Name: Rishi Tiku

Class: SE DS

Subject: DAA LAB

UID: 2021700067

Experiment – 4

Aim: To implement Dynamic Programming technique on LCS – Longest Common Subsequence Problem.

Theory:

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.

If S1 and S2 are the two given sequences then, Z is the common subsequence of S1 and S2 if Z is a subsequence of both S1 and S2. Furthermore, Z must be a strictly increasing sequence of the indices of both S1 and S2.

In a strictly increasing sequence, the indices of the elements chosen from the original sequences must be in ascending order in Z.

Algorithm:

1) Naïve:

```
function LCS(S, T)
  if S is empty or T is empty then
    return empty string
  if first char of S == first char of T then
```

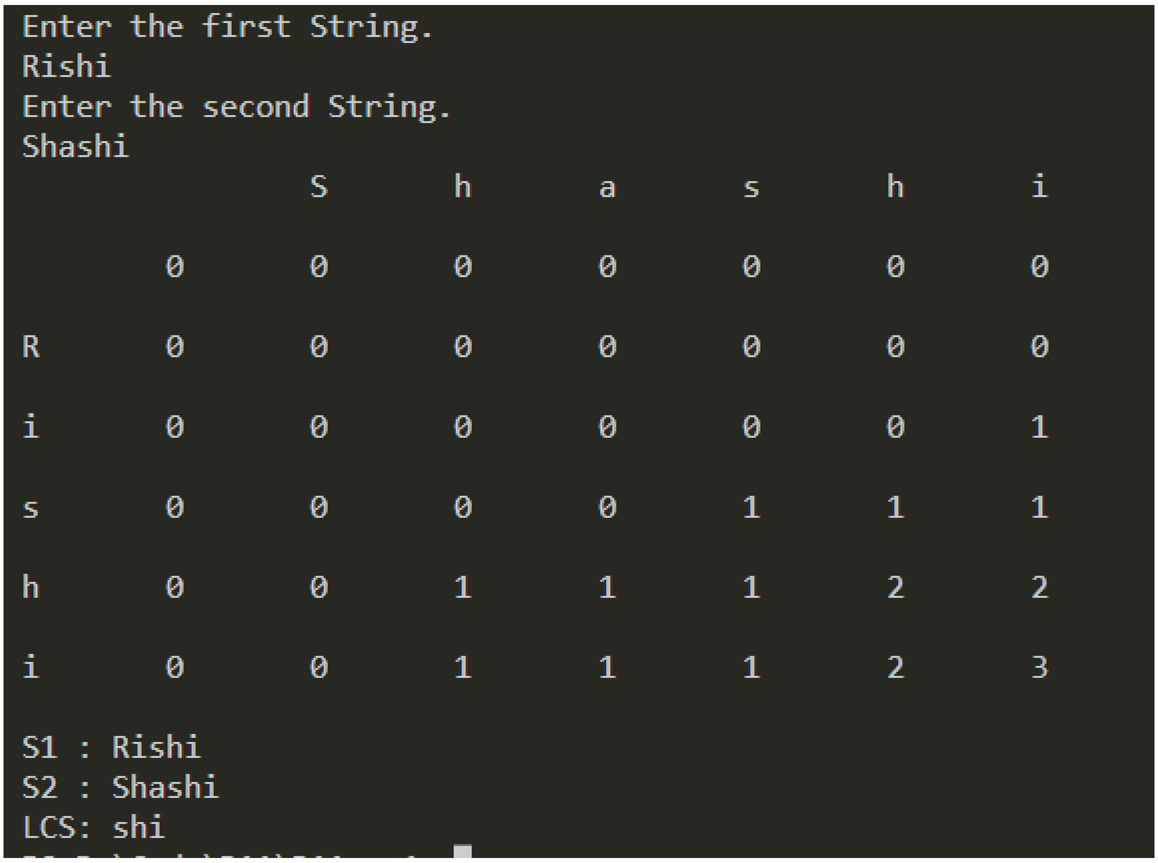
```
        return (first char of S) + LCS(S - first char, T - first char)
    otherwise // first chars are different
        return longer of LCS(S - first char, T) and LCS(S, T - first char)
```

2) Dynamic Programming
   a. Create a 2D array dp[][] with rows and columns equal to the length of each input string plus 1 [the number of rows indicates the indices of S1 and the columns indicate the indices of S2].
   b. Initialize the first row and column of the dp array to 0.
   c. Iterate through the rows of the dp array, starting from 1 (say using iterator i).
   d. For each i, iterate all the columns from j = 1 to n:
   e. If S1[i-1] is equal to S2[j-1], set the current element of the dp array to the value of the element to (dp[i-1][j-1] + 1).
   f. Else, set the current element of the dp array to the maximum value of dp[i-1][j] and dp[i][j-1].
   g. After the nested loops, the last element of the dp array will contain the length of the LCS

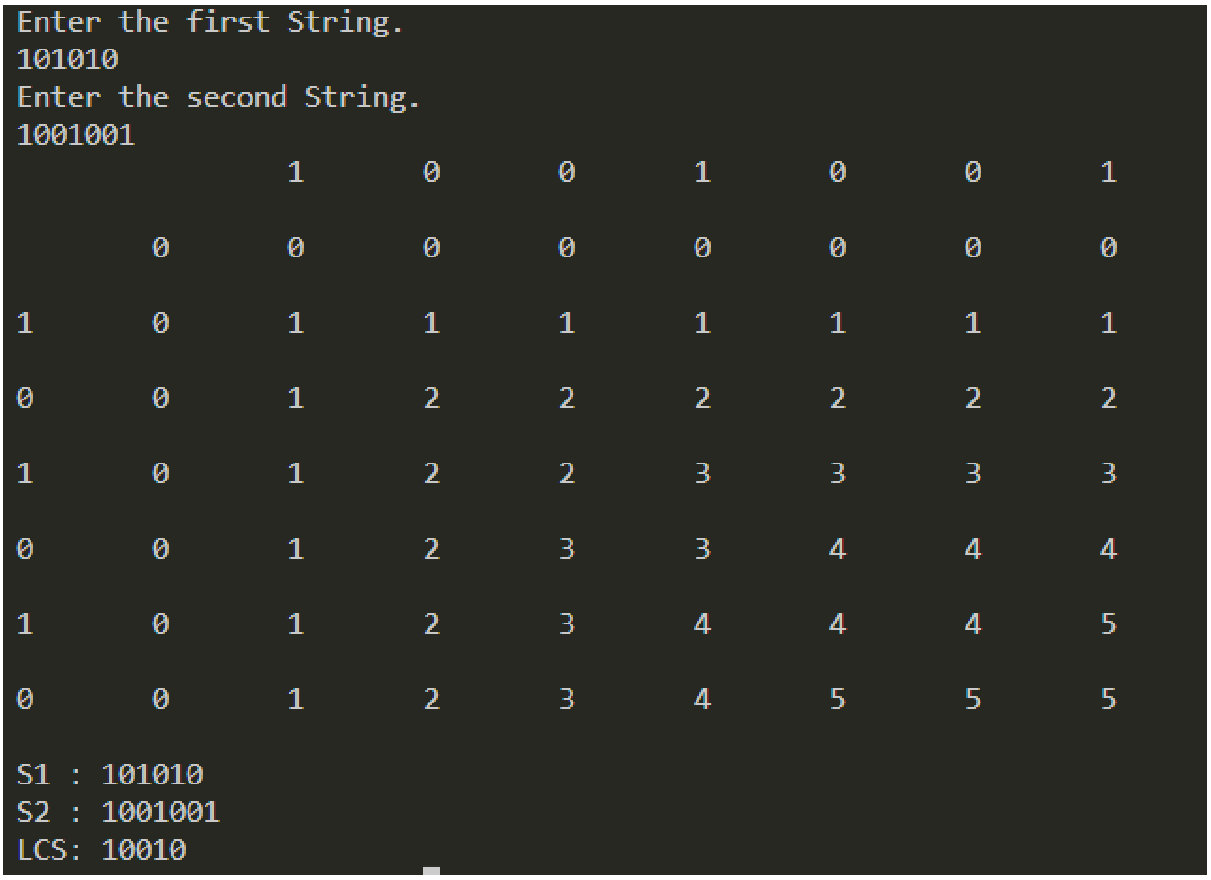
Code:

```c
#include <stdio.h>
#include <string.h>

int i, j, m, n, LCS_table[20][20];
char S1[20], S2[20];

void print()
{
  printf("\t");
  for(int k = 0; k<=n; k++)
    printf(" \t%c", S2[k]);
  printf("\n\n");
  for(int k = 0; k<=n; k++)
    printf("\t%d", LCS_table[k][0]);
  printf("\n\n");

  for(int k = 1; k<=m; k++)
  {
    printf("%c\t", S1[k-1]);
    for(int l = 0; l<=n; l++)
    {
      printf("%d\t", LCS_table[k][l]);
    }
    printf("\n\n");
  }
}

void lcsAlgo() {
  m = strlen(S1);
  n = strlen(S2);

  // Filling 0's in the matrix
  for (i = 0; i <= m; i++)
    LCS_table[i][0] = 0;
  for (i = 0; i <= n; i++)
    LCS_table[0][i] = 0;

  // Building the mtrix in bottom-up way
  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';

  print();
  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--;
  }

  // Printing the sub sequences
  printf("S1 : %s \nS2 : %s \n", S1, S2);
  printf("LCS: %s", lcsAlgo);
}

int main() {
  printf("Enter the first String.\n");
  scanf("%s", S1);
  printf("Enter the second String.\n");
  scanf("%s", S2);
  lcsAlgo();
  printf("\n");
}
```

Output:



```
Enter the first String.
Rishi
Enter the second String.
Shashi
                S       h       a       s       h       i

        0       0       0       0       0       0       0

R       0       0       0       0       0       0       0

i       0       0       0       0       0       0       1

s       0       0       0       0       1       1       1

h       0       0       1       1       1       2       2

i       0       0       1       1       1       2       3

S1 : Rishi
S2 : Shashi
LCS: shi
```



```
Enter the first String.
101010
Enter the second String.
1001001
                1       0       0       1       0       0       1

        0       0       0       0       0       0       0       0

1       0       1       1       1       1       1       1       1

0       0       1       2       2       2       2       2       2

1       0       1       2       2       3       3       3       3

0       0       1       2       3       3       4       4       4

1       0       1       2       3       4       4       4       5

0       0       1       2       3       4       5       5       5

S1 : 101010
S2 : 1001001
LCS: 10010
```

Conclusion:

Dynamic Programming technique is implemented for LCS problem. Its time complexity is O (m * n) which is much faster than naive solution O (2 ^ n).