

Longest Common Subsequence(LCS)

*subsequence mean Order wise but not contiguous

String1 : a b c d e f g h i j
String2 : c d g i
cdgi

***Intersection line not allowed**

String1 : a b c d e f g h i j
String2 : e c d g i

Example 1:

String1 : a b d a c e
String2 : b a b c e
bace

String1: a b d a c e

String2: b a b c e

b a c e

a b c e

A

b	d	␣
0	1	2

B

a	b	c	d	␣
0	1	2	3	4

```

graph TD
    A["A[0], B[0] = 2  
b, a"] --> B["A[1], B[0] = 1  
d, a"]
    A --> C["A[0], B[1] = 2  
b, b"]
    B --> D["A[2], B[0] = 0  
␣, a"]
    B --> E["A[1], B[1] = 1  
d, b"]
    C --> F["1 + A[1], B[2] = 1  
d, c"]
    E --> G["A[2], B[1] = 0  
␣, b"]
    E --> H["A[1], B[2] = 1  
d, c"]
    F --> I["1 + A[2], B[3] = 0  
␣, c"]
    F --> J["1 + A[1], B[3] = 1  
d, d"]
    H --> K["1 + A[2], B[2] = 0  
␣, c"]
    H --> L["1 + A[1], B[2] = 1  
d, c"]
    J --> M["1 + A[2], B[3] = 0  
␣, d"]
    J --> N["1 + A[1], B[3] = 1  
d, d"]
    L --> O["1 + A[2], B[2] = 0  
␣, c"]
    L --> P["1 + A[1], B[2] = 1  
d, c"]
    
```

```

int LCS(i, j)
{
    if (A[i] == '␣' || B[j] == '␣')
        return 0;
    else if (A[i] == B[j])
        return 1 + LCS(i+1, j+1);
    else
        return max(LCS(i+1, j), LCS(i, j+1));
}
    
```

Longest Common Subsequence (LCS)

	a	b	c	d	∅
∅	0	1	2	3	4
a	2	2			
b	1	1	1	1	
c	0	0	0		0

A:

b	d	∅
0	1	2

 m

B:

a	b	c	d	∅
0	1	2	3	4

 n

$O(m \times n)$

```

int LCS(i, j)
{
    if (A[i] == '∅' || B[j] == '∅')
        return 0;
    else if (A[i] == B[j])
        return 1 + LCS(i+1, j+1);
    else
        return max(LCS(i+1, j), LCS(i, j+1));
    }
  
```

Recursion Tree

```

graph TD
    Node0["A[0], B[0] = 2  
b, a"]
    Node1["A[1], B[0] = 1  
d, a"]
    Node2["A[0], B[1] = 2  
b, b"]
    Node3["A[2], B[0] = 0  
∅, a"]
    Node4["A[1], B[1] = 1  
d, b"]
    Node5["A[1], B[2] = 1  
d, c"]
    Node6["A[2], B[1] = 0  
∅, b"]
    Node7["A[1], B[3] = 1  
d, d"]
    Node8["A[2], B[2] = 0  
∅, c"]
    Node9["A[1], B[4] = 0  
d, ∅"]
    Node10["A[2], B[3] = 0  
∅, d"]
    Node11["A[1], B[5] = 0  
d, ∅"]
    Node12["A[2], B[4] = 0  
∅, ∅"]

    Node0 --> Node1
    Node0 --> Node2
    Node1 --> Node3
    Node1 --> Node4
    Node2 --> Node5
    Node2 --> Node6
    Node4 --> Node7
    Node4 --> Node8
    Node5 --> Node9
    Node5 --> Node10
    Node7 --> Node11
    Node7 --> Node12
  
```

m str1: stone (one)
n str2: longest 3

		l	o	n	g	e	s	t	
		0	1	2	3	4	5	6	7
s	0	0	0	0	0	0	0	0	0
t	1	0	0	0	0	0	0	1	1
o	2	0	0	0	0	0	0	1	2
n	3	0	0	1	1	1	1	1	2
e	4	0	0	1	2	2	2	2	2
	5	0	0	1	2	2	3	3	3

if ($A[i] = B[j]$)
 $LCS[i, j] = 1 + LCS[i-1, j-1]$
else
 $LCS[i, j] = \max(LCS[i-1, j], LCS[i, j-1])$

mxn
 $O(mn)$

o n e

LSC - Using Dynamic Programming Implementation:

```
1  /* Returns length of LCS for X[0..m-1], Y[0..n-1] */
2  int lcs(char[] X, char[] Y, int m, int n)
3  {
4      int L[][] = new int[m + 1][n + 1];
5
6      /* Following steps build L[m+1][n+1] in bottom up fashion. Note
7       that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1] */
8      for (int i = 0; i <= m; i++) {
9          for (int j = 0; j <= n; j++) {
10             if (i == 0 || j == 0)
11                 L[i][j] = 0;
12             else if (X[i - 1] == Y[j - 1])
13                 L[i][j] = L[i - 1][j - 1] + 1;
14             else
15                 L[i][j] = max(L[i - 1][j], L[i][j - 1]);
16         }
17     }
18     return L[m][n];
19 }
20
21 /* Utility function to get max of 2 integers */
22 int max(int a, int b)
23 {
24     return (a > b) ? a : b;
25 }
26
27 public static void main(String[] args)
28 {
29     String s1 = "AGGTAB";
30     String s2 = "GXTXAYB";
31
32     char[] X = s1.toCharArray();
33     char[] Y = s2.toCharArray();
34     int m = X.length;
35     int n = Y.length;
36
37     System.out.println("Length of LCS is"
38         + " " + lcs(X, Y, m, n));
39 }
```

Output: Length of LCS is 4