

NAME : ROHIT SHEKHAR BAVISKAR

UID : 2021700004

BATCH : CSE DS D1

EXP : DAA - 5

AIM:

Implement matrix chain multiplication using dynamic programming

THEORY:

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

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 $\{[i, i+1], [i+1, j]\}$, $\{[i, i+2], [i+2, j]\}$, \dots , $\{[i, j-1], [j-1, j]\}$.

- 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.

ALGORITHM :

MatrixChainMultiplication(mat[], low, high):

// 0 steps are required if low equals high

// case of only 1 sized sub-problem.

If(low==high):

return 0

// Initialize minCost to a very big number.

minCost = Infinity

// Iterate from k = low to k = high-1

For(k=low to high-1):

/*

Cost = Cost of Multiplying chain on left side + Cost
of Multiplying chain on right side + Cost of
Multiplying matrix obtained from left and right
side.

*/

cost=MatrixChainMultiplication(mat, low, k)+
MatrixChainMultiplication(mat, low+1, high)+
mat[low-1]*mat[k]*mat[high]

```

        // Update the minCost if cost<minCost.
        If(cost<minCost):
minCost=costreturn minCost

```

PROGRAM :

```

#include <stdio.h>
#include <limits.h>
#include <stdlib.h>
void printParanthesis(int **s, int i, int j)
{
    if (i == j)
        printf("A%d", i + 1);
    else
    {
        printf("(");
        printParanthesis(s, i, s[i][j]);
        printf("*");
        printParanthesis(s, s[i][j] + 1, j);
        printf(")");
    }
}

void parenthesizeMatrixChain(int *p, int n)
{
    int m[n][n];
    // int s[n][n];
    int **s = malloc(sizeof(int *) * n);
    for (int i = 0; i < n; i++)
        s[i] = malloc(sizeof(int) * n);
    for (int i = 0; i < n; i++)
        m[i][i] = 0;
    int j, cost;
    for (int chain_length = 2; chain_length <= n;
        chain_length++)
    {
        for (int i = 0; i <= n - chain_length; i++) // i
            is the starting index of a matrix subchain
        {
            j = i + chain_length - 1; // j is the ending
            index of the matrix subchain
            m[i][j] = INT_MAX;
            for (int k = i; k <= j - 1; k++)
            {
                cost = m[i][k] + m[k + 1][j] + p[i] * p[k
                    + 1] * p[j + 1];
            }
        }
    }
}

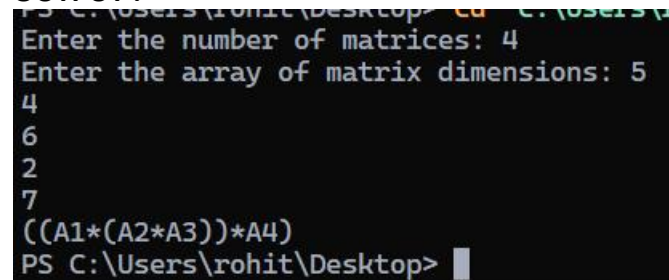
```

```

if (cost < m[i][j])
{
    m[i][j] = cost;
    s[i][j] = k;
}
}
}
}
printParanthesis(s, 0, n - 1);
for (int i = 0; i < n; i++)
    free(s[i]);
free(s);
}
int main()
{
    printf("Enter the number of matrices: ");
    int n;
    scanf("%d", &n);
    int p[n + 1];
    printf("Enter the array of matrix dimensions: ");
    for (int i = 0; i < n + 1; i++)
        scanf("%d", p + i);
    parenthesizeMatrixChain(p, n);
}

```

OUTPUT :



```

PS C:\Users\rohit\Desktop> cd C:\Users\rohit\Desktop
Enter the number of matrices: 4
Enter the array of matrix dimensions: 5
4
6
2
7
((A1*(A2*A3))*A4)
PS C:\Users\rohit\Desktop>

```

RESULT ANALYSIS :

Time Complexity -

We are using three nested for loops, each of which is iterating roughly $O(n)$ times. Hence, the overall time complexity is $O(n^3)$.

Space Complexity - We are using an auxiliary dp array of dimensions, $(n-1) \times (n-1)$ hence space complexity is $O(n^2)$

In the matrix chain multiplication problem, the minimum number of multiplication steps required to multiply a chain of matrices has been calculated.

- Determining the minimum number of steps required can highly speed up the multiplication process.

- It takes $O(n^3)$ time and $O(n^2)$ auxiliary space to calculate the minimum number of steps required to multiply a chain of matrices using the dynamic programming method.
- It is very important to decide the order of multiplication of matrices to perform the task efficiently.
- So the minimum number of multiplication steps required to multiply a chain matrix of length n has been computed.
- Trying out all the possibilities recursively is very time consuming hence the method of dynamic programming is used to find the same.
- The minimum number of steps required to multiply a chain of matrices can be found in $O(n^3)$ time complexity using dynamic programming.

CONCLUSION :

In this experiment I understood about how to find the minimal cost of matrix chain multiplication using dynamic programming