Longest Common Subsequence

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`.

If

```
S1 = {B, C, D, A, A, C, D}
```

Then, `{A, D, B}` cannot be a subsequence of `S1` as the order of the elements is not the same (ie. not strictly increasing sequence).

Let us understand LCS with an example.

If

```
S1 = {B, C, D, A, A, C, D}
S2 = {A, C, D, B, A, C}
```

Then, common subsequences are

`{B, C}, {C, D, A, C}, {D, A, C}, {A, A, C}, {A, C}, {C, D},`

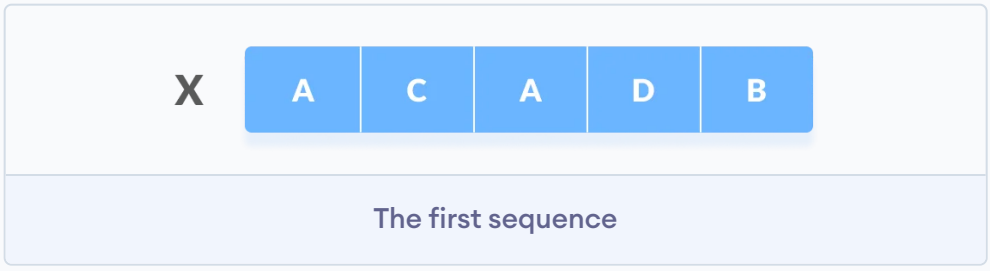
`...`

Among these subsequences, `{C, D, A, C}` is the longest common subsequence. We are going to find this longest common subsequence using dynamic programming.

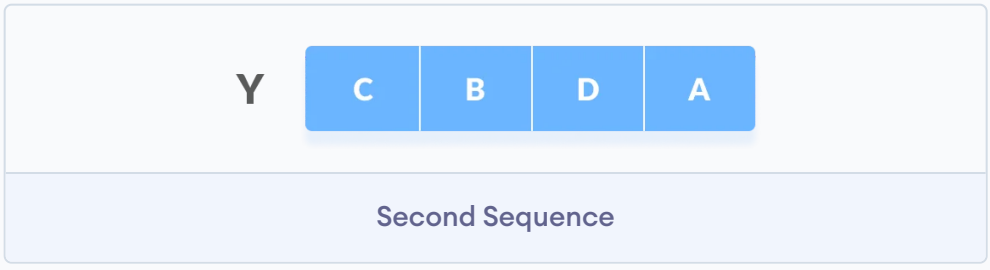
Before proceeding further, if you do not already know about dynamic programming, please go through [dynamic programming (/dsa/dynamic-programming)](/dsa/dynamic-programming).

Using Dynamic Programming to find the LCS

Let us take two sequences:



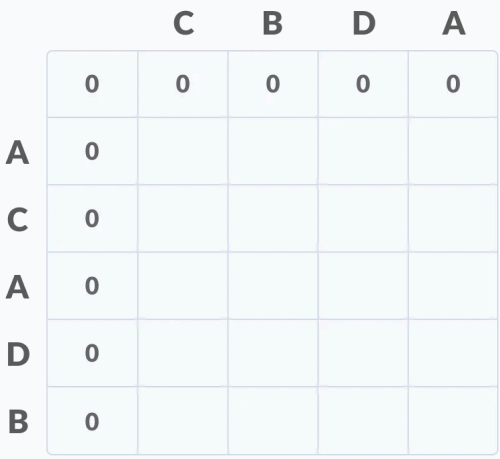
The first sequence



Second Sequence

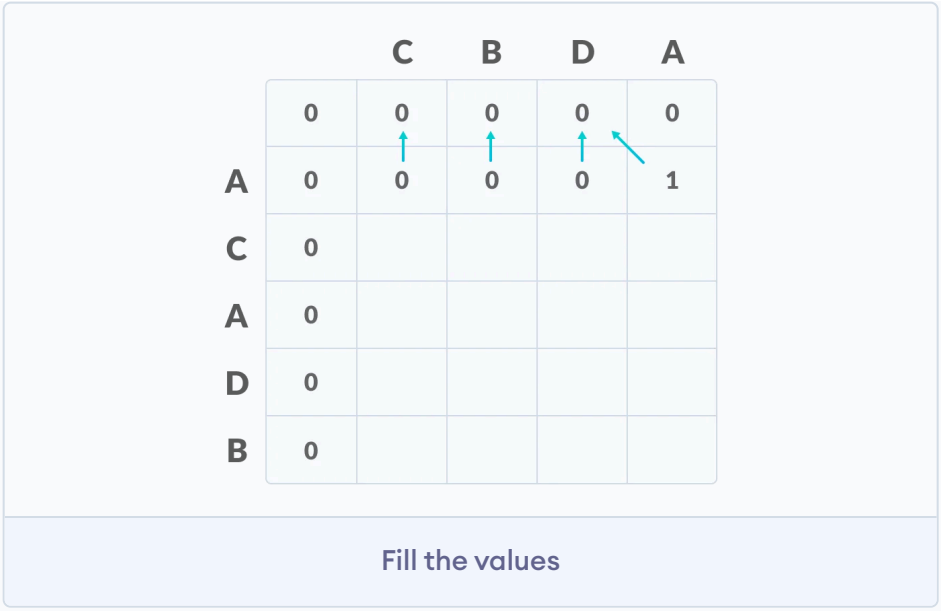
The following steps are followed for finding the longest common subsequence.

1. Create a table of dimension `n+1*m+1` where n and m are the lengths of `X` and `Y` respectively. The first row and the first column are filled with zeros.



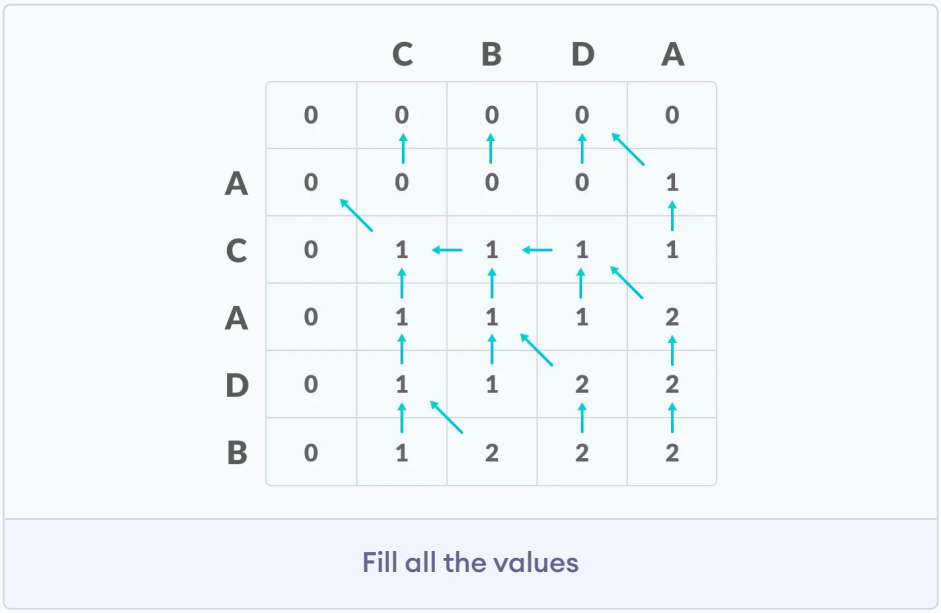
Initialise a table

2. Fill each cell of the table using the following logic.
3. If the character correspoding to the current row and current column are matching, then fill the current cell by adding one to the diagonal element. Point an arrow to the diagonal cell.
4. Else take the maximum value from the previous column and previous row element for filling the current cell. Point an arrow to the cell with maximum value. If they are equal, point to any of them.



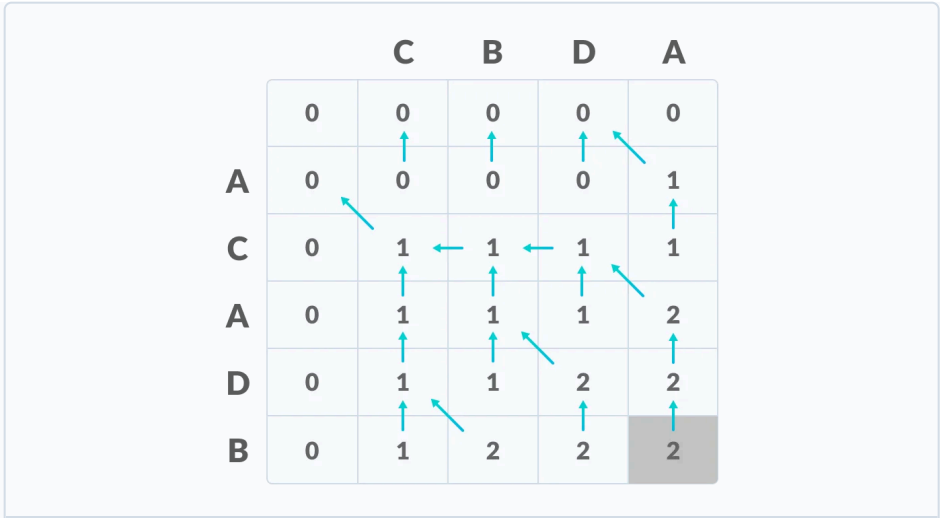
Fill the values

5. **Step 2** is repeated until the table is filled.



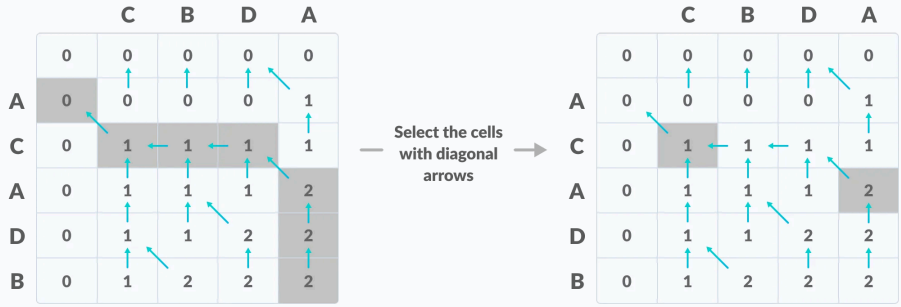
Fill all the values

6. The value in the last row and the last column is the length of the longest common subsequence.



The bottom right corner is the length of the LCS

7. In order to find the longest common subsequence, start from the last element and follow the direction of the arrow. The elements corresponding to () symbol form the longest common subsequence.



Create a path according to the arrows

Thus, the longest common subsequence is `CA`.

C A

LCS

How is a dynamic programming algorithm more efficient than the recursive algorithm while solving an LCS problem?

The method of dynamic programming reduces the number of function calls. It stores the result of each function call so that it can be used in future calls without the need for redundant calls.

In the above dynamic algorithm, the results obtained from each comparison between elements of `X` and the elements of `Y` are stored in a table so that they can be used in future computations.

So, the time taken by a dynamic approach is the time taken to fill the table (ie. O(mn)). Whereas, the recursion algorithm has the complexity of $2^{max(m, n)}$.

Longest Common Subsequence Algorithm

```
X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y
LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
    If X[i] = Y[j]
        LCS[i][j] = 1 + LCS[i-1, j-1]
        Point an arrow to LCS[i][j]
    Else
        LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
        Point an arrow to max(LCS[i-1][j], LCS[i][j-1])
```

Python, Java and C/C++ Examples

Python | Java | C | C++

```
# The longest common subsequence in Python


# Function to find lcs_algo
def lcs_algo(S1, S2, m, n):
    L = [[0 for x in range(n+1)] for x in range(m+1)]

    # Building the mtrix in bottom-up way
    for i in range(m+1):
        for j in range(n+1):
            if i == 0 or j == 0:
                L[i][j] = 0
            elif S1[i-1] == S2[j-1]:
                L[i][j] = L[i-1][j-1] + 1
            else:
                L[i][j] = max(L[i-1][j], L[i][j-1])

    index = L[m][n]

    lcs_algo = [""] * (index+1)
    lcs_algo[index] = ""

    i = m
    j = n
    while i > 0 and j > 0:

        if S1[i-1] == S2[j-1]:
            lcs_algo[index-1] = S1[i-1]
```

Longest Common Subsequence Applications

1. in compressing genome resequencing data
2. to authenticate users within their mobile phone through in-air signatures