Longest common subsequence:

Let us understand what is longest common subsequence by considering a real world example, 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, where the possible bases are adenine, guanine, cytosine, and thymine. Representing each of these bases by its initial letters. the DNA of one organism may be S1 = "ACCGGTCGAGTGCGCGGAAGCCGGCCGAA", and the DNA of another organism may be S2 = "GTCGTTCGGAATGCCGTTGCTCTGTAAA". One reason to compare two strands of DNA is to determine how "similar" the two strands are. We can say that two DNA strands are similar if one is a substring of the other. In our example, neither S1 nor S2 is a substring of the other. Alternatively, we could say that two strands are similar if the number of changes needed to turn one into the other is small. Yet another way to measure the similarity of strands S1 and S2 is by finding a third strand S3 in which the bases in S3 appear in each of S1 and S2; these bases must appear in the same order, but not necessarily consecutively. The longer the strand S3 we can find, the more similar S1 and S2 are. In our example, the longest strand S3 is "GTCGTCGGAAGCCGGCCGAA". We formalize this last notion of similarity as the longest-common-subsequence problem.

A subsequence of a given sequence is just the given sequence with zero or more elements left out. Formally, given a sequence X = {X_1, X_2,……., X_m}, another sequence Z = { Z_1,Z_2,…..,Z_k } is a subsequence of X if there exists a strictly increasing sequence {i_1,i_2,…..,i_k} of indices of X such that for all j = 1, 2, ……. ,k we have xij = Zj .

For example, Z = {B, C, D, B} is a subsequence of X = {A, B, C, B, D, A, B} with corresponding index sequence {2, 3, 5, 7}.

Given two sequences X and Y, we say that a sequence Z is a common sub-sequence of X and Y if Z is a subsequence of both X and Y.

For example, if X = {A, B, C, B, D, A, B} and Y = {B, D, C, A, B, A}, the sequence {B, C, A} is a common subsequence of both X and Y. The sequence {B, C, A} is not a longest common subsequence (LCS) of X and Y, however, since it has length 3 and the sequence {B, C, B, A}, which is also common to both X and Y, has length 4. The sequence {B, C, B, A} is an LCS of X and Y, as is the sequence {B, D, A, B}, since X and Y have no common subsequence of length 5 or greater.

Characterizing a longest common subsequence:

In a brute-force approach to solving the LCS problem, we would 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, ……. ,m} of X . Because X has 2^m subsequences, this approach requires exponential time, making it impractical for long sequences.

Theorem .1 (Optimal substructure of an LCS)

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

Proof (1) If $z_k \neq x_m$, then we could append $x_m = y_n$ to Z to obtain a common subsequence of X and Y of length $k + 1$, contradicting the supposition that Z is a *longest* common subsequence of X and Y. Thus, we must have $z_k = x_m = y_n$. Now, the prefix Z_{k-1} is a length-$(k - 1)$ common subsequence of X_{m-1} and Y_{n-1}. We wish to show that it is an LCS. Suppose for the purpose of contradiction that there exists a common subsequence W of X_{m-1} and Y_{n-1} with length greater than $k - 1$. Then, appending $x_m = y_n$ to W produces a common subsequence of X and Y whose length is greater than k, which is a contradiction.

(2) If $z_k \neq x_m$, then Z is a common subsequence of X_{m-1} and Y. If there were a common subsequence W of X_{m-1} and Y with length greater than k, then W would also be a common subsequence of X_m and Y, contradicting the assumption that Z is an LCS of X and Y.

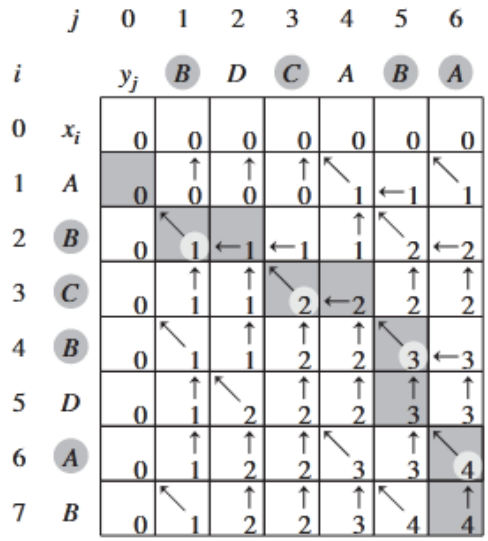
(3) The proof is symmetric to (2).

Computing the length of an LCS:

We could easily write an exponential-time recursive algorithm to compute the length of an LCS of two sequences. Since the LCS problem has only, Θ(mn) distinct sub-problems, however, we can use dynamic programming to compute the solutions bottom up.

Procedure LCS-L ENGTH takes two sequences X = {x1, x2, ……, xm} and Y = {y1, y2,….., yn} as inputs. It stores the c[i, j] • values in a table c[0 … m, 0 … n] ,and it computes the entries in row-major order. The procedure also maintains the table b [1 … m, 1 … n] to help us construct an optimal solution. Intuitively, b[i, j] points to the table entry corresponding to the optimal sub-problem solution chosen when computing c[i, j]. The procedure returns the b and c tables; c[m, n] contains the length of an LCS of X and Y .

```
LCS-LENGTH(X, Y)
    m = X.length
    n = Y.length
    let b[1..m, 1..n] and c[0..m, 0..n] be new tables
    for i = 1 to m
        c[i, 0] = 0
    for j = 0 to n
        c[0, j] = 0
    for i = 1 to m
        for j = 1 to n
            if x_i == y_j
                c[i, j] = c[i − 1, j − 1] + 1
                b[i, j] = "↖"
            elseif c[i − 1, j] ≥ c[i, j − 1]
                c[i, j] = c[i − 1, j]
                b[i, j] = "↑"
            else c[i, j] = c[i, j − 1]
                b[i, j] = "←"
    return c and b
```



	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Figure shows the tables produced by LCS-L ENGTH on the sequences X = {A, B, C, B, D, A, B} and Y = {B, D, C, A, B, A}. The running time of the procedure is , Θ{mn}, since each table entry takes , Θ (1) time to compute.

Constructing an LCS:

The b table returned by LCS-LENGTH enables us to quickly construct an LCS of X = {x1, x2, …. , xm} and Y = {y1, y2, …. , yn}. We simply begin at b[m, n] and trace through the table by following the arrows. Whenever we encounter a "↖" in entry b[i, j], it implies that xi = yj is an element of the LCS that LCS-LENGTH found. With this method, we encounter the elements of this LCS in reverse order.
The following recursive procedure prints out an LCS of X and Y in the proper,
forward order. The initial call is PRINT-LCS (b, X, X.length, Y.length). The procedure takes time Θ(m+n), since it decrements at least one of i and j in each recursive call.

```
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 x_i
6  elseif b[i, j] == "↑"
7      PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```