|  |  |
|---|---|
| **Name** | Parth Gandhi |
| **UID No.** | 2021300033 |
| **Class & Division** | S.E. COMPS A (BATCH B) |
| **Experiment No.** | 4 |

**Aim:** Experiment on finding LCS using dynamic programming.

**Theory:**
LCS stands for Longest Common Subsequence, which is a classic problem in computer science and is used in various applications such as bioinformatics, data mining, and version control systems. Given two sequences of characters, the problem is to find the longest subsequence that is common to both sequences. A subsequence of a sequence is a new sequence that is formed by deleting some (or none) of the elements from the original sequence, without changing the order of the remaining elements. For example, if we have the sequences "ABCD" and "ACDF", then the longest common subsequence is "ACD", since it appears in both sequences and has the maximum length among all common subsequences.
The LCS problem can be solved using dynamic programming. We can define a 2D table where each cell represents the length of the longest common subsequence between two prefixes of the input sequences. We can fill in the table by comparing each character of the two sequences and using the previously computed values to build the final solution.

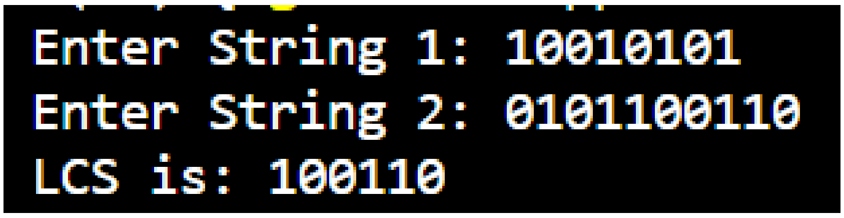
**Algorithm:**
1. Create an (n+1) x (m+1) table, where n and m are the lengths of the input sequences.
2. Initialize the first row and column to 0.
3. For each i from 1 to n and for each j from 1 to m: a. If s1[i] == s2[j], set table[i][j] = table[i-1][j-1] + 1. b. Otherwise, set table[i][j] = max(table[i-1][j], table[i][j-1]).
4. The value of table[n][m] represents the length of the longest common subsequence.
5. To reconstruct the subsequence, we can use a similar approach as the one used for printing the optimal sequence of matrix multiplications in the Matrix Chain Multiplication problem.

**Code:**
```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string a, b;
    cout << "Enter String 1: ";
    cin >> a;
    cout << "Enter String 2: ";
    cin >> b;
    vector<vector<int>> mat(a.size() + 1, vector<int>(b.size() + 1, 0));
    for (int i = 1; i < a.size() + 1; i++)
    {
        for (int j = 1; j < b.size() + 1; j++)
        {
            if (a[i - 1] == b[j - 1])
            mat[i][j] = mat[i - 1][j - 1] + 1;
            else
            mat[i][j] = max(mat[i - 1][j], mat[i][j - 1]);
        }
    }
    string lcs = "";
    int i = a.size();
    int j = b.size();
    while (i != 0 && j != 0)
    {
        if (a[i - 1] == b[j - 1])
        {
            lcs = b[j - 1] + lcs;
            i -= 1;
            j -= 1;
        }
        else
        {
            if (mat[i][j] == mat[i - 1][j])
                i -= 1;
            else
                j -= 1;
        }
    }
    cout << "LCS is: " << lcs << endl;

    return 0;
}
```

**<u>Output:</u>**



```
Enter String 1: 10010101
Enter String 2: 0101100110
LCS is: 100110
```

**<u>Conclusion:</u>** Successfully wrote a program to implement LCS using dynamic programming approach.