Longest Common Subsequence (LCS)

Given two strings s1 and s2, find the length of the longest subsequence which is common in both strings.

S1 = "**A**C**ED**B**F**D" AEDF
S2 = "**AE**C**DF**" LCS: 4

S1 = "**A**BC**D**G**H**" ADH
S2 = "**A**E**D**F**H**R" LCS: 3

S1 = "A**G**G**TAB**" GTAB
S2 = "**G**X**T**G**A**Y**B**" LCS: 4

S1 = "NAD**EEM**" Counter: I = 5 LCS: 0
S2 = "NA**EEM**" Counter: j = 4

Application:
- Bio-Infomratics
- Molecular Biology (DNA)
- File Comparison
- Screen redisplay

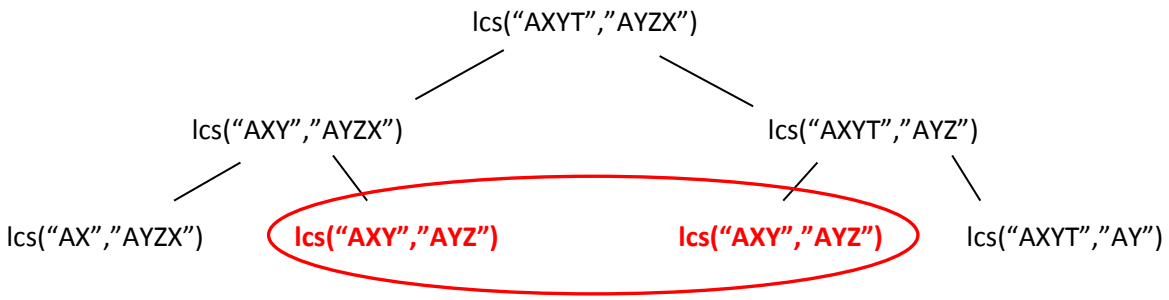
Case 1: Increment length of LCS by 1 and process S1[m-1] and S2[n-1], where m is the length of S1 and n is the length of S2.

Case 2:

Case 1: S1[i] == S2[j] LCS++ S1[i-1] and S2[j-1], i=4, j=3
Case 1: S1[i] == S2[j] LCS++ S1[i-1] and S2[j-1], i=3, j=2
Case 1: S1[i] == S2[j] LCS++ S1[i-1] and S2[j-1], i=2, j=1

Lcs(S1,S2,i,j)
1 + Lcs(S1,S2,i-1,j-1)
1 + 1 + Lcs(S1,S2,i-2,j-2)
2 + 1 + Lcs(S1,S2,i-3,j-3) …

Case 2: S1[i] != S2[j] LCS(S1,S2,i,j) = max(LCS(S1,S2,i,j-1), LCS(S1,S2,i-1,j))



Case 1:

lcs(S1,S2,i,j) = 1 + *lcs(S1,S2,i-1,j-1)*

Case 2:

lcs(S1,S2,i,j) = $\max \begin{cases} lcs(S1, S2, i, j-1) \\ lcs(S1, S2, i-1, j) \end{cases}$

lcs(S1,S2,i,j) = $\begin{cases} 0 \quad if\ i \leq 0\ or\ j \leq 0 \\ 1 + lcs(S1, S2, i-1, j-1)\ if\ S1[i] = S2[j]) \\ max \begin{cases} lcs(S1, S2, i, j-1)\ if\ S1[i] \neq S2[j] \\ lcs(S1, S2, i-1, j) \end{cases} \end{cases}$

S1 = "A**G**G**TAB**" m = 6 Table: lcs[m+1][n+1]

S2 = "**G**X**T**G**A**Y**B**" n = 7

lcs	ø	A	G	G	T	A	B
Ø	0	0	0	0	0	0	0
G	0	0	1	1	1	1	1
X	0	0	1	1	1	1	1
T	0	0	1	1	2	2	2
G	0	0	1	1	2	2	2
A	0	1	1	1	2	3	3
Y	0	1	1	1	2	3	3
B	0	1	1	1	2	3	4

- Fill 1st row and 1st column with ZEROs
- Move to the next row and compare G with each column.
  - If there is no match then select the max(leftCell,aboveCell).
  - If it matches, store diagonalCellValue + 1.

Bottom Right Corner: 4
Is it MAX of upper and left cells? NO. It means it is part of the LCS. Select B and move up diagonally.
Result: B.

Current Cell value: 3
Is it MAX of upper and left cells? YES. Because upper cell has the same value as the current cell, move up.

Current Cell value: 3
Is it MAX of upper and left cells? NO. It means it is part of the LCS. Select A and move diagonally. Result: AB.

Current Cell value: 2
Is it MAX of upper and left cells? YES. Because upper cell has the same value as the current cell, move up.

Current cell value: 2
Is it MAX of upper and left cells? NO. It means it is part of the LCS. Select T and move diagonally. Result: TAB.

Current cell value: 1
Is it MAX of upper and left cells? YES. Because the upper cell value and the left cell value, both are same as the current cell value, we can move to either direction. Let us move up.

Current cell value: 1
Is it MAX of upper and left cells? YES. Because left cell has the same value as the current cell, move left.

Current cell value: 1
Is it MAX of upper and left cells? NO. It means it is part of the LCS. Select G and move diagonally. Result: GTAB. LCS: 4.

Recursive Implementation:

```
// A Naive C++ recursive implementation
// of LCS of two strings
#include <bits/stdc++.h>
using namespace std;

// Returns length of LCS for X[0..m-1], Y[0..n-1]
int lcs(string X, string Y, int m, int n)
{
   if (m == 0 || n == 0)
      return 0;

   if (X[m - 1] == Y[n - 1])
      return 1 + lcs(X, Y, m - 1, n - 1);
   else
      return max(lcs(X, Y, m, n - 1),
               lcs(X, Y, m - 1, n));
}

// Driver Code
int main()
{
   string X = "AGGTAB";
   string Y = "GXTXAYB";

   // Find the length of string
   int m = X.length();
   int n = Y.length();

   cout << "Length of LCS: " << lcs(X, Y, m, n);

   return 0;
```

```
}
```

Dynamic Programming:

```
// C++ program to memoize
// recursive implementation of LCS problem
#include <bits/stdc++.h>
using namespace std;

const int maximum = 1000;

// Returns length of LCS for X[0..m-1], Y[0..n-1] */
// memoization applied in recursive solution
int lcs(string X, string Y, int m, int n, int dp[][maximum])
{
   // base case
   if (m == 0 || n == 0)
      return 0;

   // if the same state has already been
   // computed
   if (dp[m - 1][n - 1] != -1)
      return dp[m - 1][n - 1];

   // if equal, then we store the value of the
   // function call
   if (X[m - 1] == Y[n - 1]) {

      // store it in arr to avoid further repetitive
      // work in future function calls
      dp[m - 1][n - 1] = 1 + lcs(X, Y, m - 1, n - 1, dp);

      return dp[m - 1][n - 1];
   }
   else {

      // store it in arr to avoid further repetitive
      // work in future function calls
      dp[m - 1][n - 1] = max(lcs(X, Y, m, n - 1, dp),
                             lcs(X, Y, m - 1, n, dp));

      return dp[m - 1][n - 1];
   }
}

// Driver Code
int main()
{
```

```
    string X = "AGGTAB";
    string Y = "GXTXAYB";
    int m = X.length();
    int n = Y.length();

    int dp[m][maximum];

    // assign -1 to all positions
    memset(dp, -1, sizeof(dp));

    cout << "Length of LCS: " << lcs(X, Y, m, n, dp);

    return 0;
}
```