



# **East West University**

## **Project Report CSE-246, Algorithm Submitted by-**

Name: Sajib Khan

Id no: 2020-1-60-186

Section: 2

**Submitted to-**  
**Jesan Ahammed Ovi,**  
Senior lecturer,  
Department of Computer Science & Engineering  
EastWest University

## 1. Problem Statement:

Matrix chain multiplication problems determine the optimal sequence for performing a series of operations. Though it has optimal substructure properties then we should take dynamic programming (DP) approach to solve this.

This problem, like other dynamic programming problems involves determining a structure (in this case, a parenthesization). We want to break the problem into subproblems, whose solutions can be combined to solve the global problem. As is common to any DP solution, we need to find some way to break the problem into smaller subproblems, and we need to determine a recursive formulation, which represents the optimum solution to each problem in terms of solutions to the subproblems

Matrix multiplication is an associative but not a commutative operation. This means that we are free to parenthesize the above multiplication however we like, but we are not free to rearrange the order of the matrices. Also recall that when two (nonsquare) matrices are being multiplied, there are restrictions on the dimensions. This algorithm does not perform the multiplications, it just determines the best order in which to perform the multiplications and the total number of operations.

## ii. System Requirements:

**Processor:** AMD Ryzen 5 3400G with Radeon Vega Graphics 3.70 GHz

**RAM:** 8GB

**Operating system:** Windows 10 Pro

**IDE:** CodeBlocks

## iii. System Design:

### **Algorithms:**

Our main motive is just to decide the order no need to actually perform the multiplication.

### **For example:**

*We have 4 matrix call*

*A B C D*

*Then possible multiplication is*

*(A)(BCD)*

*(AB)(CD)*

*(ABC)(D)*

*(A)(BC)(D).....*

*Considering two matrixes,*

$A=2*3$  and  $B=3*4$

Then cost is  $= 2*3*4$

### Our Naive solution:

- ❖ Place parentheses at all possible places
- ❖ Calculate the cost for each placement
- ❖ Return the minimum value

### Recursive Algorithm:

Assume that someone tells us the position of the last product, say  $k$ . Then we have to compute recursively the best way to multiply the chain from  $i$  to  $k$ , and from  $k + 1$  to  $j$ , and add the cost of the final product.

This means that  $m(i, j) = m(i, k) + m(k + 1, j) + p_{i-1} \cdot p_k \cdot p_j$

- If no one tells us  $k$ , then we have to try all possible values of  $k$  and pick the best solution.

### Recursive formula:

$$m(i, j) = \begin{cases} 0 & \text{If } i = j \\ \min_{i \leq k < j} \{m(i, k) + m(k + 1, j) + p_{i-1} \cdot p_k \cdot p_j\} & \text{If } i < j \end{cases}$$

### Pseudocode:

```
Algorithm RECURSIVE_MATRIX_CHAIN_ORDER(p, i, j)
// p is sequence of n matrices
if i == j then
    return 0
m[i, j] ← ∞

for j ← i to j - 1 then
    x ← RECURSIVE_MATRIX_CHAIN_ORDER(p, i, k) +
        RECURSIVE_MATRIX_CHAIN_ORDER(p, k + 1, j) +
        pi-1*pk*pj
    if q < m[i, j] then
        m[i, j] ← x
end
end
return m[i, j]
```

### Simulation:

Suppose,

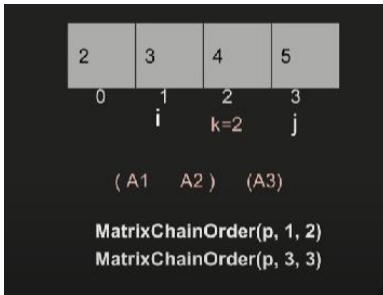
$N=4$ ,  $P[n]=\{2,3,4,5\}$

$MCM(p,1,n-1)$  //MCM means MATRIX\_CHAIN\_ORDER

$MCM(p,1,3)$  //A1 A2 A3

p	2	3	4	5
	0	1	2	3
		i		j

(When  $K=1$ )



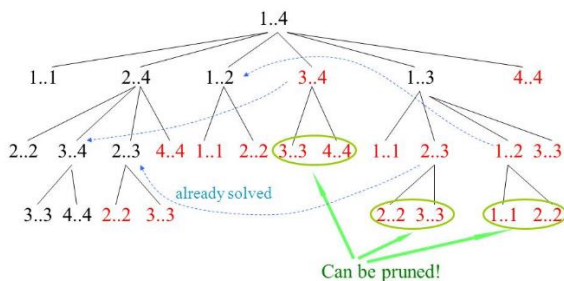
when  $k=2$

Now it will call recursively for the base case, and return the minimum cost.

### Recursion tree:

Assuming this MCM recursion tree,

### Matrix-Chain Recursion Tree



Though it has optimal substructure properties then we should take the dynamic programming (DP) approach to solve this.

**Run time: Exponential ( $2^n$ )**

### A dynamic programming algorithm:

To begin, let us assume that all we really want to know is the minimum cost, or minimum number of arithmetic operations needed to multiply out the matrices. If we are only multiplying two matrices, there is only one way to multiply them, so the minimum cost is the cost of doing this. In general, we can find the minimum cost using the following recursive algorithm:

- Take the sequence of matrices and separate it into two subsequences.
- Find the minimum cost of multiplying out each subsequence.
- Add these costs together and add in the cost of multiplying the two result matrices.
- Do this for each possible position at which the sequence of matrices can be split, and take the minimum over all of them.

For example, if we have four matrices  $ABCD$ , we compute the cost required to find each of  $(A)(BCD)$ ,  $(AB)(CD)$ , and  $(ABC)(D)$ , making recursive calls to find the minimum cost to compute  $ABC$ ,  $AB$ ,  $CD$ , and  $BCD$ . We then

choose the best one. Better still, this yields not only the minimum cost, but also demonstrates the best way of doing the multiplication: group it the way that yields the lowest total cost, and do the same for each factor.

1) Now, we memorize using table.

We'll have a table  $T[1..n][1..n]$  such that  $T[i][j]$  stores the solution to problem Matrix-CHAIN( $i, j$ ). Initially all entries will be set to -1.

```
FOR  $i = 1$  to  $n$  DO
  FOR  $j = i$  to  $n$  DO
     $T[i][j] = -1$ 
  OD
OD
```

2) The code for ( $i, j$ ) stays the same, except that it now uses the table. The first thing MCM( $p[], i, j$ ) does is to check the table to see if  $T[i][j]$  is already computed. If so, it returns it, otherwise, it computes it and writes it in the table. Below is the updated code.

```
MCM( $p[], i, j$ )
IF  $T[i][j] < \infty$  THEN return  $T[i][j]$ 
IF  $i = j$  THEN  $T[i][j] = 0$ , return 0
 $m = \infty$ 
FOR  $k = i$  to  $j - 1$  DO
 $q = \text{Matrix-chain}(p, i, k) + \text{Matrix-chain}(p, k + 1, j) + p_{i-1} \cdot p_k \cdot p_j$ 
IF  $q < m$  THEN  $m = q$ 
OD
 $T[i][j] = m$ 
return  $m$ 
END
return MCM( $p, 1, n$ )
```

3) The table will prevent a subproblem MATRIX-CHAIN( $i, j$ ) to be computed more than once.

4) DP pseudocode

```
FOR  $i = 1$  to  $n$  DO
   $T[i][i] = 0$ 
OD
FOR  $l = 1$  to  $n - 1$  DO
  FOR  $i = 1$  to  $n - l$  DO
     $j = i + l$ 
     $T[i][j] = 99999$ 
    FOR  $k = i$  to  $j - 1$  DO
       $q = T[i][k] + T[k + 1][j] + p_{i-1} \cdot p_k \cdot p_j$ 
      IF  $q < T[i][j]$  THEN  $T[i][j] = q$ 
    OD
  OD
OD
```

## Simulation:

Top -down thinking

Key is that  $m(i, j)$  only depends on  $m(i, k)$  and  $m(k + 1, j)$  where  $i \leq k < j$

⇒ if we have computed them, we can compute  $m(i, j)$

⇒ We can easily compute  $m(i, i)$  for all  $1 \leq i \leq n$  ( $m(i, i) = 0$ )

⇒ Then we can easily compute  $m(i, i + 1)$  for all  $1 \leq i \leq n - 1$   $m(i, i + 1) = m(i, i) + m(i + 1, i + 1) + p_{i-1} \cdot p_i \cdot p_{i+1}$

⇒ Then we can compute  $m(i, i + 2)$  for all  $1 \leq i \leq n - 2$   $m(i, i + 2) = \min\{m(i, i) + m(i + 1, i + 2) + p_{i-1} \cdot p_i \cdot p_{i+2}, m(i, i + 1) + m(i + 2, i + 2) + p_{i-1} \cdot p_{i+1} \cdot p_{i+2}\} \dots \dots \dots$

⇒ Until we compute  $m(1, n)$

⇒ Computation order:

$j \rightarrow$

	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2		1	2	3	4	5	6
3			1	2	3	4	5
4				1	2	3	4
5					1	2	3
6						1	2
7							1

$i \downarrow$

– Computation order

## Analysis:

$O(n^2)$  entries,  $O(n)$  time to compute each ⇒  $O(n^3)$

#### iv. Implementation:

```
cout << "Minimum number of multiplications is "  
      << MCM(arr, 1, n-1);
```

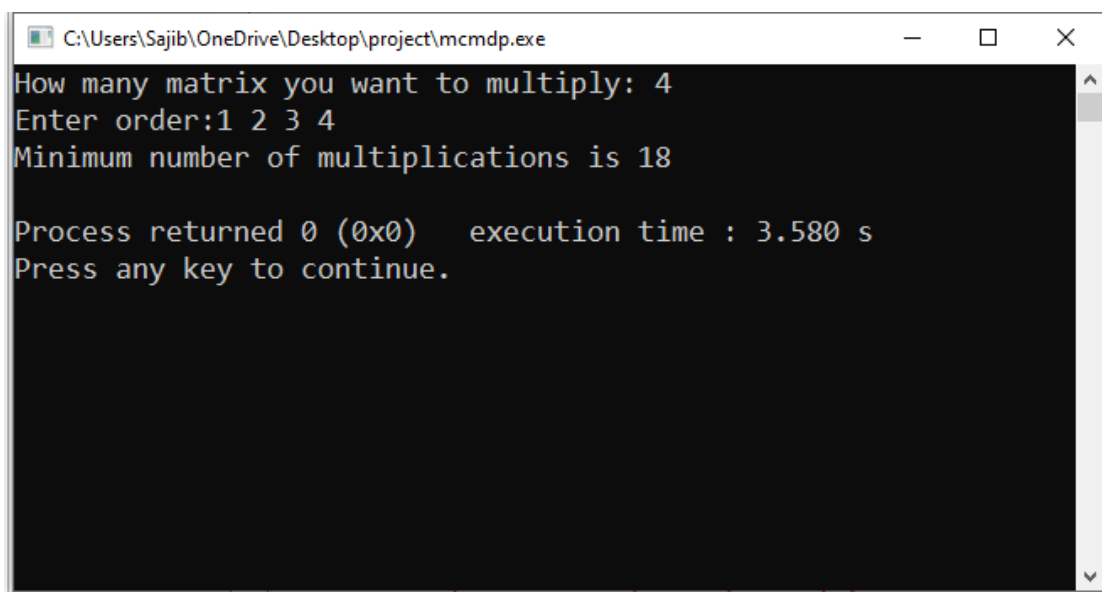
calling MCM function by passing (an array, starting index, stop index) for output

MCM FUNCTION:

```
int MCM(int p[], int i, int j)  
{  
    if (i == j)  
    {  
        return 0;           Base case  
    }  
    if (dp[i][j] != -1)  
    {  
        return dp[i][j];    checking index already  
                             visited or not  
    }  
    dp[i][j] = 999999;  
    for (int k = i; k < j; k++)  
    {  
        dp[i][j] = min(  
            dp[i][j], MCM(p, i, k)  
                + MCM(p, k + 1, j)  
                + p[i - 1] * p[k] * p[j]);  
    }  
    return dp[i][j]; ➡ Returning result  
}
```

expecting minimum value by calling recursively and storing it in dp[][] array

#### v. Testing Results:



```
C:\Users\Sajib\OneDrive\Desktop\project\mcmdp.exe  
How many matrix you want to multiply: 4  
Enter order:1 2 3 4  
Minimum number of multiplications is 18  
  
Process returned 0 (0x0)   execution time : 3.580 s  
Press any key to continue.
```

**INPUT:**

First, we have to enter how many matrices we have for multiplication,

Then we have to enter the order

**OUTPUT:**

And our program will show you minimum number of multiplaciton

**vi. Future Scope:**

- This kind of problem is important in compiler design for code operation
- what is seen on the computer screen is a 2D picture representing a point in 3D space. Matrix multiplication allows you to convert between 2D and 3D worlds.

**Thank you**