Name: Yash Ravindra Burad
UID: 2022301004
Subject: DAA
Experiment number: 04
Aim: To implement dynamic algorithms
Theory:
Details – Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping subproblems and optimal substructure property. If any problem can be divided into sub-problems, which in turn are divided into smaller sub-problems, and if there are overlapping among these sub-problems, then the solutions to these sub-problems can be saved for future reference. The approach of solving problems using dynamic programming algorithm has following steps:

1 .Characterize the structure of an optimal solution.
2.Recursively define the value of an optimal solution.
3.Compute the value of an optimal solution, typically in a bottom-up fashion.
4.Construct an optimal solution from computed information.

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] + p_{i-1} p_k p_j
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 ")"

Program:

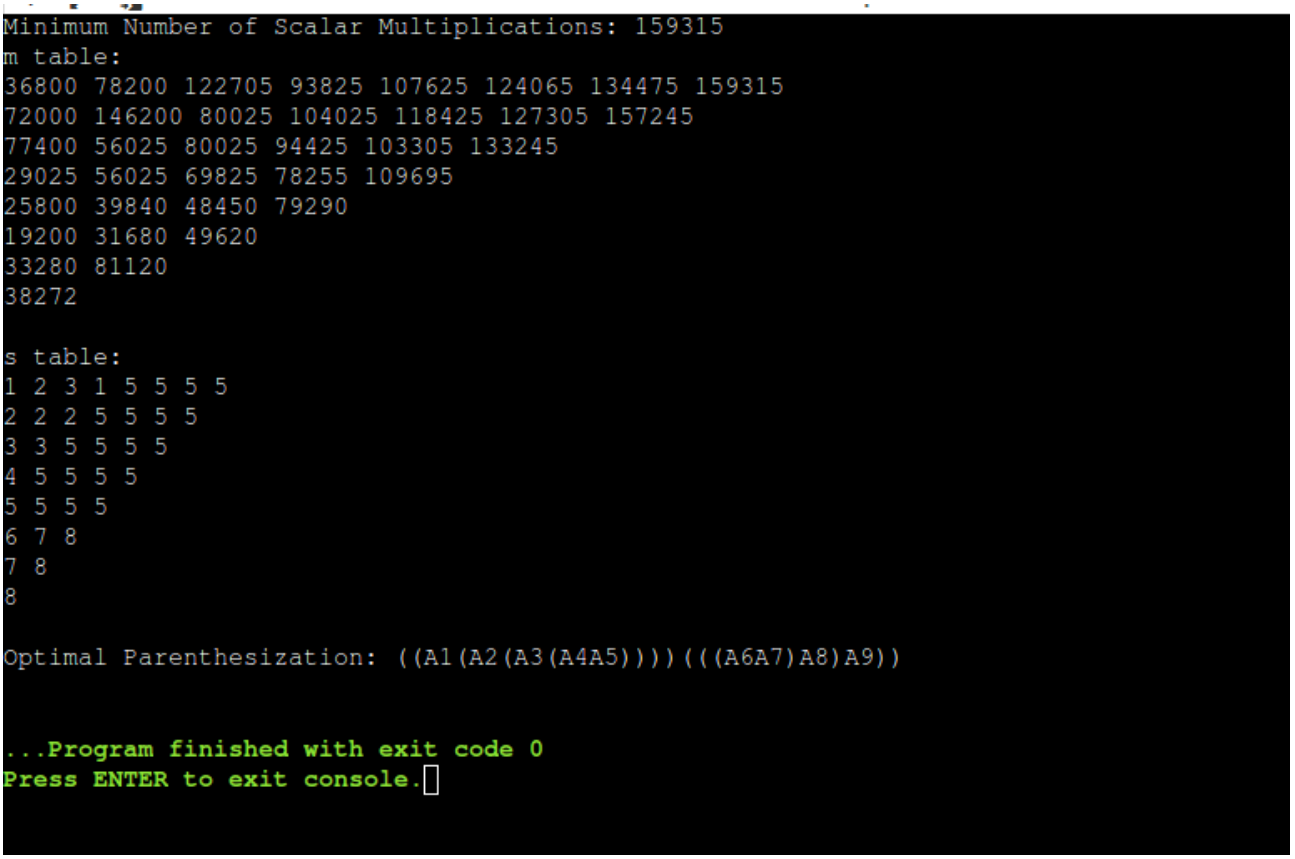
```cpp
#include <iostream>
#include <climits>
#include <random>
#include <ctime>
using namespace std;
int **matrix;

//determining cost of multiplication
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;

            //finding cost i.e., number of Multiplications
            for(int k=i; k<=j-1; k++) {
                int q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];

                //finding minimum number of scalar Multiplications
                if(q < m[i][j]) {
                    m[i][j] = q;
                    s[i][j] = k;
                }
```

```
                    s[i][j] = k;
                }
            }
        }
    }
}

//printing the parenthisis
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 << "Minimum Number of Scalar Multiplications: " << m[1][n] << endl;

//printing cost table
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;
}


//printing parenthesization table
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;
    }

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


    return 0;
}
```

output:



Time complexity Analysis:

There are three nested loops. Each loop executes a maximum n times.

1. l, length, O (n) iterations.
2. i, start, O (n) iterations.
3. k, split point, O (n) iterations

So, Body of loop constant complexity

Total Complexity is: O (n^3)

Inference:
While we perform matrix multiplication, the number of multiplications increases (as multiplication takes more time than addition and subtraction). But if we optimally parenthesize the matrices then the number of multiplications may decrease. For that we use a dynamic programming approach and perform the matrix chain multiplication and give the optimal parenthesis. In the matrix chain multiplication problem, the minimum number of multiplication steps required to multiply a chain of matrices has been calculated.

Conclusion:

In this Practical, I performed the program of matrix chain multiplication and found the minimum number of multiplication steps required to multiply the chain of metrics and accordingly give the optimal parenthesization.