NAME:	Prithvi Singh
UID:	2022301014
SUBJECT	DAA
EXPERIMENT NO:	04
AIM:	Experiment to implement matrix chain multiplication
Algorithm:	• **MATRIX-CHAIN-ORDER (p)** 1. n length[p]-1 2. for i ← 1 to n 3. do m [i, i] ← 0 4. for l ← 2 to n // l is the chain length 5. do for i ← 1 to n-l + 1 6. do j ← i+ l -1 7. m[i,j] ← ∞ 8. for k ← i to j-1 9. do q ← m [i, k] + m [k + 1, j] + pi-1 pk pj 10. If q < m [i,j] 11. then m [i,j] ← q 12. s [i,j] ← k 13. return m and s. • **PRINT-OPTIMAL-PARENS (s, i, j)** 1. if i=j 2. then print "A" 3. else print "(" 4. PRINT-OPTIMAL-PARENS (s, i, s [i, j]) 5. PRINT-OPTIMAL-PARENS (s, s [i, j] + 1, j) 6. print ")"

Code:

```
#include <iostream>
#include <climits>
#include <random>
#include <ctime>
using namespace std;
void matrixChainOrder(int p[], int n, int m[][100], int s[][100])
{
    for(int i=1; i<=n; i++)
    m[i][i] = 0; for(int l=2; l<=n; l++)
    {
        for(int i=1; i<=n-l+1; i++)
        { int j = i+l-1;
            m[i][j] = INT_MAX;

            for(int k=i; k<=j-1; k++)
            {

                int 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;

                }

            }

        }

    }

}



void printOptimalParenthesis(int s[][100], int i, int j)
{
    if(i == j)
    cout << "A" << i;
    else
    {
        cout << "("; printOptimalParenthesis(s, i, s[i][j]);
printOptimalParenthesis(s, s[i][j]+1, j); cout << ")";
```

```
    }

}

int main()
{
    int p[10];
    srand ( time(NULL) ); random_device rd; mt19937 gen(rd());
    uniform_int_distribution<> distr(15, 46); for(int i=0; i<10;
++i)
    p[i] = distr(gen);

    int n = sizeof(p)/sizeof(p[0]) - 1; generateMatrices(p);
    int m[100][100];

    int s[100][100];

    matrixChainOrder(p, n, m, s);

    cout << "Optimal Parenthesization: "; printOptimalParenthesis(s,
1, n);
    cout << endl;

    cout << "Minimum Number of Scalar Multiplications: " << m[1][n]
<< endl; cout << "m table:";
    for(int a = 0; a < 10; a++)

    {

        for(int b = 0; b < 10; b++)

        {

            if(m[a][b] == 0){continue;}

            cout << m[a][b] << " ";

        }

        cout << endl;

    }
    cout << "s table:";
```

```
    for(int a = 0; a < 10; a++)
    {
        for(int b = 0; b < 10; b++)
        {
            if(s[a][b] == 0){continue;}
            cout << s[a][b] << " ";
        }

        cout << endl;

    }

    return 0;

}
```

Output:

- **With Random P values:**

```
PS C:\Users\prith\OneDrive\Desktop\Semester 4\DAA Practicals\EXP4> g++ mcm.cpp
PS C:\Users\prith\OneDrive\Desktop\Semester 4\DAA Practicals\EXP4> ./a.exe
Optimal Parenthesization: ((A1(A2(A3A4)))((((A5A6)A7)A8)A9))
Minimum Number of Scalar Multiplications: 165225
m table:
31616 68224 50220 66540 91800 106050 133695 165225
43472 31980 51360 77340 90600 119505 151125
17160 30420 54960 70200 96585 128025
22440 49140 61410 91575 123285
21420 40950 61875 92925
44268 91698 156922
58590 124062
64170

s table:
1 2 1 4 4 4 4 4
2 2 4 4 4 4 4
3 4 4 4 4 4
4 4 4 4 4
5 6 7 8
6 7 7
7 7
8
```

- **Observation:**

- Hence, after performing the experiment I have observed that the order of matrix while multiplication is critical while multiplying the matrices.

	• Determining the optimal way to parenthesize the matrices will significantly decrease the number of scalar multiplications needed to obtain the final result. For example • If A is a 10 × 30 matrix, B is a 30 × 5 matrix, and C is a 5 × 60 matrix, then 1. Computing (AB)C needs (10×30×5) + (10×5×60) = 1500 + 3000 = 4500 operations, while 2. Computing A(BC) needs (30×5×60) + (10×30×60) = 9000 + 18000 = 27000 operations. Therefore, I found out that finding an optimal way to multiply will significantly reduce running times especially with large matrix values.
Conclusion:	Thus, by performing this experiment I understood the significance of matrix chain multiplication and was also able to implement it.