1, 2, 3, 4, 3 };
        int N = arr.length;

        // Function call
        System.out.println(
            "Minimum number of multiplications is "
            + MatrixChainOrder(arr, 1, N - 1));
    }
}
/* This code is contributed by Rajat Mishra*/
```

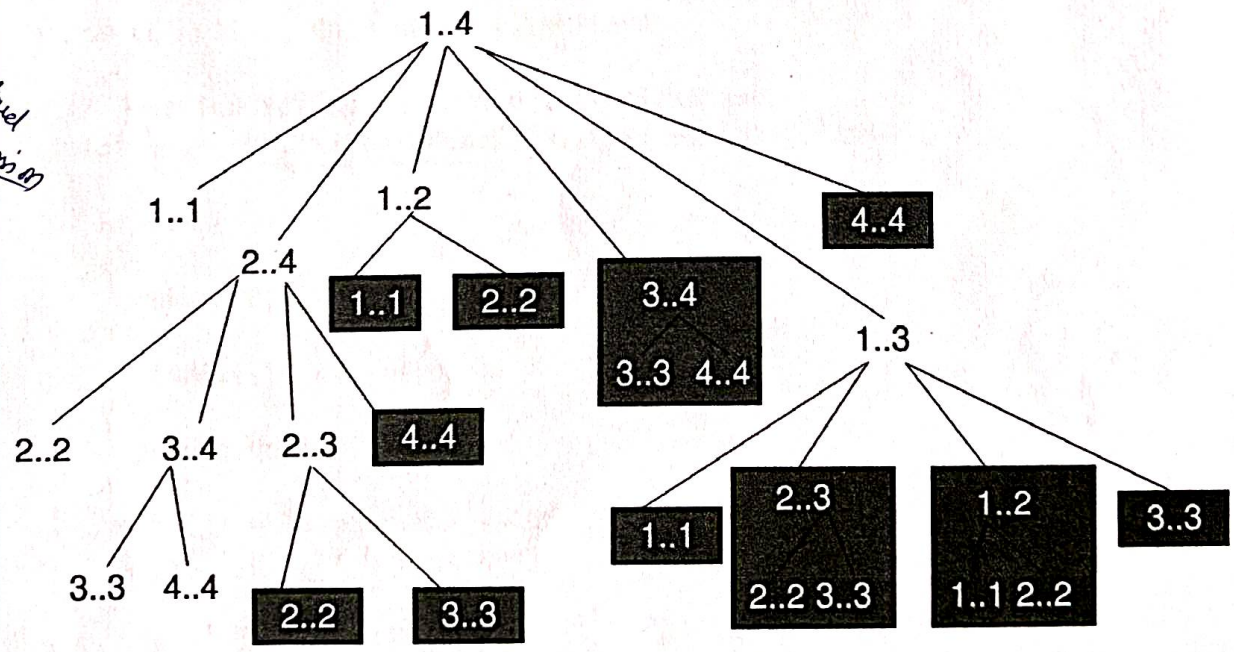
Output

```
Minimum number of multiplications is 30
```

The time complexity of the solution is exponential
Auxiliary Space: O(1)

Dynamic Programming Solution for Matrix Chain Multiplication using Memoization:

Below is the recursion tree for the 2nd example of the above recursive approach:



If observed carefully you can find the following two properties:

1) Optimal Substructure: *In the above case, we are breaking the bigger groups into smaller subgroups and solving them to finally find the minimum number of multiplications. Therefore, it can be said that the problem has optimal substructure property.*

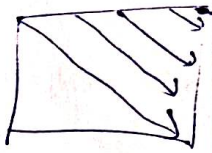
2) Overlapping Subproblems: *We can see in the recursion tree that the same subproblems are called again and again and this problem has the Overlapping Subproblems property.*

*So Matrix Chain Multiplication problem has both properties of a dynamic programming problem. So recomputations of same subproblems can be avoided by constructing a temporary array **dp[][]** in a bottom up manner.*

→ Iterative next page

Follow the below steps to solve the problem:

- Build a matrix **dp[][]** of size **N*N** for memoization purposes.
- Use the same recursive call as done in the above approach:
  - When we find a range **(i, j)** for which the value is already calculated, return the minimum value for that range (i.e., **dp[i][j]**).
  - Otherwise, perform the recursive calls as mentioned earlier.
- The value stored at **dp[0][N-1]** is the required answer.



Below is the implementation of the above approach

C++ | Java | Python3 | C# | Javascript

```
// Java program using memoization
import java.io.*;
import java.util.*;
class GFG
{

  static int[][] dp = new int[100][100];

  // Function for matrix chain multiplication
  static int matrixChainMemoised(int[] p, int i, int j)
  {
    if (i == j)
    {
      return 0;
    }
    if (dp[i][j] != -1)
    {
      return dp[i][j];
    }
    dp[i][j] = Integer.MAX_VALUE;
    for (int k = i; k < j; k++)
    {
      dp[i][j] = Math.min(
        dp[i][j], matrixChainMemoised(p, i, k)
        + matrixChainMemoised(p, k + 1, j)
        + p[i - 1] * p[k] * p[j]);
    }
    return dp[i][j];
  }

  static int MatrixChainOrder(int[] p, int n)
```

→ K as mid division

First understand Recursion then you will understand

```
                }
            }
        }

        return m[1][n - 1];
    }

    // Driver code
    public static void main(String args[])
    {
        int arr[] = new int[] { 1, 2, 3, 4 };
        int size = arr.length;

        System.out.println(
            "Minimum number of multiplications is "
            + MatrixChainOrder(arr, size));
    }
}
/* This code is contributed by Rajat Mishra*/
```

Output

```
Minimum number of multiplications is 18
```

Time Complexity: $O(N^3)$
Auxiliary Space: $O(N^2)$

Matrix Chain Multiplication (A O(N^2) Solution)
Printing brackets in Matrix Chain Multiplication Problem
Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Applications:
Minimum and Maximum values of an expression with * and +

Words: 915
Characters: 5511

- The number of multiplications to be performed to multiply these two matrices (say **X**) are **arr[i-1]*arr[k]*arr[j]**.
- The total number of multiplications is **dp[i][k]+ dp[k+1][j] + X**.

- The value stored at **dp[1][N-1]** is the required answer.

Below is the implementation of the above approach.

C++ C Java Python3 C# PHP Javascript

```
// Dynamic Programming Java implementation of Matrix
// Chain Multiplication.
// See the Cormen book for details of the following
// algorithm
class MatrixChainMultiplication
{

    // Matrix Ai has dimension p[i-1] x p[i] for i = 1..n
    static int MatrixChainOrder(int p[], int n)
    {
        /* For simplicity of the
        program, one extra row and
        one extra column are allocated in m[][]. 0th row
        and 0th column of m[][] are not used */
        int m[][] = new int[n][n];

        int i, j, k, L, q;

        /* m[i, j] = Minimum number of scalar
        multiplications needed to compute the matrix
        A[i]A[i+1]...A[j] = A[i..j] where
        dimension of A[i] is p[i-1] x p[i] */

        // cost is zero when multiplying one matrix.
        for (i = 1; i < n; i++)
            m[i][i] = 0;

        // L is chain length.
        for (L = 2; L < n; L++)
        {
            for (i = 1; i < n - L + 1; i++)
            {
                j = i + L - 1;
                if (j == n)
                    continue;
                m[i][j] = Integer.MAX_VALUE;
                for (k = i; k <= j - 1; k++)
                {
                    // q = cost/scalar multiplications
                    q = m[i][k] + m[k + 1][j]
                        + p[i - 1] * p[k] * p[j];
                    if (q < m[i][j])
                        m[i][j] = q;
```

Look this as well, on making the columns.

So

This means Taken Matrix & chain length at a time

Taken i at a time makes no sense as it is not matrix at all,

Let say this is our matrix

M_1 M_2 M_3 M_4 M_5

5, 4, 2, 3, 4, 6 = P[]

P_0 P_1 P_2 P_3 P_4 P_5

L=6

i=1 i++ (L=2) j

j++

j should be L-2 range

Iterate b/w them

understanding this is still a mystery for me, will check later stages.

```
    {
      int i = 1, j = n - 1;
      return matrixChainMemoised(p, i, j);
    }

    // Driver Code
    public static void main (String[] args)
    {

      int arr[] = { 1, 2, 3, 4 };
      int n= arr.length;

      for (int[] row : dp)
        Arrays.fill(row, -1);

      System.out.println("Minimum number of multiplications is " + MatrixCh
    }
}

// This code is contributed by avanitrachhadiya2155
```

Output

```
Minimum number of multiplications is 18
```

Time Complexity: $O(N^3)$
Auxiliary Space: $O(N^2)$ ignoring recursion stack space

Dynamic Programming Solution for Matrix Chain Multiplication using Tabulation (Iterative Approach):

In iterative approach, we initially need to find the number of multiplications required to multiply two adjacent matrices. We can use these values to find the minimum multiplication required for matrices in a range of length 3 and further use those values for ranges with higher lengths.

Build on the answer in this manner till the range becomes [0, N-1].

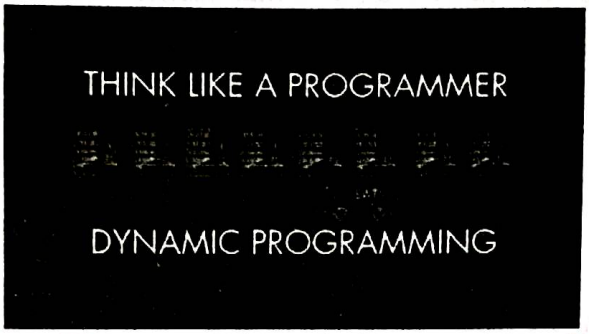
Follow the steps mentioned below to implement the idea:

- Iterate from **l = 2 to N-1** which denotes the length of the range:
  - Iterate from **i = 0 to N-1**:
    - Find the right end of the range (j) having **l** matrices.
    - Iterate from **k = i+1 to j** which denotes the point of partition.
      - Multiply the matrices in range **(i, k)** and **(k, j)**.
      - This will create two matrices with dimensions **arr[i-1]*arr[k]** and **arr[k]*arr[j]**.

Chetan More Follow
Aug 26, 2019 · 8 min read · Listen

Save

Matrix Chain Multiplication using Dynamic Programming



First of all What is Dynamic Programming ??

Dynamic programming is a method for solving optimization problems.
It is algorithm technique to solve a complex and overlapping sub-problems.
Compute the solutions to the sub-problems once and store the solutions in a table, so that they can be reused (repeatedly) later.

Dynamic programming is more efficient then other algorithm methods like as Greedy method, Divide and Conquer method, Recursion method, etc....

Why ? the Dynamic programming needed ??

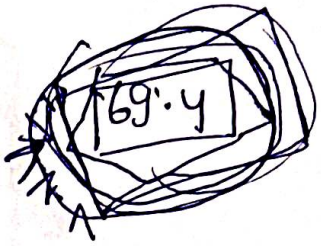
The real time many of problems are not solve using simple and traditional approach methods. like as coin change problem , knapsack problem, Fibonacci sequence generating , complex matrix multiplication....To solve using Iterative formula, tedious method , repetition again and again it become a more time consuming and foolish. some of the problem it should be necessary to divide a sub problems and compute its again and again to solve a such kind of problems and give the optimal solution , effective solution the Dynamic programming is needed...

Basic Features of Dynamic pr

Get all the possible solution a... ...id optimal solution.

176 2

```
{
// length[dims] = n + 1
n = dims.length - 1;
// m[i,j] = Minimum number of scalar multiplications(i.e., cost)
// needed to compute the matrix A[i]A[i+1]...A[j] = A[i..j]
// The cost is zero when multiplying one matrix
for (i = 1; i <= n; i++)
m[i, i] = 0;

for (len = 2; len <= n; len++){
// Subsequence lengths
for (i = 1; i <= n - len + 1; i++) {
j = i + len - 1;
m[i, j] = MAXINT;
for (k = i; k <= j - 1; k++) {
cost = m[i, k] + m[k+1, j] + dims[i-1]*dims[k]*dims[j];
if (cost < m[i, j]) {
m[i, j] = cost;
s[i, j] = k;
// Index of the subsequence split that achieved minimal cost
}
}
}
}
}
```



Example of Matrix Chain Multiplication

Example: We are given the sequence {4, 10, 3, 12, 20, and 7}. The matrices have size 4 x 10, 10 x 3, 3 x 12, 12 x 20, 20 x 7. We need to compute M [i,j], 0 ≤ i, j≤ 5. We know M [i, i] = 0 for all i.

x r matrix,the resulting matrix C is a p x r matrix. The time to compute C is dominated by the number of scalar multiplications is pqr. we shall express costs in terms of the number of scalar multiplications.For example, if we have three matrices (A1,A2,A3) and its cost is (10x100),(100x5),(5x500) respectively. so we can calculate the cost of scalar multiplication is 10*100*5=5000 if ((A1A2)A3), 10*5*500=25000 if (A1(A2A3)), and so on cost calculation. ***Note that in the matrix-chain multiplication problem, we are not actually multiplying matrices. Our goal is only to determine an order for multiplying matrices that has the lowest cost.***that is here is minimum cost is 5000 for above example .So problem is we can perform a many time of cost multiplication and repeatedly the calculation is performing. so this general method is very time consuming and tedious.So we can apply *dynamic programming* for solve this kind of problem.

when we used the Dynamic programming technique we shall follow some steps.

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution.
4. Construct an optimal solution from computed information.

Dynamic Programming

we have matrices of any of order. our goal is find optimal cost multiplication of matrices.when we solve the this kind of problem using DP step 2 we can get

→ $m[i , j] = min \{ m[i , k] + m[i+k , j] + pi-1*pk*pj \}$ *if* $i < j$.... *where p is dimension of matrix ,* $i \leq k < j$ .....

The basic algorithm of matrix chain multiplication:-

```
// Matrix A[i] has dimension dims[i-1] x dims[i] for i = 1..n
MatrixChainMultiplication(int dims[])
```

- Work on principal of optimality.
- Define sub-parts and solve them using recursively.
- Less space complexity But more Time complexity.
- Dynamic programming saves us from having to recompute previously calculated sub-solutions.
- Difficult to understanding.

Let's Discuss a matrix chain multiplication problem using Dynamic Programming :-

We are covered a many of the real world problems.In our day to day life when we do making coin change, robotics world, aircraft, mathematical problems like Fibonacci sequence, simple matrix multiplication of more then two matrices and its multiplication possibility is many more so in that get the best and optimal solution. NOW we can look about one problem that is ***MATRIX CHAIN MULTIPLICATION PROBLEM.***

Suppose, We are given a sequence (chain) ***(A1, A2.....An)*** of n matrices to be multiplied, and we wish to compute the product ***(A1A2.....An)***.We can evaluate the above expression using the standard algorithm for multiplying pairs of matrices as a subroutine once we have parenthesized it to resolve all ambiguities in how the matrices are multiplied together. Matrix multiplication is associative, and so all parenthesizations yield the same product. For example, if the chain of matrices is ***(A1, A2, A3, A4)*** then we can fully parenthesize the product ***(A1A2A3A4)*** in five distinct ways:

1:-(A1(A2(A3A4))) ,

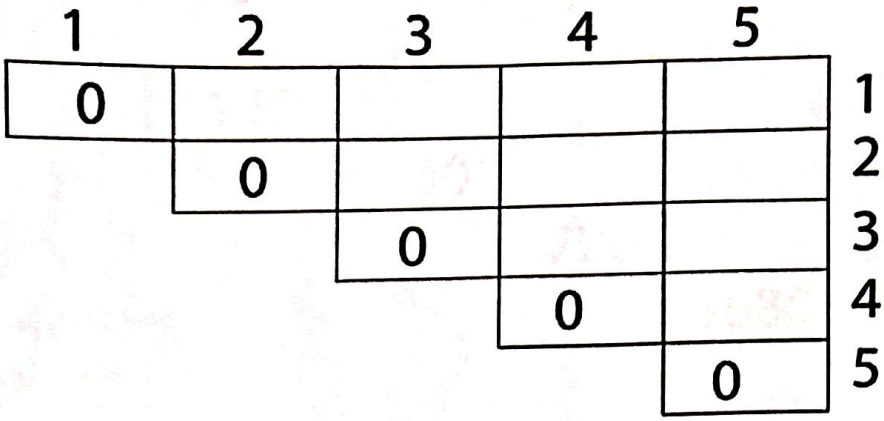
2:-(A1((A2A3)A4)),

3:- ((A1A2)(A3A4)),

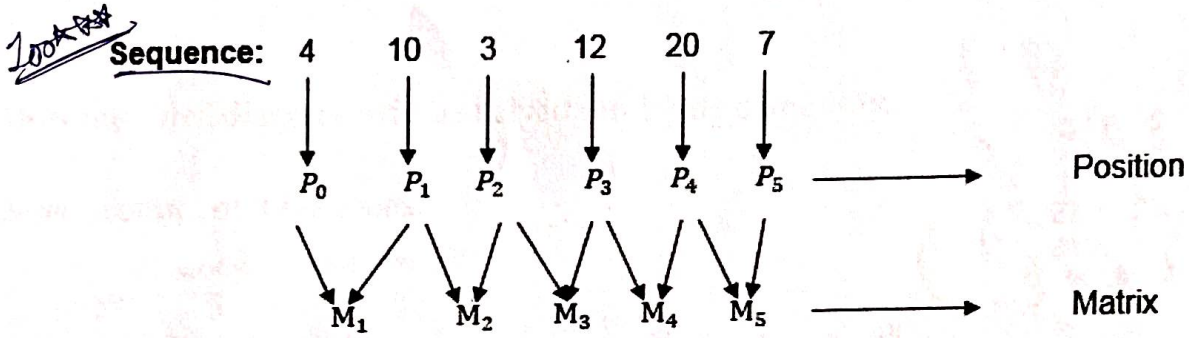
4:-((A1(A2A3))A4),

5:-(((A1A2)A3)A4) .

We can multiply two matrices A and B only if they are compatible. the number of columns of A must equal the number of rows of B. If A is a p x q matrix and B is a q



Let us proceed with working away from the diagonal. We compute the optimal solution for the product of 2 matrices.

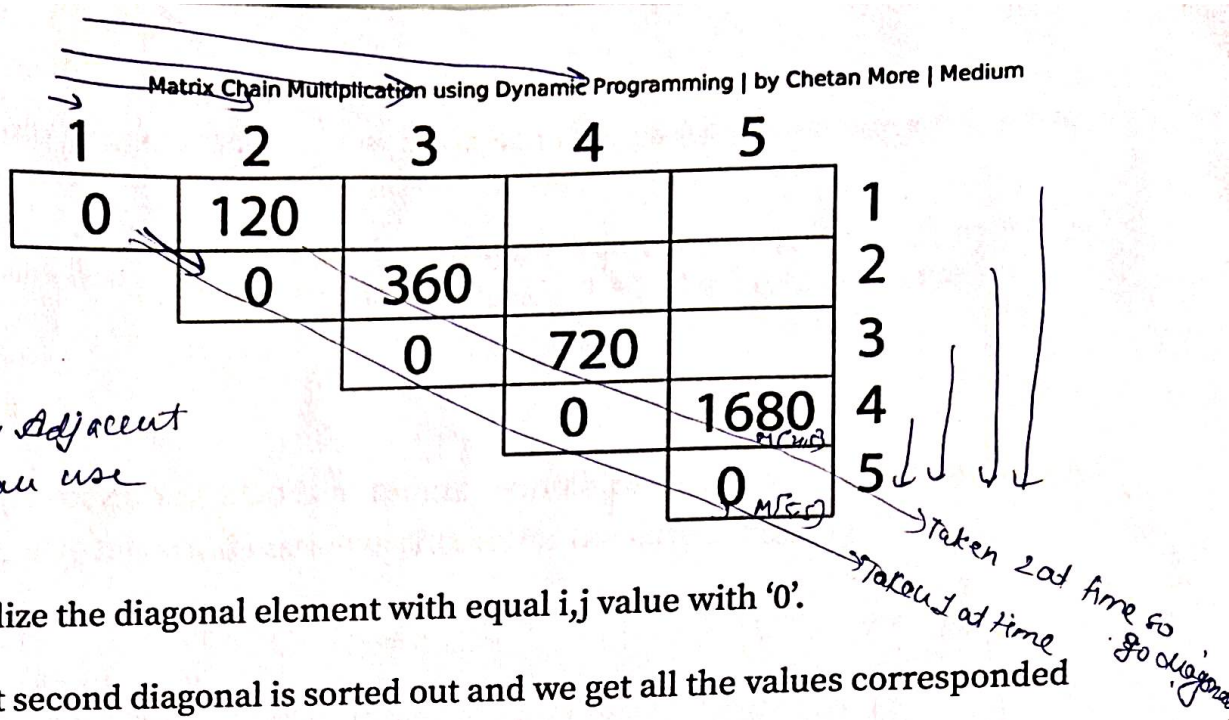


In Dynamic Programming, initialization of every method done by '0'.So we initialize it by '0'.It will sort out diagonally.

We have to sort out all the combination but the minimum output combination is taken into consideration.

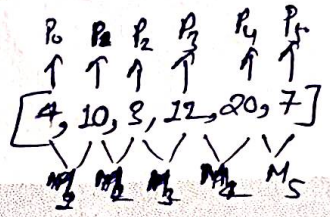
Calculation of Product of 2 matrices:

```
1. m (1,2) = m1  x m2
           = 4 x 10 x  10 x 3
           = 4 x 10 x 3 = 120

2. m (2, 3) = m2 x m3
            = 10 x 3  x  3 x 12
            = 10 x 3 x 12 = 360

3. m (3, 4) = m3 x m4
            = 3 x 12  x  12 x 20
            = 3 x 12 x 20 = 720

4. m (4,5) = m4 x m5
           = 12 x 20  x  20 x 7
           = 12 x 20 x 7 = 1680
```



- We initialize the diagonal element with equal i,j value with '0'.
- After that second diagonal is sorted out and we get all the values corresponded to it

Now the third diagonal will be solved out in the same way.



Now product of 3 matrices:

Should be continous only

```
M [1, 3] = M1 M2 M3
```

→ includes $P_0\ P_1\ P_2\ P_3$

1. There are two cases by which we can solve this multiplication: (M1 x M2) + M3, M1+ (M2x M3)
2. After solving both cases we choose the case in which minimum output is there.

$$M[1,3] = \min \begin{cases} M[1,2] + M[3,3] + p_0 p_2 p_3 = 120 + 0 + 4.3.12 = 264 \\ M[1,1] + M[2,3] + p_0 p_1 p_3 = 0 + 360 + 4.10.12 = 840 \end{cases}$$

(B) (P2)

M [1, 3] =264

As Comparing both output 264 is minimum in both cases so we insert 264 in table and (M1 x M2) + M3 this combination is chosen for the output making.

```
M [2, 4] = M2 M3 M4
```

1. There are two cases by which we can solve this multiplication: (M2x M3)+M4, M2+(M3 x M4)

2. After solving both cases we choose the case in which minimum output is there.

$$M[2,4] = \min \begin{cases} M[2,3] + M[4,4] + p_1p_3p_4 = 360 + 0 + 10.12.20 = 2760 \\ M[2,2] + M[3,4] + p_1p_2p_4 = 0 + 720 + 10.3.20 = 1320 \end{cases}$$

M [2, 4] = 1320

As Comparing both output 1320 is minimum in both cases so we insert 1320 in table and M2+(M3 x M4) this combination is chosen for the output making.

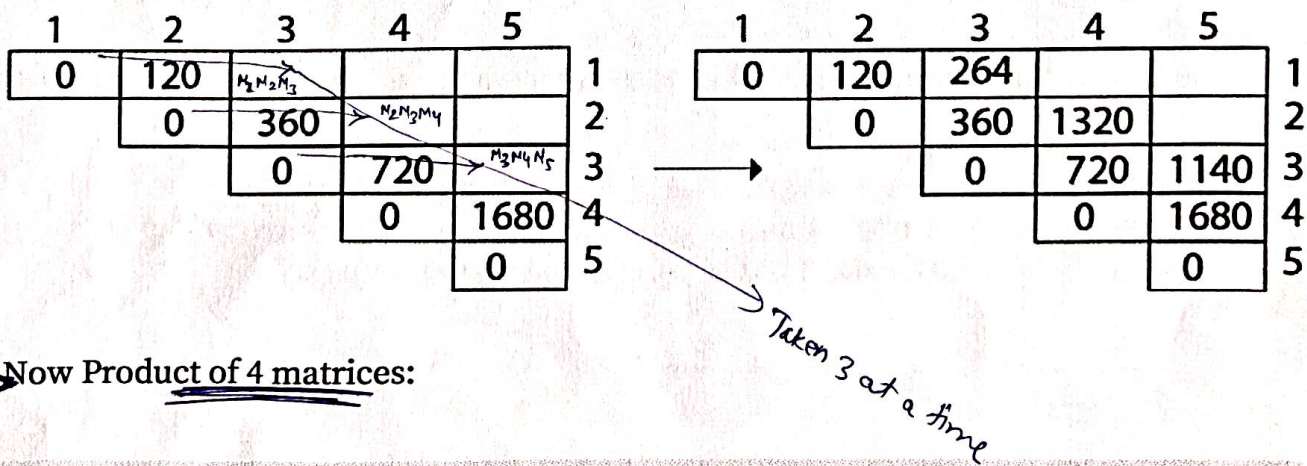
```
M [3, 5] = M3  M4  M5
```

1. There are two cases by which we can solve this multiplication: (M3 x M4) + M5, M3+ (M4xM5)
2. After solving both cases we choose the case in which minimum output is there.

$$M[3,5] = \min \begin{cases} M[3,4] + M[5,5] + p_2p_4p_5 = 720 + 0 + 3.20.7 = 1140 \\ M[3,3] + M[4,5] + p_2p_3p_5 = 0 + 1680 + 3.12.7 = 1932 \end{cases}$$

```
M [3, 5] = 1140
```

As Comparing both output 1140 is minimum in both cases so we insert 1140 in table and (M3 x M4) + M5this combination is chosen for the output making.



Now Product of 4 matrices:

```
M [1, 4] = M1  M2 M3 M4
```

Final Output is:

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080	1344	1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

So we can get the optimal solution of matrices multiplication....

SO it is end of our topic...👍🙏🙏🙏🙏👍

If you like my article click on *clap icon*👏, and *share* & *follow me*.....

THANK YOU!!!!!

References:-

Book:- Introduction to Algorithms Third Edition By Thomas H.cormen, Charles E.Leiserson, Ronald L. Rivest and Clifford Stein, PHI..

ON google wikipedia....

As comparing the output of different cases then '1350' is minimum output, so we insert 1350 in the table and M2 x(M3 x M4xM5)combination is taken out in output making.

1	2	3	4	5	
0	120	264			1
	0	360	1320		2
		0	720	1140	3
			0	1680	4
				0	5

→

1	2	3	4	5	
0	120	264	1080		1
	0	360	1320	1350	2
		0	720	1140	3
			0	1680	4
				0	5

Now Product of 5 matrices:

```
M [1, 5] = M1  M2 M3 M4 M5
```

There are five cases by which we can solve this multiplication:

1. (M1 x M2 xM3 x M4)x M5
2. M1 x(M2 xM3 x M4 xM5)
3. (M1 x M2 xM3)x M4 xM5
4. M1 x M2x(M3 x M4 xM5)

After solving these cases we choose the case in which minimum output is there

$$M[1,5] = \min\begin{cases} M[1,4] + M[5,5] + p_0p_4p_5 = 1080 + 0 + 4.20.7 = 1544 \\ M[1,3] + M[4,5] + p_0p_3p_5 = 264 + 1680 + 4.12.7 = 2016 \\ M[1,2] + M[3,5] + p_0p_2p_5 = 120 + 1140 + 4.3.7 = 1344 \\ M[1,1] + M[2,5] + p_0p_1p_5 = 0 + 1350 + 4.10.7 = 1630 \end{cases}$$

```
M [1, 5] = 1344
```

As comparing the output of different cases then '1344' is minimum output, so we insert 1344 in the table and M1 x M2 x(M3 x M4 x M5)combination is taken out in output making.

There are three cases by which we can solve this multiplication:

1. (M1 x M2 x M3) M4
2. M1 x(M2 x M3 x M4)
3. (M1 xM2) x (M3 x M4)

After solving these cases we choose the case in which minimum output is there

$$M[1,4] = \min \begin{cases} M[1,3] + M[4,4] + p_0p_3p_4 = 264 + 0 + 4.12.20 = 1224 \\ M[1,2] + M[3,4] + p_0p_2p_4 = 120 + 720 + 4.3.20 = 1080 \\ M[1,1] + M[2,4] + p_0p_1p_4 = 0 + 1320 + 4.10.20 = 2120 \end{cases}$$

M [1, 4] =1080

As comparing the output of different cases then '1080' is minimum output, so we insert 1080 in the table and (M1 xM2) x (M3 x M4) combination is taken out in output making,

```
M [2, 5] = M2 M3 M4 M5
```

There are three cases by which we can solve this multiplication:

1. (M2 x M3 x M4)x M5
2. M2 x(M3 x M4 x M5)
3. (M2 x M3)x (M4 x M5)

After solving these cases we choose the case in which minimum output is there

$$M[2,5] = \min \begin{cases} M[2,4] + M[5,5] + p_1p_4p_5 = 1320 + 0 + 10.20.7 = 2720 \\ M[2,3] + M[4,5] + p_1p_3p_5 = 360 + 1680 + 10.12.7 = 2880 \\ M[2,2] + M[3,5] + p_1p_2p_5 = 0 + 1140 + 10.3.7 = 1350 \end{cases}$$

```
M [2, 5] = 1350
```