Name	Shivam S. Kamble
UID	2022301007
Subject	DAA
Exp No.	04

Aim – Experiment to implement matrix chain multiplication.

Theory: Matrix chain multiplication (or the matrix chain ordering problem [1]) is an optimization problem concerning 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. The problem may be solved using dynamic programming. There are many options because matrix multiplication is associative. In other words, no matter how the product is parenthesized, the result obtained will remain the same.

For example, for four matrices A, B, C, and D, there are five possible options:

((AB)C)D = (A(BC))D = (AB)(CD) = A((BC)D) = A(B(CD)).

The idea is to break the problem into a set of related subproblems that group the given matrix to yield the lowest total cost. Following is the recursive algorithm to find the minimum cost: • Take the sequence of matrices and separate it into two subsequences.

• Find the minimum cost of multiplying out each subsequence.

• Add these costs together, and add in the price of multiplying the two result matrices.

• Do this for each possible position at which the sequence of matrices can be split, and take the minimum over all of them.

For example, if we have four matrices ABCD, we compute the cost required to find each of (A) (BCD), (AB)(CD), and (ABC)(D), making recursive calls to find the minimum cost to compute ABC, AB, CD, and BCD and then choose the best one. Better still, this yields the minimum cost and demonstrates the best way of doing the multiplication

Code :

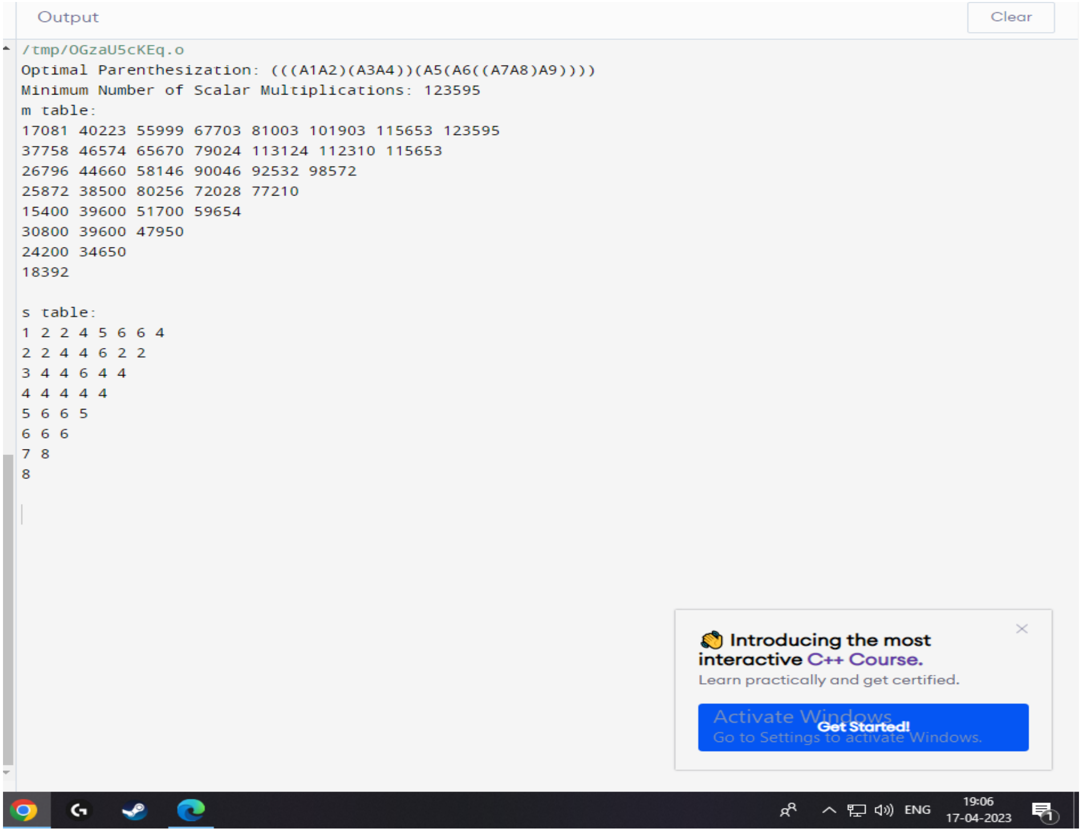
```
#include <iostream>

#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;

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 :



Conclusion : Thus, by performing this experiment I have understood the importance of matrix chain multiplication and was also able to implement it.