|  |  |
| --- | --- |
| **Name** | Parth Gandhi |
| **UID No.** | 2021300033 |
| **Class & Division** | S.E. COMPS A (BATCH B) |
| **Experiment No.** | 4 |

**Aim:** To implement dynamic algorithms

**Theory:**

Matrix Chain Multiplication is a well-known optimization problem in computer science that deals with the optimal way to multiply a sequence of matrices. Given a sequence of matrices {A1, A2, A3, ..., An}, where the dimensions of matrix Ai is given by pi-1 x pi, the goal is to find the minimum number of scalar multiplications required to compute the product A1 x A2 x A3 x ... x An.

The number of scalar multiplications required to compute the product of two matrices Ai and Aj is given by pi-1 x pj x pk, where k is an integer such that i <= k < j. Therefore, the total number of scalar multiplications required to compute the product of the entire sequence of matrices can be expressed as the sum of the products of all possible pairs of matrices in the sequence.

The problem of Matrix Chain Multiplication can be solved efficiently using dynamic programming. The basic idea is to build a table M of size n x n, where M[i][j] stores the minimum number of scalar multiplications required to compute the product Ai x Ai+1 x ... x Aj.

**Algorithm:**

1. Initialize the diagonal elements of the table M to 0, since the product of a single matrix does not require any multiplication.
2. For each subsequence of matrices of increasing length, compute the minimum number of scalar multiplications required to compute their product. The outer loop iterates over the length of the subsequence, while the inner loop iterates over the starting index of the subsequence.
3. To compute M[i][j], consider all possible ways to split the subsequence Ai x Ai+1 x ... x Aj into two subsequences Ai x Ai+1 x ... x Ak and Ak+1 x Ak+2 x ... x Aj, where i <= k < j. The number of scalar multiplications required to compute the product of these two subsequences is given by M[i][k] + M[k+1][j] + pi-1 x pk x pj. Choose the minimum among all possible splits and store the result in M[i][j].
4. The final result is given by M[1][n], which gives the minimum number of scalar multiplications required to compute the product A1 x A2 x ... x An.

**Code:**

#include<bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cout<<"Enter number of dimensions:";

    cin>>n;

    int a[n];

    cout<<"Enter dimensions:";

    for(int i=0;i<n;i++)

    cin>>a[i];

    int m[n][n];

    for(int i=0;i<n;i++)

    {

        for(int j=0;j<n;j++)

        {

            if(i==0 || j==0 || i==j)

            m[i][j]=0;

        }

    }

    for(int r=1;r<n;r++)

    {

        int t,j=1+r;

        for(int i=1;j<n;i++,j=i+r)

        {

            for(int k=i;k<j;k++)

            {

                t=m[i][k]+m[k+1][j]+a[i-1]\*a[k]\*a[j];

                if(k==i)

                m[i][j]=t;

                else

                m[i][j]=min(t,m[i][j]);

            }

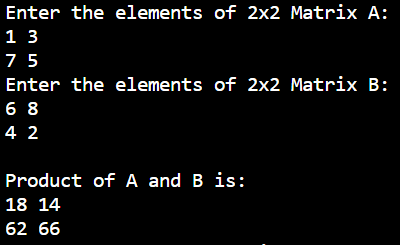
        }

    }

    cout<<"Optimal cost for parenthesization is:"<<m[1][n-1]<<endl;

}

**Output:**



**Conclusion:** Successfully wrote a program to implement Strassen matrix mutliplication.