BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to Mumbai University)
Department Of Computer Engineering

Name	Manish Shashikant Jadhav
UID	2023301005
Subject	Design and Analysis of Algorithms (DAA)
Experiment No.	4
Aim	To implement Dynamic Algorithms. a) Assembly Line Scheduling. b) Longest Common Subsequence.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

#define NUM_MATRICES 10

// Function to allocate memory for a 2D matrix
int** allocateMatrix(int rows, int cols) {
    int **matrix = (int **)malloc(rows * sizeof(int *));
    for (int i = 0; i < rows; i++) {
        matrix[i] = (int *)malloc(cols * sizeof(int));
    }
    return matrix;
}

// Function to free memory for a 2D matrix
void freeMatrix(int **matrix, int rows) {
    for (int i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

// Function to generate random matrices
void generateRandomMatrices(int ***matrices, int *dims) {
    for (int i = 0; i < NUM_MATRICES; i++) {
```

```
        int rows = dims[i];
        int cols = dims[i + 1];
        matrices[i] = allocateMatrix(rows, cols);
        for (int row = 0; row < rows; row++) {
            for (int col = 0; col < cols; col++) {
                matrices[i][row][col] = rand() % 2; // Random
values between 0 and 1
            }
        }
    }
}

// Function to perform Matrix Chain Multiplication using
dynamic programming
void matrixChainMultiplication(int dims[], int n, int **m, int
**c) {
    for (int i = 1; i <= n; i++) {
        m[i][i] = 0;
    }

    for (int len = 2; len <= n; len++) {
        for (int i = 1; i <= n - len + 1; i++) {
            int j = i + len - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k < j; k++) {
                int cost = m[i][k] + m[k+1][j] + dims[i-
1]*dims[k]*dims[j];
                if (cost < m[i][j]) {
                    m[i][j] = cost;
                    c[i][j] = k;
                }
            }
        }
    }
}

int main() {
```

```
    srand(time(NULL));

    // Generate random dimensions for matrices
    int dims[NUM_MATRICES + 1]; // +1 to include the
dimensions of result matrix
    printf("Random Dimensions for Matrices:\n");
    for (int i = 0; i <= NUM_MATRICES; i++) {
        dims[i] = rand() % 32 + 15; // Random dimensions
between 15 and 46
        printf("M%d: %2d x %2d\n", i, dims[i-1], dims[i]);
    }

    // Generate random matrices
    int ***matrices = (int ***)malloc(NUM_MATRICES *
sizeof(int **));
    generateRandomMatrices(matrices, dims);

    // Allocate memory for storing optimal solutions and
parenthesizations
    int **m = allocateMatrix(NUM_MATRICES + 1, NUM_MATRICES +
1);
    int **c = allocateMatrix(NUM_MATRICES + 1, NUM_MATRICES +
1);

    // Perform Matrix Chain Multiplication and measure time
    clock_t start = clock();
    matrixChainMultiplication(dims, NUM_MATRICES, m, c);
    clock_t end = clock();
    double duration_mcm = ((double)(end - start)) /
CLOCKS_PER_SEC;

    // Print optimal solutions
    printf("\nOptimal Solutions (No. of Multiplications):\n");
    for (int i = 1; i <= NUM_MATRICES; i++) {
        for (int j = 1; j <= NUM_MATRICES; j++) {
            printf("%6d", m[i][j]);
        }
```

```
        printf("\n");
    }
    // Print time for Matrix Chain Multiplication
    printf("\nTime for Matrix Chain Multiplication: %.6f
seconds\n", duration_mcm);

    // Print the cost matrix and the k matrix
    printf("\nCost Matrix:\n");
    for (int i = 1; i <= NUM_MATRICES; i++) {
        for (int j = 1; j <= NUM_MATRICES; j++) {
            printf("%6d", m[i][j]);
        }
        printf("\n");
    }

    printf("\nK Matrix:\n");
    for (int i = 1; i <= NUM_MATRICES; i++) {
        for (int j = 1; j <= NUM_MATRICES; j++) {
            printf("%6d", c[i][j]);
        }
        printf("\n");
    }

    // Free allocated memory for matrices
    for (int i = 0; i < NUM_MATRICES; i++) {
        freeMatrix(matrices[i], dims[i]);
    }
    free(matrices);

    // Free allocated memory for optimal solutions and
parenthesizations
    freeMatrix(m, NUM_MATRICES + 1);
    freeMatrix(c, NUM_MATRICES + 1);

    return 0;
}
```

Output

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    GITLENS    SEARCH ERROR

PS D:\Manish\SPIT\4th SEM\DAA\Exp5> cd 'd:\Manish\SPIT\4th SEM\DAA\Exp5\output'
PS D:\Manish\SPIT\4th SEM\DAA\Exp5\output> & .\'mcm.exe'
Random Dimensions for Matrices:
M0:  8 x 35
M1: 35 x 33
M2: 33 x 21
M3: 21 x 34
M4: 34 x 16
M5: 16 x 23
M6: 23 x 26
M7: 26 x 28
M8: 28 x 34
M9: 34 x 29
M10: 29 x 25

Optimal Solutions (No. of Multiplications):
     0 24255 49245 40992 53872 65120 77888 9648010945611816
8586656     0 23562 22512 34656 45808 58512 76912 90048 99536
8586656     0     0 11424 19152 29728 42048 59296 73392 83648
8586656     0     0     0 12512 23712 36448 54944 68000 77424
8586656     0     0     0     0  9568 21216 36448 52224 63824
8586656     0     0     0     0     0 16744 38640 61318 77993
8586656     0     0     0     0     0     0 24752 48720 66108
8586656     0     0     0     0     0     0     0 27608 47908
8586656     0     0     0     0     0     0     0     0 24650
8586656     0     0     0     0     0     0     0     0     0

Time for Matrix Chain Multiplication: 0.000000 seconds
```

```
Cost Matrix:
     0 24255 49245 40992 53872 65120 77888 9648010945611816
8586656     0 23562 22512 34656 45808 58512 76912 90048 99536
8586656     0     0 11424 19152 29728 42048 59296 73392 83648
8586656     0     0     0 12512 23712 36448 54944 68000 77424
8586656     0     0     0     0  9568 21216 36448 52224 63824
8586656     0     0     0     0     0 16744 38640 61318 77993
8586656     0     0     0     0     0     0 24752 48720 66108
8586656     0     0     0     0     0     0     0 27608 47908
8586656     0     0     0     0     0     0     0     0 24650
8586656     0     0     0     0     0     0     0     0     0

K Matrix:
8586656     1     2     1     4     4     4     4     4     4
8586656     0     2     2     4     4     4     4     4     4
     0     0     0     3     4     4     4     4     4     4
     0     0     0     0     4     4     4     4     4     4
     0     0     0     0     0     5     6     7     8     9
     0     0     0     0     0     0     6     7     8     9
     0     0     0     0     0     0     0     7     7     7
     0     0     0     0     0     0     0     0     8     9
     0     0     0     0     0     0     0     0     0     9
     0     0     0     0     0     0     0     0     0     0
PS D:\Manish\SPIT\4th SEM\DAA\Exp5\output>
```

Pseudo Code	

॥श्री॥

Date
Page

Experiment No:- 5

1. Pseudo Code

Matrix_chain_order (P)

1. n ← length [P] - 1
2. For i ← 1 to n
3. do m [i, j] ← 0
4. For L ← 2 to n   ▷ L in the chain length
5. do for i ← 1 to n - l + 1

do j ← i + l - 1

m [1, j] ← ∞

for k ← i to j - 1

do q ← m [i, k] + m [k + 1, j] + P_{i-1} $P_k P_j$

if q < m [i, j]

then m [i, j] ← q

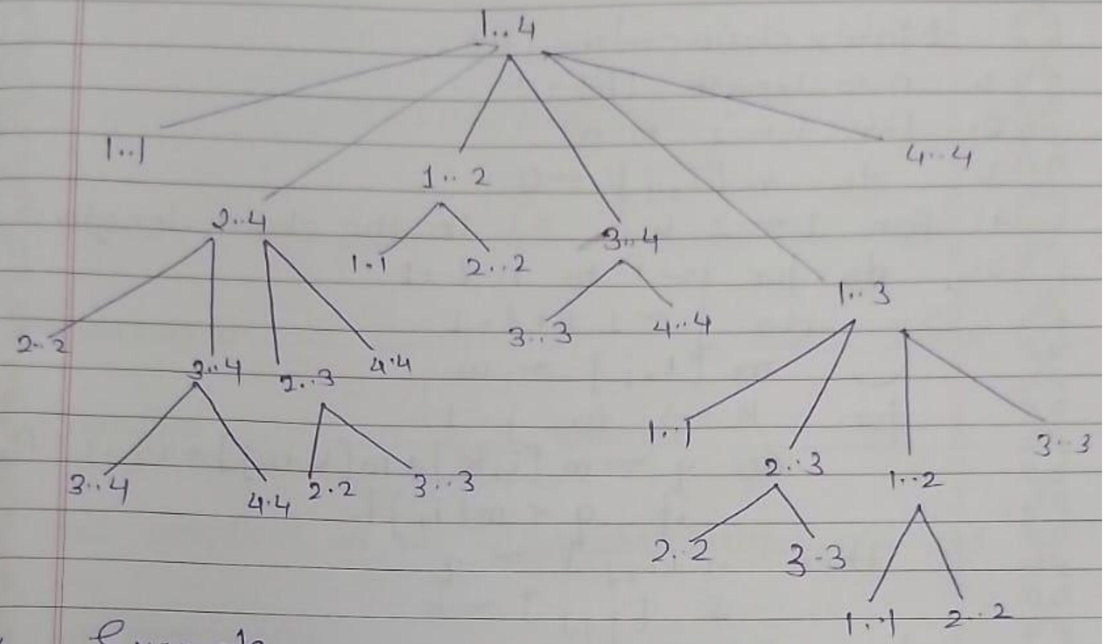
s [i, j] ← k

return m and s

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \leq k < j} \left\{ m[i,k] + m[k+1,j] + P_{i-1} P_k P_j \right. & \text{if } (i < j) \end{cases}$$

- Below is the recursion tree for recursive approach:-

Example



* Example

*

P_0	P_1	P_2	P_3	P_4
5	6	4	2	3

A_1	$A2$	$A3$	$A4$
5×6	6×4	4×2	2×3

m	1	2	3	4
1	0	120	108	54
2	0	0	48	84
3	0	0	0	24
4	0	0	0	0

s	1	2	3	4
1	0	1	1	3
2	0	0	2	3
3	0	0	0	3
4	0	0	0	0

$$m[1,2] = m[1,1] + m[2,2] + P_0 \cdot P_1 \cdot P_2$$
$$= 0 + 0 + 5 \cdot 6 \cdot 4$$
$$= \underline{\underline{120}}$$

$$m[2,3] = m[2,2] + m[3,3] + P_1 \cdot P_2 \cdot P_3$$
$$= 0 + 0 + 6 \cdot 4 \cdot 2$$
$$= \underline{\underline{48}}$$

$$m[3,4] = m[3,3] + m[4,4] + P_2 \cdot P_3 \cdot P_4$$
$$= 0 + 0 + 4 \cdot 2 \cdot 3$$
$$= \underline{\underline{24}}$$

$$m[1,3] = m[1,1] + m[2,3] + P_0 \cdot P_1 \cdot P_3$$
$$= 0 + 48 + 5 \cdot 6 \cdot 2$$
$$= \underline{\underline{108}}$$

$$m[1,3] = m[1,2] + m[3,3] + P_0 \cdot P_2 \cdot P_3$$
$$= 120 + 0 + 5 \cdot 4 \cdot 2$$
$$= \underline{\underline{160}}$$

$m[2,4] = A_2 \cdot (A_3 \cdot A_4) \qquad (A_2 \cdot A_3) \cdot A_4$

$m[2,4] = m[2,2] + m[3,4] + P_1 \cdot P_2 \cdot P_4$
$= 0 + 24 + 6 \cdot 4 \cdot 3$
$= \underline{96}$

$m[2,4] = m[2,3] + m[4,4] + P_1 \cdot P_3 \cdot P_4$
$= 48 + 0 + 6 \cdot 2 \cdot 3$
$= \underline{84}$

$m[1,4] = m[1,1] + m[2,4] + P_0 \cdot P_1 \cdot P_4$
$= 0 + 84 + 5 \cdot 6 \cdot 3$
$= \underline{174}$

$m[1,4] = m[1,2] + m[3,4] + P_0 \cdot P_2 \cdot P_4$
$= 120 + 24 + 5 \cdot 4 \cdot 3$
$= \underline{204}$

$m[1,4] = m[1,3] + m[4,4] + P_0 \cdot P_3 \cdot P_4$
$= 24 + 0 + 5 \cdot 2 \cdot 3$
$= \underline{54}$

Conclusion	Hence, by completing this experiment I came to know about implementation of Matrix Chain Multiplication.