East West University

CSE246 (Section: 3)

PROJECT

Prepared for

Jesan Ahammed Ovi

Senior Lecturer,

Department of CSE

Prepared by

Name	ID
Mysha Maliha Priyanka	2020-1-60-230
K. M. Safin Kamal	2020-1-60-235

Date of Submission: 19th January 2022

Matrix Chain Multiplication Using DP

Problem statement:

Given a sequence of matrices, find the most efficient way to multiply these matrices together. However, the problem is not actually to perform the multiplications, rather you have to decide in which order the multiplications should be performed.

Since matrix multiplication is associative, there are so many options to multiply a chain of matrices. In other words, no matter how we parenthesize the product, the result will be the same. However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency.

The time complexity of the Matrix chain multiplication will be **O(n^3)**.

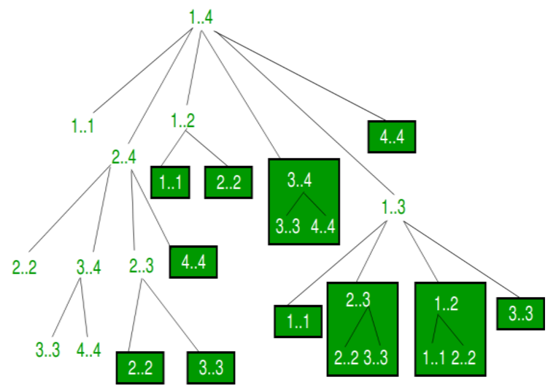
System Requirements:

Here we used Intel i5 7500 processor with 16GB DDR4 RAM. Our operating system is Windows 10 pro and we used Codeblocks IDE for the code. But people can also run this project in low end devices.

System design:

Matrix chain multiplication is an optimization problem that to find the most efficient way to multiply a given sequence of matrices. The problem is not actually to perform the multiplications but merely to decide the sequence of the matrix multiplications involved. For this we have to take dimensions of matrices as input and resultant the minimum number of multiplications needed to multiply the chain.

The idea is to break the problem into a set of related subproblems of that group the given matrix to yield the lowest total cost which is the main idea of DP.



As the recursion grows deeper, more and more unnecessary repetition occurs. The idea is to use memoization. Now each time we compute the minimum cost needed to multiply out a specific subsequence and save it for future computation. If we are ever asked to compute it again, simply give the saved answer and do not recompute it.

Implementation:

In the main function -

```
→x=y*2;
```

*There will be 2 values (row and column) for each matrix

```
→for(int i=1;i<=x;i++){
        cin>>ar[i];
}
```

*Putting row and column in ar[] array

```
→n=y+1;
   int arr[n];
for(int i=1;i<=x;i++)
        {
        if(i%2!=0 || i==x){
        arr[z]=ar[i];
        z++;
        }
```

*To put the row number of every matrix and the column number of last matrix for calculation purpose

In the matrixChain(int p[], int n) function -

```
→ for (int i=1; i<n; i++)
      dp[i][i] = 0;
```

*If matrix is same return 0. As, if same matrix comes the result must be zero. Here will be no computation. Like dp[1,1], dp[2,2],…… these will be always resultant in zero.

```
→for (int x=2; x<n; x++)
   {for (int i=1; i<n-x+1; i++)
      {int j = i+x-1;
         dp[i][j] = INT_MAX;
         for (int k=i; k < j; k++)
         {int q = dp[i][k] + dp[k+1][j] + p[i-1]*p[k]*p[j];
            if (q < dp[i][j])
            {dp[i][j] = q;
             bracket[i][j] = k;}
            }}
  char name = 'A';
  Parenthesis(1, n-1, n, (int *)bracket, name);
   return dp[1][n-1];
```

*Here first we initialize dp[i][j] = INT_MAX, which means the maximum integer value. Then from the loop we are getting the minimum required value. The loop will iterate from i to j-1. By this we can calculate differernt values of k. here k is the position of bracket. It should be mention that dp[][] will be a upper triangular matrix. That’s why we take value of x, i and j like that.

If the difference between i and j is equal to 1, there will be one value of k. and loop will be iterate only a single time and that is the result. Here as we initialize dp[i][j] with INT_MAX, minimum value will definitely be the calculated one.

If the difference between i and j is greater than 1, there will we more than 1 values of k. so the loop will be iterate for k times. Firstly, d[i][j] will be stored as INT_MAX and for the second part of min, will be calculated, then min value will be the calculated value. So d[i][j] will be stored newly with the calculated value. Then the k will be k++ and again calculate for k+1, then compare with the previous dp[i][j] with the newly calculated value with k+1, then stored the minimum value. The process will repeat until the loop iterates. Finally after the loop ends, dp[i][j] will be the minimum then. After that it returns in the main function. It also call the Parenthesis function to print the parameterization.

In the Parenthesis (int i, int j, int n, int *bracket, char &name) function

```
→if (i == j)
   {cout << name++;
  return;}
```

*Here if it’s same matrix then it will print the matrix name, like A,B,C……….

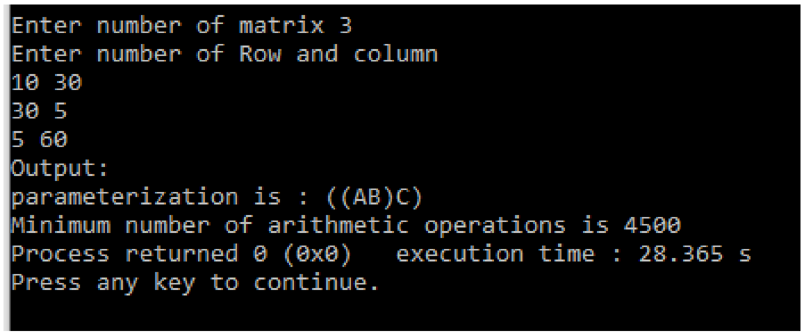
```
 →cout << "(";
   Parenthesis(i, *((bracket+i*n)+j), n, bracket, name);
   Parenthesis(*((bracket+i*n)+j) + 1, j, n, bracket, name);
   cout << ")";
```

*Here *((bracket+i*n) +j) means bracket[i][j].

This function works recursively and put brackets around subexpression. When we call this function from matrixChain function, we send first matrix(1) and last matrix(n-1) with bracket[][] 2Darray. Then in the function it again call its own function 2time. First time it call with the value of i, value in bracket[i][j] and second time with bracket[i][j]+1 and j.

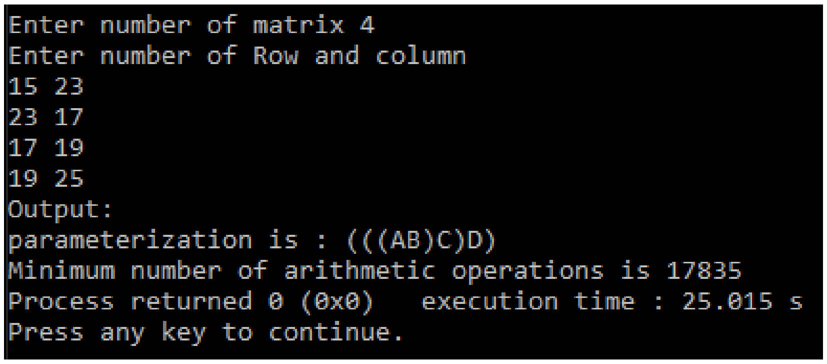
Testing Results:

*Here we take 3 matrixes as input. Let, A is a 10 × 30 matrix, B is a 30 × 5 matrix, and C is a 5 × 60 matrix. So output of Minimum number of arithmetic operations is 4500



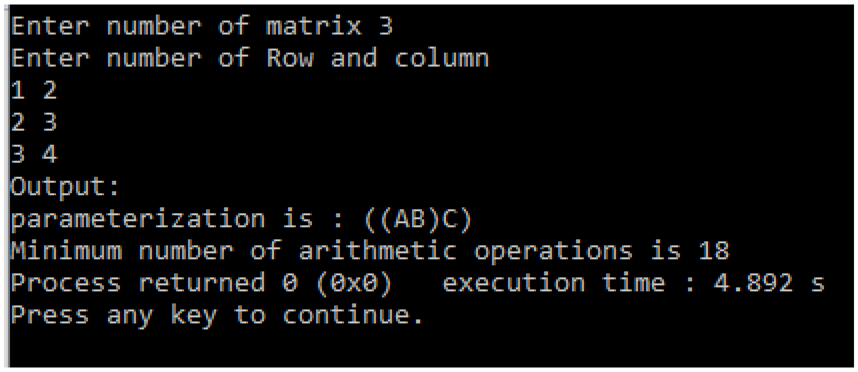
```
Enter number of matrix 3
Enter number of Row and column
10 30
30 5
5 60
Output:
parameterization is : ((AB)C)
Minimum number of arithmetic operations is 4500
Process returned 0 (0x0)   execution time : 28.365 s
Press any key to continue.
```

*Here we take 4 matrixes as input. Let, A is a 15 × 23 matrix, B is a 23 × 17 matrix, and C is a 17 × 19 matrix, D is a 19 x 25 matrix. So output of Minimum number of arithmetic operations is 17835.



```
Enter number of matrix 4
Enter number of Row and column
15 23
23 17
17 19
19 25
Output:
parameterization is : (((AB)C)D)
Minimum number of arithmetic operations is 17835
Process returned 0 (0x0)   execution time : 25.015 s
Press any key to continue.
```

*Here we take 3 matrixes as input. Let, A is a 1 × 2 matrix, B is a 2 × 3 matrix, and C is a 3 × 4 matrix. So output of Minimum number of arithmetic operations is 18



```
Enter number of matrix 3
Enter number of Row and column
1 2
2 3
3 4
Output:
parameterization is : ((AB)C)
Minimum number of arithmetic operations is 18
Process returned 0 (0x0)   execution time : 4.892 s
Press any key to continue.
```

Future Scope:

In this Matrix Chain Multiplication project, we calculate the minimum number of arithmetic operations. The matrix chain multiplication problem generalizes to solving a more abstract problem: given a linear sequence of objects, an associative binary operation on those objects, and a way to compute the cost of performing that operation on any two given objects (as well as all partial results), compute the minimum cost way to group the objects to apply the operation over the sequence.