



**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

# **School of Computer Science and Engineering (SCOPE)**

**Assessment – 2**

**Design and analysis of Algorithm (Lab Component)**

**MCSE502L**

Course Name : Design and analysis of Algorithm (Lab Component)

Course Code : MCSE502L

Slot: L35+L36

Course instructor: MOHAMMAD ARIF

**Name : NIDHI SINGH**  
**Registration no. : 22MAI0015**

## Assessment – 2

### MCSE502L Design and analysis of Algorithm (Lab Component)

- 1. Implement Matrix Chain Multiplication.**
- 2. Implement Longest Common Subsequence.**
- 3. Implement Subset-sum.**

### 1. Implement Matrix Chain Multiplication.

In the matrix chain multiplication problem, an array encoding the dimensions of the matrices  $[M_1, M_2, \dots, M_n]$  has been provided.  $[M_1, M_2, \dots, M_n]$  and we're trying to figure out how many multiplication steps are needed to multiply them.

Assume that we have three matrices,  $M_1$ ,  $M_2$ , and  $M_3$ , with dimensions of  $5 \times 10$ ,  $10 \times 8$ , and  $8 \times 5$  correspondingly, and that we are interested in multiplying all of them for whatever reason. Two different multiplication orders are possible:

#### EXAMPLE:-

$M_1 \times (M_2 \times M_3)$

- Steps required in  $M_2 \times M_3$  will be  $10 \times 8 \times 5 = 400$ .
- Dimensions of  $M_{23}$  will be  $10 \times 5$ .
- Steps required in  $M_1 \times M_{23}$  will be  $5 \times 10 \times 5 = 250$ .
- Total Steps  $= 400 + 250 = 650$

$(M_1 \times M_2) \times M_3$

- Steps required in  $M_1 \times M_2$  will be  $5 \times 10 \times 8 = 400$ .
- Dimensions of  $M_{12}$  will be  $5 \times 8$ .
- Steps required in  $M_{12} \times M_3$  will be  $5 \times 8 \times 5 = 200$ .
- Total Steps  $= 400 + 200 = 600$

**Algorithm:**

- Iterate from  $l = 2$  to  $N-1$  which denotes the length of the range:
  - Iterate from  $i = 0$  to  $N-1$ :
    - Find the right end of the range ( $j$ ) having  $l$  matrices.
    - Iterate from  $k = i+1$  to  $j$  which denotes the point of partition.
      - Multiply the matrices in range ( $i, k$ ) and ( $k, j$ ).
      - This will create two matrices with dimensions  $arr[i-1]*arr[k]$  and  $arr[k]*arr[j]$ .
      - The number of multiplications to be performed to multiply these two matrices (say  $X$ ) are  $arr[i-1]*arr[k]*arr[j]$ .
      - The total number of multiplications is  $dp[i][k] + dp[k+1][j] + X$ .
- The value stored at  $dp[1][N-1]$  is the required answer.

**Algorithm matrix Chain(S):****Algorithm matrixChain(S):****Input:** sequence  $S$  of  $n$  matrices to be multiplied**Output:** number of operations in an optimal parametrization of  $S$ **for**  $i \leftarrow 1$  to  $n-1$  **do**     $C_{i,i} \leftarrow 0$ **for**  $b \leftarrow 1$  to  $n-1$  **do**    **for**  $i \leftarrow 0$  to  $n-b-1$  **do**         $j \leftarrow i+b$          $C_{i,j} \leftarrow +\text{infinity}$         **for**  $k \leftarrow i$  to  $j-1$  **do**             $C_{i,j} \leftarrow \min\{C_{i,j}, C_{i,k} + C_{k+1,j} + d_i d_{k+1} d_{j+1}\}$ **Complexity Analysis****Time Complexity:**  $O(N^3)$ **Auxiliary Space:**  $O(N^2)$

**Program:**

```
#include <limits.h>
#include <stdio.h>
int MatrixChainOrder(int p[], int n)
{
    int m[n][n];
    int i, j, k, L, q;
    for (i = 1; i < n; i++)
        m[i][i] = 0;
    for (L = 2; L < n; L++) {
        for (i = 1; i < n - L + 1; i++)
        {
            j = i + L - 1;
            m[i][j] = INT_MAX;
            for (k = i; k <= j - 1; k++)
            {
                q = m[i][k] + m[k + 1][j]
                    + p[i - 1] * p[k] * p[j];
                if (q < m[i][j])
                    m[i][j] = q;
            }
        }
    }
    return m[1][n - 1];
}

int main()
{
    int arr[] = { 1, 2, 3, 4 };
    int size = sizeof(arr) / sizeof(arr[0]);
    printf("Minimum number of multiplications is %d ", MatrixChainOrder(arr, size));
    getchar();
    return 0;
}
```

The screenshot shows a code editor with the file 'MCM.c' open. The code defines a function 'MatrixChainOrder' that takes an array 'p' and an integer 'n'. It uses a 2D array 'm' to store the minimum number of multiplications required for different subproblems. The function iterates over all possible subproblems of length 'L' from 2 to 'n', and for each length, it iterates over all possible starting indices 'i' and ending indices 'j'. For each subproblem, it calculates the minimum number of multiplications by trying all possible split points 'k'.

```

1  #include <limits.h>
2  #include <stdio.h>
3  int MatrixChainOrder(int p[], int n)
4  {
5      int m[n][n];
6      int i, j, k, L, q;
7      for (i = 1; i < n; i++)
8          m[i][i] = 0;
9      for (L = 2; L < n; L++) {
10         for (i = 1; i < n - L + 1; i++)
11             {
12                 j = i + L - 1;
13                 m[i][j] = INT_MAX;
14                 for (k = i; k <= j - 1; k++)
15                     {
16                         q = m[i][k] + m[k + 1][j]
17                           + p[i - 1] * p[k] * p[j];
18                         if (q < m[i][j])
19                             m[i][j] = q;
20                     }
21             }
22     }
23     return m[1][n - 1];
24 }

```

The status bar at the bottom indicates 'Ln 1, Col 1 Spaces: 4 UTF-8 CRLF C Win32'.

The screenshot shows the same code editor with the file 'MCM.c' open. The code defines a function 'main' that initializes an array 'arr' with the values {1, 2, 3, 4}, calculates the size of the array, and calls the 'MatrixChainOrder' function to find the minimum number of multiplications required for the given array. The result is printed using 'printf' and 'getchar' is used to pause the program.

```

12         j = i + L - 1;
13         m[i][j] = INT_MAX;
14         for (k = i; k <= j - 1; k++)
15             {
16                 q = m[i][k] + m[k + 1][j]
17                   + p[i - 1] * p[k] * p[j];
18                 if (q < m[i][j])
19                     m[i][j] = q;
20             }
21     }
22     return m[1][n - 1];
23 }
24
25 int main()
26 {
27     int arr[] = { 1, 2, 3, 4 };
28     int size = sizeof(arr) / sizeof(arr[0]);
29     printf("Minimum number of multiplications is %d ", MatrixChainOrder(arr, size));
30     getchar();
31     return 0;
32 }
33
34

```

The status bar at the bottom indicates 'Ln 34, Col 1 Spaces: 4 UTF-8 CRLF C Win32'.

**Output:-**

```
D:\VIT\Design and Analysis of Algorithms\lab>matrix_chain_multiplication.exe
Minimum number of multiplications is 18
```

**2. Implement Longest Common Subsequence.**

- if  $|X| = m$ ,  $|Y| = n$ , then there are  $2^m$  subsequences of  $x$ ; we must compare each with  $Y$  ( $n$  comparisons)
- So the running time of the brute-force algorithm is  $O(n 2^m)$
- Notice that the LCS problem has *optimal substructure*: solutions of subproblems are parts of the final solution.
- Subproblems: “find LCS of pairs of *prefixes* of  $X$  and  $Y$ ”

The longest common subsequence (LCS) is defined as the longest subsequence that is common to all the given sequences, provided that the elements of the subsequence are not required to occupy consecutive positions within the original sequences.

**Example:**

LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length 3.

LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4.

**LCS Length Algorithm**

LCS-Length( $X, Y$ )

1.  $m = \text{length}(X)$  // get the # of symbols in  $X$
2.  $n = \text{length}(Y)$  // get the # of symbols in  $Y$
3. for  $i = 1$  to  $m$        $c[i,0] = 0$       // special case:  $Y_0$
4. for  $j = 1$  to  $n$        $c[0,j] = 0$       // special case:  $X_0$
5. for  $i = 1$  to  $m$       // for all  $X_i$
6.     for  $j = 1$  to  $n$       // for all  $Y_j$
7.             if ( $X_i == Y_j$ )
8.                      $c[i,j] = c[i-1,j-1] + 1$
9.             else  $c[i,j] = \max(c[i-1,j], c[i,j-1])$
10. return  $c$

**Time Complexity:**  $O(mn)$

**Program :-**

```
#include <stdio.h>
#include <string.h>
int i, j, m, n, LCS_table[20][20];
char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];
void lcsAlgo() {
    m = strlen(S1);
    n = strlen(S2);
    for (i = 0; i <= m; i++)
        LCS_table[i][0] = 0;

    for (i = 0; i <= n; i++)
        LCS_table[0][i] = 0;
    for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++) {
            if (S1[i - 1] == S2[j - 1]) {
                LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
            } else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1]) {
                LCS_table[i][j] = LCS_table[i - 1][j];
            } else {
                LCS_table[i][j] = LCS_table[i][j - 1];
            }
        }
    int index = LCS_table[m][n];
    char lcsAlgo[index + 1];
    lcsAlgo[index] = '\0';
    int i = m, j = n;
    while (i > 0 && j > 0) {
        if (S1[i - 1] == S2[j - 1]) {
            lcsAlgo[index - 1] = S1[i - 1];
            i--;
            j--;
            index--;
        }
        else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
            i--;
        else
            j--;
    }
    printf("S1 : %s \nS2 : %s \n", S1, S2);
    printf("LCS: %s", lcsAlgo);
}

int main() {
    lcsAlgo();
    printf("\n");
}
```

```

1 #include <stdio.h>
2 #include <string.h>
3 int i, j, m, n, LCS_table[20][20];
4 char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];
5 void lcsAlgo() {
6     m = strlen(S1);
7     n = strlen(S2);
8     for (i = 0; i <= m; i++)
9         LCS_table[i][0] = 0;
10
11     for (i = 0; i <= n; i++)
12         LCS_table[0][i] = 0;
13     for (i = 1; i <= m; i++)
14         for (j = 1; j <= n; j++) {
15             if (S1[i - 1] == S2[j - 1]) {
16                 LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
17             } else if (LCS_table[i - 1][j] > LCS_table[i][j - 1]) {
18                 LCS_table[i][j] = LCS_table[i - 1][j];
19             } else {
20                 LCS_table[i][j] = LCS_table[i][j - 1];
21             }
22         }
23     int index = LCS_table[m][n];
24     char lcsAlgo[index + 1];

```

```

24 char lcsAlgo[index + 1];
25 lcsAlgo[index] = '\0';
26 int i = m, j = n;
27 while (i > 0 && j > 0) {
28     if (S1[i - 1] == S2[j - 1]) {
29         lcsAlgo[index - 1] = S1[i - 1];
30         i--;
31         j--;
32         index--;
33     }
34     else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
35         i--;
36     else
37         j--;
38 }
39 printf("S1 : %s \nS2 : %s \n", S1, S2);
40 printf("LCS: %s", lcsAlgo);
41 }
42
43 int main() {
44     lcsAlgo();
45     printf("\n");
46 }
47

```



**OUTPUT:-**

```
PS C:\Users\User\Desktop\c program\ALGO_LAB> gcc LCS.c -o LCS.EXE
```

```
PS C:\Users\User\Desktop\c program\ALGO_LAB> .\LCS
```

```
S1 : ACADB
```

```
S2 : CBDA
```

```
LCS: CB
```

```
char lcsAlgo[index + 1];
lcsAlgo[index] = '\0';
int i = m, j = n;
while (i > 0 && j > 0) {
    if (S1[i - 1] == S2[j - 1]) {
        lcsAlgo[index - 1] = S1[i - 1];
        i--;
        j--;
        index--;
    }
}
```

```
PS C:\Users\User\Desktop\c program\ALGO_LAB> gcc LCS.c -o LCS.EXE
PS C:\Users\User\Desktop\c program\ALGO_LAB> .\LCS
S1 : ACADB
S2 : CBDA
LCS: CB
PS C:\Users\User\Desktop\c program\ALGO_LAB> .\LCS
```

### 3. Implement Subset-sum.

Given a set of non-negative integers, and a value sum, determine if there is a subset of the given set with sum equal to given sum.

**Example:**

Suppose we are given  $n$ -distinct positive numbers  $w_i, i=1, 2, \dots, n$ .  
Our objective is to find out all subsets of these numbers whose sum is  $m$ .

Consider an example:

$(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$

$m = 31$

Solution:

$(w_1, w_2, w_4) = (11, 13, 7)$

$(w_3, w_4) = (24, 7)$

**Example:**

**Input:** set[] = {3, 34, 4, 12, 5, 2}, sum = 9

**Output:** True

There is a subset (4, 5) with sum 9.

**Input:** set[] = {3, 34, 4, 12, 5, 2}, sum = 30

**Output:** False

There is no subset that add up to 30.

### Algorithm:-

1. Sum of Subsets( $S, k, r$ )
2.  $k = 1, S = 0$
3.  $x[k] = 1$
4. If( $S + w[k] = m$ )
5. then write  $x[1:k]$
6. Else if( $S + w[k] + w[k+1] \leq m$ )
7. then Sum of Subsets( $S + w[k], k+1, r - w[k]$ )
8. If( $S + r - w[k] \leq m$ ) and ( $S + w[k+1] \leq m$ )
9.  $x[k] = 0$
10. Sum of Subsets( $S, k+1, r - w[k]$ )

**Program:**

```

#include <stdio.h>
int isSubsetSum(int set[], int n, int sum)
{
    int subset[n + 1][sum + 1];
    for (int i = 0; i <= n; i++)
        subset[i][0] = 1;
    for (int i = 1; i <= sum; i++)
        subset[0][i] = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= sum; j++) {
            if (j < set[i - 1])
                subset[i][j] = subset[i - 1][j];
            if (j >= set[i - 1])
                subset[i][j] = subset[i - 1][j]
                    || subset[i - 1][j - set[i - 1]];
        }
    }
    return subset[n][sum];
}

int main()
{
    int set[] = { 3, 34, 4, 12, 5, 2 };
    int sum = 9;
    int n = sizeof(set) / sizeof(set[0]);
    if (isSubsetSum(set, n, sum) == 1)
        printf("Found a subset with given sum");
    else
        printf("No subset with given sum");
    return 0;
}

```

```

c program > ALGO_LAB > C subset_sum.c > ...
1  #include <stdio.h>
2  int isSubsetSum(int set[], int n, int sum)
3  {
4      int subset[n + 1][sum + 1];
5      for (int i = 0; i <= n; i++)
6          subset[i][0] = 1;
7      for (int i = 1; i <= sum; i++)
8          subset[0][i] = 0;
9      for (int i = 1; i <= n; i++) {
10         for (int j = 1; j <= sum; j++) {
11             if (j < set[i - 1])
12                 subset[i][j] = subset[i - 1][j];
13             if (j >= set[i - 1])
14                 subset[i][j] = subset[i - 1][j]
15                     || subset[i - 1][j - set[i - 1]];
16         }
17     }
18     return subset[n][sum];
19 }
20 int main()
21 {
22     int set[] = { 3, 34, 4, 12, 5, 2 };
23     int sum = 9;
24     int n = sizeof(set) / sizeof(set[0]);

```

```

c program > ALGO_LAB > C subset_sum.c > ...
16     }
17     }
18     return subset[n][sum];
19 }
20 int main()
21 {
22     int set[] = { 3, 34, 4, 12, 5, 2 };
23     int sum = 9;
24     int n = sizeof(set) / sizeof(set[0]);
25     if (isSubsetSum(set, n, sum) == 1)
26         printf("\nFound a subset with given sum\n");
27     else
28         printf("\nNo subset with given sum");
29     return 0;
30 }
31
DEBUG CONSOLE  PROBLEMS 8  OUTPUT  TERMINAL
Found a subset with given sum
PS C:\Users\User\Desktop\c program\ALGO_LAB>

```

## OUTPUT:-

PS C:\Users\User\Desktop\c program\ALGO\_LAB> gcc subset\_sum.c

PS C:\Users\User\Desktop\c program\ALGO\_LAB> ./a.exe

Found a subset with given sum

```

c program > ALGO_LAB > C subset_sum.c > main()
21 {
22     int set[] = { 3, 34, 4, 12, 5, 2 };
23     int sum = 9;
24     int n = sizeof(set) / sizeof(set[0]);
25     if (isSubsetSum(set, n, sum) == 1)
26         printf("\nFound a subset with given sum\n");
27     else
28         printf("\nNo subset with given sum");
29     return 0;
30 }
+ ./a.out
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (./a.out:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\User\Desktop\c program\ALGO_LAB> gcc subset_sum.c
PS C:\Users\User\Desktop\c program\ALGO_LAB> ./a.exe

Found a subset with given sumPS C:\Users\User\Desktop\c program\ALGO_LAB> gcc subset_sum.c
PS C:\Users\User\Desktop\c program\ALGO_LAB> ./a.exe

Found a subset with given sum
PS C:\Users\User\Desktop\c program\ALGO_LAB>

```