DESIGN AND ANALYSIS OF ALGORITHMS LAB

NAME: Mohammed Saad Belgi

UID: 2021700005

BATCH: A

BRANCH: CSE DS

EXPT. NO.: 5

AIM: Experiment using dynamic programming approach: finding optimal parenthesization for matrix chain multiplication

ALGORITHM:

```
MATRIX-CHAIN-ORDER(p)
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i, j] = ∞
        for k = i to j - 1
            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
return m and s
```

```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```

CODE:

```
#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:

```
Enter the number of matrices: 5
Enter the array of matrix dimensions: 4 10 3 12 20 7
((A1*A2)*((A3*A4)*A5))
(base) PS C:\Users\arifa\Desktop\sem4 work\daa lab\exp5>
```

CONCLUSION:

The problem of finding optimal parenthesization of a matrix chain has optimal substructure property as well as overlapping subproblems property. Hence, it can be solved in $O(n^3)$ time using dynamic programming (n is the length of matrix chain) instead of exponential time required by simple divide and conquer approach.