DESIGN AND ANALYSIS OF ALGORITHMS

EXPERIMENT 5

ANUSHKA ACHARYA
CSE DS D1 BATCH
UID: 2021700001

AIM:

TO IMPLEMENT MATRIX CHAIN MULTIPLICATION USING DYNAMIC PROGRAMMING

THEORY:

It is a Method under Dynamic Programming in which previous output is taken as input for next.Here, Chain means one matrix's column is equal to the second matrix's row. In general:

If A = $\lfloor a_{ij} \rfloor$ is a p x q matrix; B = $\lfloor b_{ij} \rfloor$ is a q x r matrix; C = $\lfloor c_{ij} \rfloor$ is a p x r matrix

$$AB = C \text{ if } c_{ij} = \sum_{k=1}^{q} a_{ik}\, b_{kj}$$

Given following matrices {A_1,A_2,A_3,...A_n} and we have to perform the matrix multiplication, which can be accomplished by a series of matrix multiplications A_1 xA_2 x,A_3 x.....x A_n. Matrix Multiplication operation is associative in nature rather commutative. By this, we mean that we have to follow the above matrix order for multiplication but we are free to parenthesize the above multiplication depending upon our need.

In general, for 1≤ i≤ p and 1≤ j ≤ r

$$C[i, j] = \sum_{k=1}^{q} A[i, k]B[k, j]$$

It can be observed that the total entries in matrix 'C' is 'pr' as the matrix is of dimension p x r Also each entry takes O (q) times to compute, thus the total time to compute all possible entries for the matrix 'C' which is a multiplication of 'A' and 'B' is proportional to the product of the dimension p q r. It is also noticed that we can save the number of operations by reordering the parenthesis.

Let $A_{i,j}$ be the result of multiplying matrices i through j. It can be seen that the dimension of $A_{i,j}$ is p_{i-1} x p_j matrix. Dynamic Programming solution involves breaking up the problems into subproblems whose solution can be combined to solve the global problem. At the greatest level of parenthesization, we multiply two matrices

$A_{1.....n}$=$A_{1....k}$ x $A_{k+1....n}$

Thus we are left with two questions:

- How to split the sequence of matrices?
- How to parenthesize the subsequence $A_{1.....k}$ and$A_{k+1......n}$?

One possible answer to the first question for finding the best value of 'k' is to check all possible choices of 'k' and consider the best among them. But that it can be observed that checking all possibilities will lead to an exponential number of total possibilities. It can also be noticed that there exists only O (n^2) different sequence of matrices, in this way do not reach the exponential growth.

Step1: Structure of an optimal parenthesization: Our first step in the dynamic paradigm is to find the optimal substructure and then use it to construct an optimal solution to the problem from an optimal solution to subproblems.

Let $A_{i....j}$ where $i \leq j$ denotes the matrix that results from evaluating the product $A_i A_{i+1}....A_j$.

If $i < j$ then any parenthesization of the product $A_i A_{i+1} ......A_j$ must split that the product between A_k and A_{k+1} for some integer k in the range $i \leq k \leq j$. That is for some value of k, we first compute the matrices $A_{i.....k}$ & $A_{k+1....j}$ and then multiply them together to produce the final product $A_{i....j}$. The cost of computing $A_{i....k}$ plus the cost of computing $A_{k+1....j}$ plus the cost of multiplying them together is the cost of parenthesization.

Step 2: A Recursive Solution: Let m [i, j] be the minimum number of scalar multiplication needed to compute the matrix$A_{i....j}$. If i=j the chain consist of just one matrix $A_{i....i}=A_i$ so no scalar multiplication are necessary to compute the product. Thus m [i, j] = 0 for i= 1, 2, 3....n. If i<j we assume that to optimally parenthesize the product we split it between A_k and A_{k+1} where $i \leq k \leq j$. Then m [i,j] equals the minimum cost for computing the subproducts $A_{i....k}$ and $A_{k+1....j}$+ cost of multiplying them together. We know A_i has dimension p_{i-1} x p_i, so computing the product $A_{i....k}$ and $A_{k+1....j}$takes $p_{i-1}\ p_k\ p_j$ scalar multiplication, we obtain

$$m\,[i,j] = m\,[i, k] + m\,[k + 1, j] + p_{i-1}\ p_k\ p_j$$

There are only (j-1) possible values for 'k' namely k = i, i+1.....j-1. Since the optimal parenthesization must use one of these values for 'k' we need only check them all to find the best. So the minimum cost of parenthesizing the product $A_i A_{i+1}......A_j$ becomes

$$m\,[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\{m\,[i,k] + m\,[k+1,j] + p_{i-1}\,p_k p_j\} & \text{if } i < j \\ i \leq k < j & \end{cases}$$

To construct an optimal solution, let us define s [i,j] to be the value of 'k' at which we can split the product $A_i A_{i+1} .....A_j$ To obtain an optimal parenthesization i.e. s [i, j] = k such that

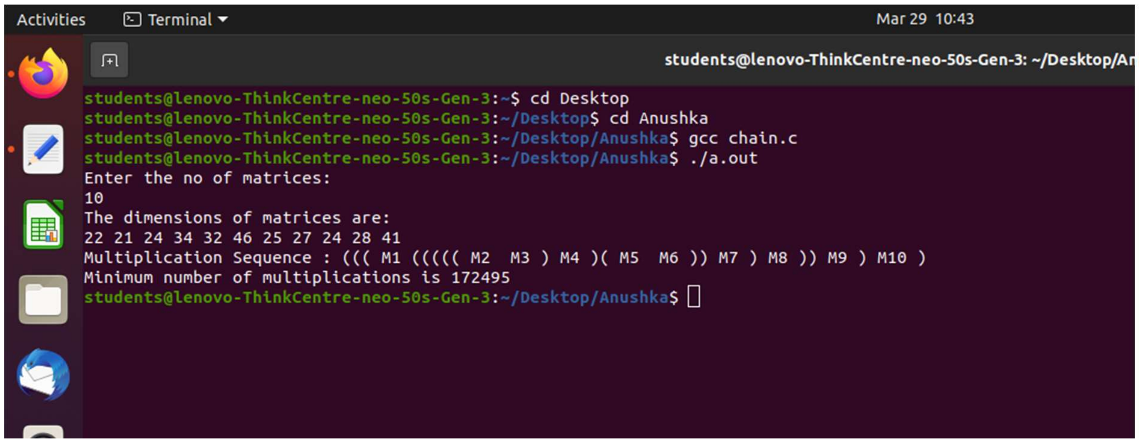
$$m\,[i,j] = m\,[i, k] + m\,[k + 1, j] + p_{i-1}\ p_k\ p_j$$

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>

int s[20][20],m[20][20],p[20];
int n;
void print(int i,int j){
if (i == j)
printf(" M%d ",i);
else
  {
      printf("(");
      print(i, s[i][j]);
      print(s[i][j] + 1, j);
      printf(")");
   }
}
void multiply(){
int q,k;
for(int i=n;i>0;i--)
 {
   for(int j=i;j<=n;j++)
    {
     if(i==j)
       m[i][j]=0;
     else
       {
        for(int k=i;k<j;k++)
        {
         q=m[i][k]+m[k+1][j]+p[i-1]*p[k]*p[j];
         if(q<m[i][j])
          {
            m[i][j]=q;
            s[i][j]=k;
          }
         }
        }
      }
 }
}
int chain(int p[], int i, int j)
{
    if(i == j)
        return 0;
    int k,min=INT_MAX,count=0;
    for (k = i; k < j; k++) {
        count = chain(p, i, k) + chain(p, k + 1, j) + p[i - 1] * p[k] * p[j];
        if (count < min)
            min = count;
    }
    return min;
}
int main(){
printf("Enter the no of matrices:\n");
scanf("%d",&n);
printf("The dimensions of matrices are:\n");
    for (int i = 0; i<=n; i++) {
        p[i]= (rand()%(46 - 15 + 1)) + 15;
        printf("%d ", p[i]);
    }
for(int i=1;i<=n;i++)
for(int j=i+1;j<=n;j++)
{
 m[i][i]=0;
 m[i][j]=INT_MAX;
 s[i][j]=0;
}
multiply();
printf("\nMultiplication Sequence : ");
print(1,n);
printf("\nMinimum number of multiplications is %d\n",chain(p, 1, n));
return 0;
}
```

OUTPUT:



```
students@lenovo-ThinkCentre-neo-50s-Gen-3:~$ cd Desktop
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop$ cd Anushka
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop/Anushka$ gcc chain.c
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop/Anushka$ ./a.out
Enter the no of matrices:
10
The dimensions of matrices are:
22 21 24 34 32 46 25 27 24 28 41
Multiplication Sequence : ((( M1 ((((( M2  M3 ) M4 )( M5  M6 )) M7 ) M8 )) M9 ) M10 )
Minimum number of multiplications is 172495
students@lenovo-ThinkCentre-neo-50s-Gen-3:~/Desktop/Anushka$
```

CONCLUSION:

From this experiment I learnt how to use Dynamic Programming Approach to multiply more than two matrices. I understood how to minimise computation time by reducing number of multiplications using Matrix Chain Multiplication Algorithm.