East West University

CSE246

Section: 2

PROJECT

Matrix Chain Multiplication Using DP

<u>SUBMIT TO</u>

Redwan Ahmed Rizvee

Lecturer

Department of Computer Science and Engineering

East West University

Prepared by

Name	ID
Md.Al Amin	2019-1-65-003
Arpon Bhattachargee	2019-3-60-061
Md.Iftekhar Uddin	2019-3-60-061

Date of Submission:2 September 2022

Contents

1. Problem statement:

Given a sequence of matrices, find the most effective way to multiply these matrices together. However, the problem is not to perform multiplication, but to decide in which order the multiplication should be performed. Since matrix multiplication is collaborative, there are many options for multiplying a chain of matrices. In other words, no matter how we bracket the product, the results will be the same. However, the order in which we shape the product in parentheses affects the number or efficiency of simple mathematical operations required to calculate the product.

2. System design:

Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping subproblems and optimal substructure properties. The idea is to divide the problem into a given matrix, in a set of sub-problems related to that group to get the lowest total cost which is the basic idea of DP.

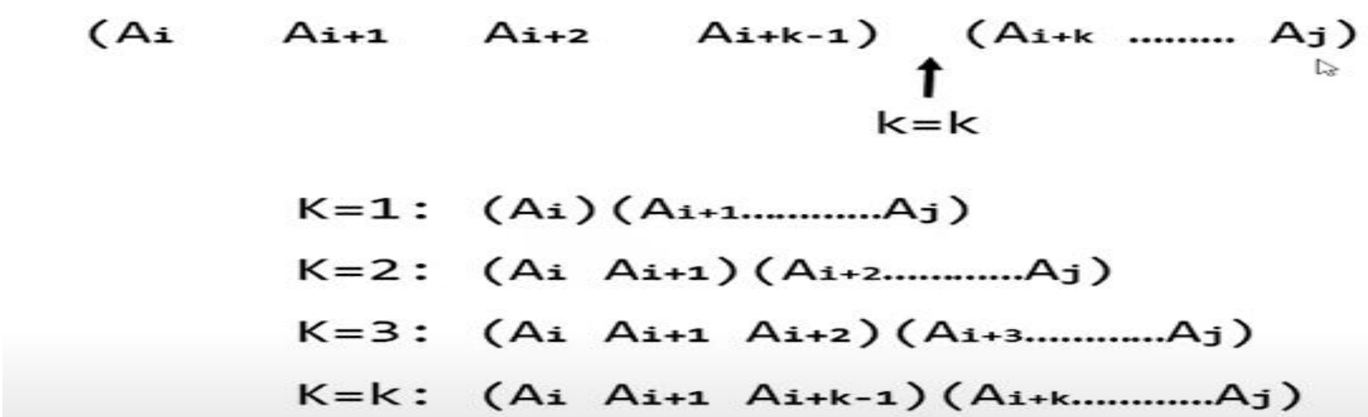
Matrix chain multiplication is an optimization problem to find the most effective way to multiply a given sequence of matrices. The problem is not to perform the multiplication but to determine the order of the matrix multiplication involved. For this, we need to take the dimension of the matrix as input and as a result, we need to multiply the minimum number of times to multiply the chain and parenthesis.

The complexity during matrix chain multiplication will be O (n3).

3. Basic Rules:

- Dimension of multiplication matrix: A (P $_x$ Q), B (Q $_x$ R) = [P R]
- Scalar Multiplication Required multiplying:A (P $_x$ Q), B (Q $_x$ R) =[P $_x$ Q $_x$ R]
- Dimension of a matrix in chain multiplication: Ai = [P $_{i-1}$ P $_i$]

4. Dividing Matrix Chain:



Where, K= 1 to j-i

Example: n=4---array size

i=4 j=7: A_4 A_5 A_6 A_7

K= 1 to n-1

= 1 to j-i+1 [n=j-i]

5. Scalar Multiplication Required multiplying

- $A\ (P_x Q), B(Q_x R)=[P_x Q_x R]=(A_i \times A_{i+1\ x} \ldots_x A_{i+k-1})_x(A_{i+k\ x}.._x A_j)$

Let, $M = (A_i \times A_{i+1\ x} \ldots_x A_{i+k-1})\ _x\ (A_{i+k\ x}\ldots_x A_j)$

Here, $X=(A_i \times A_{i+1\ x} \ldots_x A_{i+k-1})$

$Y= (A_{i+k\ x}\ldots\ldots_x A_j)$

Multiplication required for forming M: MCM(i, j)

Multiplication required for forming X: MCM(i, i+k+1)

Multiplication required for forming Y: MCM(i+k, j)

Dimension of X: [P_{i-1} P_{i+k-1}]

Dimension of Y: [P_{i+k-1} P_j]

Multiplication required for forming XY: MCM (i, i+k+1) + MCM(i+k, j) + $P_{i-1\ x}\ P_{i+k-1\ x}\ P_j$

6.Pseudo Code

```
Matrix_Chain( par_i,  par_j)
   if ← par_i  is  equal par_j   return 0;
   if ← m[i][j] equal to 0
      do Min  equal to infinity
      do index  equal to 1
      for  k← 1 to j-i
          do x ← mcm(i,i+k-1)+mcm(i+k,j)+p[i-1]*p[i+k-1]*p[j];
         if x<M
                do Min←x;
                 do index←k;
      m[i][j]←Min
       s[i][j]←index
return m
```

```
parenthesis (par_i, par_j)
   if ← par_i  is  equal to par_j   return
    do k← S[par_i] [par_j]
    do  ← parenthesis ( par_i, par_i+k-1)
    do  ← parenthesis (par_i+k-1, par_j)
```

Time complexity: O (N^3)

7. Optimal Cost Simulation

- A1 (30 x 35)
- A2 (35 x 15)
- A3 (15 x 5)
- A4 (5 x 10)

P	30	35	15	5	10
	0	1	2	3	4

m	1	2	3	4
1				
2				
3				
4				

Optimal Cost for multiplying **A_1 to A_4**

Optimal Cost for multiplying **A_1 to A_2**

Optimal Cost for multiplying **A_3 to A_4**

m[i][j] = Optimal Cost for multiplying **A_i to A_j**

- A1 (30 x 35)
- A2 (35 x 15)
- A3 (15 x 5)
- A4 (5 x 10)

P	30	35	15	5	10
	0	1	2	3	4

m	1	2	3	4
1	0			
2		0		
3			0	
4				0

s	1	2	3	4
1	0			
2		0		
3			0	
4				0

For Parenthesis

K=1: $m[1][1] + m[2][2] + P_0 \times P_1 \times P_2$

[(A_1) (A_2)] **A_3** A_4

↑ k=1

- A1 (30 x 35)
- A2 (35 x 15)
- A3 (15 x 5)
- A4 (5 x 10)

P	30	35	15	5	10
	0	1	2	3	4

m	1	2	3	4
1	0	15750	7875	9375
2		0	2625	4375
3			0	750
4				0

s	1	2	3	4
1	0	1	1	
2		0	1	2
3			0	1
4				0

For Parenthesis

[A_1 A_2 **A_3** A_4]

↑ k=3

K=1: $m[1][1] + m[2][4] + P_0 \times P_1 \times P_4 = 14875$

K=2: $m[1][2] + m[3][4] + P_0 \times P_2 \times P_4 = 20820$

K=3: $m[1][3] + m[4][4] + P_0 \times P_3 \times P_4 = 9375$

K=k: $m[i][i+k-1] + m[i+k][j] + P_{i-1} \times P_{i+k-1} \times P_j$

8. Optimal Cost Implement

Declare necessary header file, variable, and array:

```
#include<bits/stdc++.h>
using namespace std;

#define INF INT_MAX

int n;//number of matrix
int m [101][101];//use for storing multiplication value
int s [101][101];//use for parenthesis
int p[101];//dimension array: A
```

- Define an INT INT_MAX
- Declare a variable for the input total number of the matrix from the user.
- Declare a 2D array m [101][101] for storing the multiplication value.]. Indexing starts from 1, so array size is 101.
- Declare a 2d array s[101][101] for parenthesis.]. Indexing starts from 1, so array size is 101.
- Declare a 1D dimensional array of p[101]. Indexing starts from 1, so the array size is 101.

Input number of row and col and store:

```
int main()
{     cin >>n;//input number of matrix from user
    for (int i=1; i<=n; i++)
    {
        int row, col;// declare  row and col
        cin >>row>>col;// input row and col from user
        p[i-1]=row;//store number of row into dimension array, start from  index: Po
        p[i]=col; //store number of row into dimension array, start from  index: P1
```

- Input the total number of the matrices from the user.
- **For loop:** 1 to 1<=total number for the matrix
    a. Input number row and col from the user.
    b. store number of rows into dimension array, start from the index: P_0
    c. Store number of col into dimension array, start from the index: P_1

printing the dimension array:

```
cout<<"Print Dimension array"<<endl;
for (int i=0; i<=n; i++)
     cout<<p[i]<<" ";//print dimension array
     cout<<endl<<endl;
```

- For loop---- 0 to 0<= total number of a matrix, then the dimension array with store storing the value into array P[i].

Printing Store Matrix:

```
cout<<"Store Matrix: "<<endl;
  int result= mcm(1,n); //initialization MCM function int result variable
  for (int i=1; i<=n; i++)//print matrix
  {
      for (int j=1; j<=n; j++)
      {
          cout<<m[i][j]<<"     ";
      }
          cout<<endl;
  }     cout<<endl<<endl;
```

- MCM function initialization into result variable.
- Nested independent For Loop, use for printing store multiplication value.

```
cout<<"Total Multiplication Need:"<<result<<endl<<endl;
cout<<endl<<endl;
cout<<"Parenthesied:";
parenthesis(1,n);
```

- Print minimum operation of matrix chain multiplication, then call parenthesis function for prin the parenthesis.

MCM function:

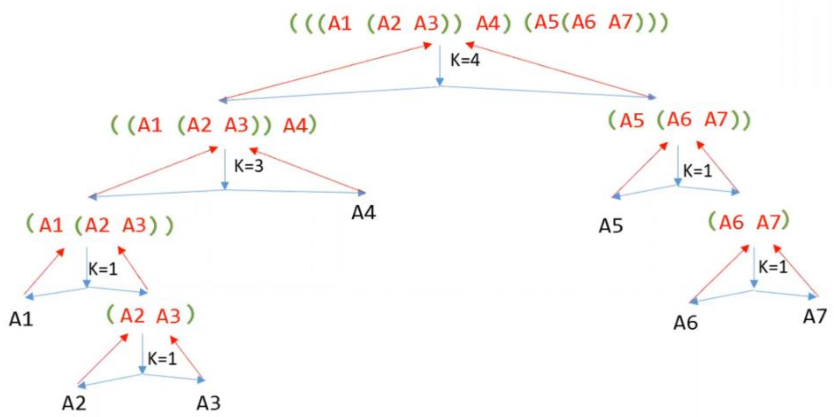
```
//calculate the total multiplication of given matrix index number
int MCM(int i, int j)//how many operation  i to j
{
    if (i==j)// Base case
    {
        return 0;
    }
    if(m[i][j]==0) //value not set yet
    {
        //value set
        int Min=INF;
        int index=1;
        for (int k=1; k<=j-i; k++)
        {
            int x=MCM(i,i+k-1)+MCM(i+k,j)+p[i-1]*p[i+k-1]*p[j];
             //left  sub chain multiplication: mcm(i,i+k-1)
              //left right sub chain multiplication: mcm(i+k,j)
               //left and right sub chain multiplication:p[i-1]*p[i+k-1]*p[j]
            if (x<Min)//check minimum
            {
                Min=x;//minimum value
                index=k;//minimum index
            }
        }
        m[i][j]=Min;//minimum value
        s[i][j]=index;//minimum index
    }
    return m[i][j]; // value always return from storage matrix

}
```

- Set a base case
- Set all value is zero in the storage matrix
- Calculate the left sub-chain multiplication: MCM(i,i+k-1)
- Calculate the left sub-chain multiplication: MCM(i+k,j)
- T calculate: **min**= MCM(i,i+k-1)+MCM(i+k,j)+p[i-1]*p[i+k-1]*p[j]
- Finally, return the storage matrix.

9.Parenthesization:

- First we see the process of parenthesization for a large chain (A_i ………… A_j). Here we will take **random value of k** at each step.But in algorithm K will be obtained from **s[i][j]**



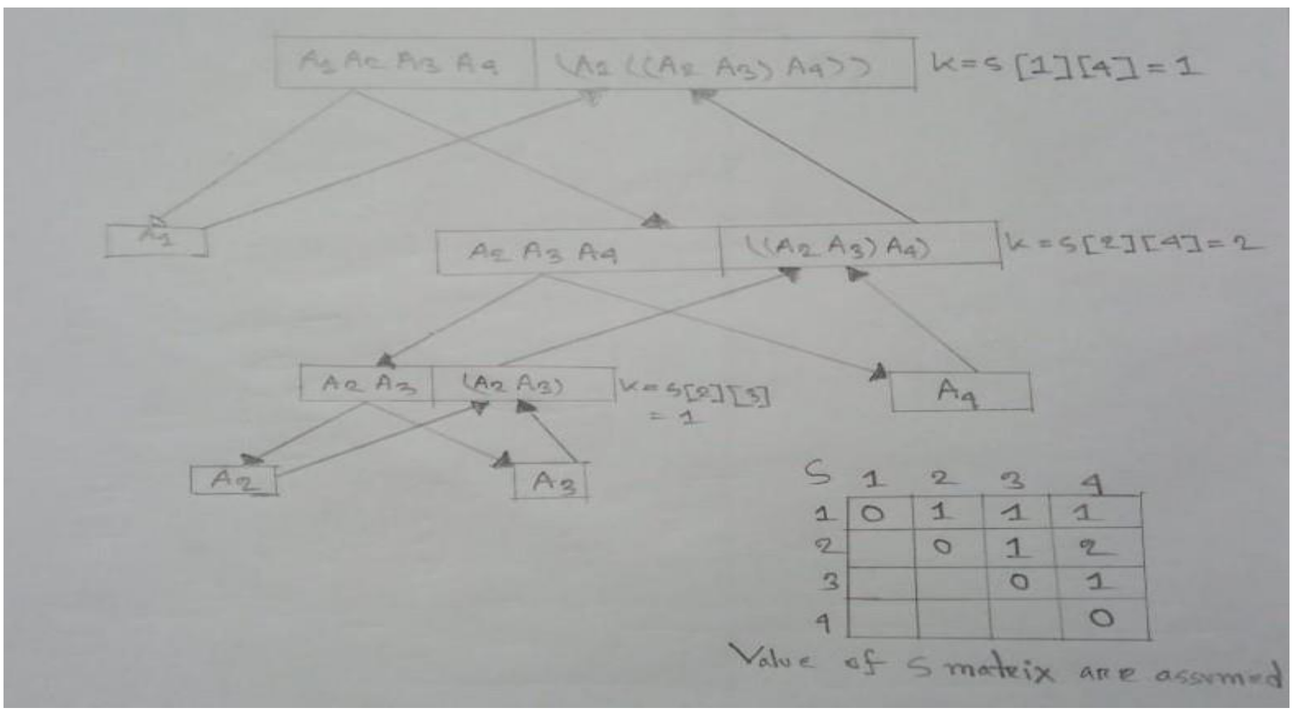
The deeper the repetition, the more unnecessary repetition occurs. The concept is used to memorize. Now each time we calculate the minimum cost required to multiply a particular follower and save it for future calculations. If we are asked to recalculate it, just give the saved answer and do not recalculate it.

calculate dimensions of a matrix: **Size of a matrix = number of rows × number of columns.**

It can be read as the size of a matrix and is equal to the number of rows “by” the number of columns.

10. Logic of Parenthesization:

- Keep dividing a chain into 2 sub chains based on the value of **k** until a chain contains a single element
- The **k** value is obtained for breaking a chain: (A_i .... A_j) from s[i][j]
- If we want to divide (A_i .... A_j) then first we need the value of s[i][j]. We consider s[i][j] = k
- The sub chains will be (A_i .... A_{i+k-1}) and (A_{i+k} .... A_j)
- For example if we want to divide the chain (A_3 .... A_9) for parenthesization first we need s[3][9]
- Let, s[3][9] = 3 then the sub chains will be (A_3 A_4 A_5) and (A_6 A_7 A_8 A_9)
- When both of the sub chains are visited the sub chains are merged and enclosed with parenthesis
- Left sub chain is merged with (at beginning and right sub chain is merged with) at end



11. Parenthesization Implementation

```
void parenthesis(int i, int j)
{

    if (i==j)//Base case
    {
        cout<<"A"<<i;
        return ;
    }
    int k =s[i][j];

    cout<<"(";
    parenthesis(i, i+k-1);//Print: left sub-chain
    parenthesis(i+k, j);//Print: right sub-chain
    cout <<")";
}
```

- If the base case is true then print the matrix name, like as: A,B,C,D…..
- If the base case is not the same then, print the left sub-chain and right sub-chain.
    - left sub-chain: parenthesis(i, i+k-1)
    - left sub-chain: parenthesis(i+k, j)

12. Output :

```
3
10 30
30 5
5 60
Print Dimension array
10 30 5 60

Matrix
0 1500 4500
0 0 9000
0 0 0


Total Multiplication Need:4500



Parenthesied:((A1A2)A3)
Process returned 0 (0x0)   execution time : 19.867 s
Press any key to continue.
```

⇒Here we take 3 matrixes as input.

A:10 × 30

B:30 × 5

C: 5 × 60

So the output of a Minimum number of arithmetic operations: 4500

```
4
5 10
10 15
15 20
20 30
Print Dimension array
5 10 15 20 30

Matrix
0 750 2250 5250
0 0 3000 9000
0 0 0 9000
0 0 0 0


Total Multiplication Need:5250



Parenthesied:(((A1A2)A3)A4)
Process returned 0 (0x0)   execution time : 17.974 s
Press any key to continue.
```

⇒Here we take 3 matrixes as input.

A:5 × 10

B:10 × 15

C: 15 × 20

D:20 x 30.

So the output of a Minimum number of arithmetic operations is 5250.

```
3
1 2
2 3
3 4
Print Dimension array
1 2 3 4

Matrix
0 6 18
0 0 24
0 0 0


Total Multiplication Need:18



Parenthesied:((A1A2)A3)
Process returned 0 (0x0)   execution time : 15.227 s
Press any key to continue.
```

⇒Here we take 3 matrixes as input.

A :1 × 2

B:2 × 3

C: 3 × 4

So the output of a Minimum number of arithmetic operations is 18.

```
4
40 20
20 30
30 10
10 30
Print Dimension array
40 20 30 10 30

Matrix
0 24000 14000 26000
0 0 6000 12000
0 0 0 9000
0 0 0 0


Total Multiplication Need:26000



Parenthesied:((A1(A2A3))A4)
Process returned 0 (0x0)   execution time : 20.037 s
Press any key to continue.
```

⇒Here we take 4 matrixes as input.

A :40 × 20

B:20 × 30

C: 30 × 10

D:10 x 30.

So the output of a Minimum number of arithmetic operations is 26000.

13. Future Scope:

In this matrix chain multiplication scheme, we calculate the minimum number of arithmetic operations. The problem of matrix chain multiplication is generalized to solve more abstract problems: a linear sequence of objects, a collaborative binary operation on those objects, and a way to calculate the cost of performing those operations on any two objects (plus all partial results), (probability tree) As well as calculating the least cost way to group objects to apply operations on all partial result sequences.

14. Source Code

```
#include<bits/stdc++.h>
using namespace std;

#define INF INT_MAX


int n;;//declare number of matrix
int m [101][101];//use for storing multiplication value
int s [101][101];//use for parenthesis
int p[101];//dimension array: A1,A2.......An



//calculate the total multiplication of given matrix index number
int mcm(int i, int j)
{
    if (i==j)//value not set yet
    {
        return 0;
    }
    if(m[i][j]==0) //value not set yet
    {
        int Min=INF;//value set
        int index=1;
        for (int k=1; k<=j-i; k++)
        {
            int x=mcm(i,i+k-1)+mcm(i+k,j)+p[i-1]*p[i+k-1]*p[j];
            if (x<Min)
```

```
            {
                Min=x;
                index=k;
            }
        }
        m[i][j]=Min;
        s[i][j]=index;
    }
    return m[i][j];

}

void parenthesis(int i, int j)
{

    if (i==j)
    {
        cout<<"A"<<i;
        return ;
    }
    int k =s[i][j];

    cout<<"(";
    parenthesis(i, i+k-1);
    parenthesis(i+k, j);
    cout <<")";
}
```

```
int main()
{

    cin >>n;//input number of matrix from user
    for (int i=1; i<=n; i++)
    {
        int row, col;// declare  row and col
        cin >>row>>col;// input row and col from user
        p[i-1]=row;//store number of row into dimension array, start from  index: Po
        p[i]=col;//store number of row into dimension array, start from  index: P1
    }

    cout<<"Print Dimension array"<<endl;
    for (int i=0; i<=n; i++)
        cout<<p[i]<<" ";//print dimension array
        cout<<endl<<endl;



  cout<<"Matrix"<<endl;
    int result= mcm(1,n);
    for (int i=1; i<=n; i++)//print matrix
    {
        for (int j=1; j<=n; j++)
        {
            cout<<m[i][j]<<" ";
        }
            cout<<endl;
    }    cout<<endl<<endl;
```

```
    cout<<"Total Multiplication Need:"<<result<<endl<<endl;
    cout<<endl<<endl;
    cout<<"Parenthesied:";
    parenthesis(1,n);
}
```