Longest common subsequence (LCS): Biological applications often need to compare the DNA of two (or more) different organisms. A strand of DNA consists of a string of molecules called bases. We formalize this last notion of similarity as the longest-common-subsequence problem. A subsequence of a given sequence is just the given sequence of zero or more elements left out.

Ex- $S = (A, B, B, A, B, B)$
positions: 1 2 3 4 5 6

$SS_1 = (B, B, B, B)$ (2 3 5 6), $SS_2 = (A, A)$ (1 4), $SS_3 = (A, A, B, B, B)$ (1 4 5 6 2) X

- **Common Subsequence:** Z is said to be common subsequence of two sequence of X & Y iff where Z is subsequence to both X & Y.

Ex

$X = (A, B, B, A, B, B)$
positions: 1 2 3 4 5 6

$Y = (B, A, A, B, A, A)$
positions: 1 2 3 4 5 6

$CS_1 = (A, A)$, $CS_2 = (B, B, B)$ X, $CS_3 = (A)$
$CS_4 = (A, B, A)$

- **Longest common subsequence:** It's a longest common subsequence from all the available common subsequences.

- This section shows how to efficiently solve the LCS problem using dynamic programming.

→ In a brute-force approach to solving the LCS problem, we should enumerate all subsequences of X and check each subsequence to see whether it is also a subsequence of Y, keeping track of the longest subsequence we find. Each subsequence of X corresponds to a subset of the indices $\{1, 2, \ldots, m\}$ of X. Because X has 2^m subsequences, this approach requires exponential time.

→ Recursive solution:

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0, \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ \& } x_i = y_j, \\ \max(c[i,j-1], c[i-1,j]) & \text{if } i,j>0 \text{ \& } x_i \neq y_i, \end{cases}$$

Procedure LCS-Length takes two sequence $X=\{x_1, x_2, \ldots x_m\}$ and $Y=\{y_1, y_2, \ldots y_n\}$ as inputs. It stores the $c[i,j]$ value in a table $c[0\ldots m, 0\ldots n]$, and it computes the entries in row major order. The procedure also maintains the table $b[1\ldots m, 1\ldots n]$ to help us construct an optimal solution.

```
LCS-Length ( X, Y)
1. m = X.length
2. n = Y.length
3. Let b[1...m, 1...n] and c[0...m, 0...n] be new tables
4. For i = 1 to m
5. c[i, 0] = 0
6. For j = 0 to n
7. c[0, j] = 0
```

```
8.  For i = 1 to m
9.      For j = 1 to n
10.         If xi == yj
11.             c[i,j] = c[i-1,j-1]+1
12.             b[i,j] = "↖"
13.     Else if c[i-1,j] >= c[i,j-1]
14.             c[i,j] = c[i-1,j]
15.             b[i,j] = "↑"
16.     Else    c[i,j] = c[i,j-1]
17.             b[i,j] = "←"
18.     return c and b
```

We encounter the elements of this LCS in reverse order with this method. The following recursive procedure print out an LCS of X & Y in proper, forward order.

```
Print_LCS(b, X, i, j)
1. If i==0 or j==0
2.     return
3. If b[i,j] == "↖"
4. Print_LCS(b, X, i-1, j-1)
5.  Print xi
6. else if b[i,j] == "↑"
7. Print_LCS(b, X, i-1, j)
8. else Print_LCS(b, X, i, j-1)
```

Example

$X = \{A, B, C, B, D, A, B\}$

$Y = \{B, D, C, A, B, A\}$

c

i \ j	0	1	2	3	4	5	6
	yj	B	D	C	A	B	A
0	xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	↑	←	1	1	2	2
3	C	0	1	1	↖	←	2	2
4	B	0	1	1	2	2	↖	3
5	D	0	1	2	2	2	↑	3
6	A	0	1	2	2	3	3	↖
7	B	0	1	2	2	3	4	↑

b

	1	2	3	4	5	6
1	↑	↑	↑	↖	←	↖
2	↖	←	←	↑	↖	←
3	↑	↑	↑	←	↑	↑
4	↖	↑	↑	↑	↖	←
5	↑	↖	↖	↑	↑	↑
6	↑	↑	↑	↖	↑	↖
7	↖	↑	↑	↑	↖	↑

$X = \{A, B, C, B, D, A, B\}$

$Y = \{B, D, C, A, B, A\}$

$C[1,1] = C[0,1] \Rightarrow 0$

$C[1,2] = C[0,2] \Rightarrow 0$

$C[1,3] = C[0,3] \Rightarrow 0$

$C[1,4] = C[0,3] + 1 \Rightarrow 0 + 1 \Rightarrow 1$

$C[1,5] = C[1,4] \Rightarrow 1$

$C[1,6] = C[1,5] \Rightarrow 1$

$C[2,1] = C[1,0] + 1 \Rightarrow 1$

$C[2,2] = \max\{C[1,2], C[2,1]\} \Rightarrow 1$

$C[2,3] = \max\{C[2,2], C[1,3]\} \Rightarrow 1$

$C[2,4] = \max\{C[1,4], C[2,3]\} \Rightarrow 1$

$C[2,5] = C[1,4] + 1 \Rightarrow 1 + 1 \Rightarrow 2$

$C[2,6] = \max\{C[1,6], C[2,5]\} \Rightarrow 2$

$C[3,1] = \max(C[2,1], C[3,0]) \Rightarrow 1$

$C[3,2] = \max(C[2,2], C[3,1]) \Rightarrow 1$

$C[3,3] = C[2,2] + 1 \Rightarrow 1 + 1 \Rightarrow 2$

$C[3,4] = \max(C[2,4], C[3,3]) \Rightarrow 2$

$c[3,5] = \max(c[2,5], c[3,4]) \Rightarrow 2$

$c[3,6] = \max(c[2,6], c[3,5]) \Rightarrow 2$

$c[4,1] = c[3,0]+1 \Rightarrow 0+1 \Rightarrow 1$

$c[4,2] = \max(c[3,2], c[4,1]) \Rightarrow 1$

$c[4,3] = \max(c[3,3], c[4,2]) \Rightarrow 2$

$c[4,4] = \max(c[3,4], c[4,3]) \Rightarrow 2$

$c[4,5] = c[3,4]+1 \Rightarrow 2+1 \Rightarrow 3$

$c[4,6] = \max(c[3,6], c[4,5]) \Rightarrow 3$

$c[5,1] = \max(c[4,1], c[5,0]) \Rightarrow 1$

$c[5,2] = c[4,1]+1 \Rightarrow 2$

$c[5,3] = \max(c[4,3], c[5,2]) \Rightarrow 2$

$c[5,4] = \max(c[4,4], c[5,3]) \Rightarrow 2$

$c[5,5] = \max(c[4,5], c[5,4]) \Rightarrow 3$

$c[5,6] = \max(c[4,6], c[5,5]) \Rightarrow 3$

$c[6,1] = \max(c[5,1], c[6,0]) \Rightarrow 1$

$c[6,2] = \max(c[5,2], c[6,1]) \Rightarrow 2$

$c[6,3] = \max(c[5,3], c[6,2]) \Rightarrow 2$

$c[6,4] = c[5,3]+1 \Rightarrow 2+1 \Rightarrow 3$

$c[6,5] = \max(c[5,5], c[6,4]) \Rightarrow 3$

$c[6,6] = c[5,5]+1 \Rightarrow 3+1 \Rightarrow 4$

$c[7,1] = c[6,0]+1 \Rightarrow 0+1 \Rightarrow 1$

$c[7,2] = \max(c[6,2], c[7,1]) \Rightarrow 2$

$c[7,3] = \max(c[6,3], c[7,2]) \Rightarrow 2$

$c[7,4] = \max(c[6,4], c[7,3]) \Rightarrow 3$

$c[7,5] = c[6,4]+1 \Rightarrow 3+1 \Rightarrow 4$

$c[7,6] = \max(c[6,6], c[7,5]) \Rightarrow 4$

TC = # of distinct function calls = $m+n \Rightarrow O(m+n)$