Longest Common Subsequence

February 25, 2023

Bangladesh University of Enginnering and Technology

What is LCS

Recursive Formulation

Using Dynamic Programming

What is LCS

Problem Definition

Given two strings text1 and text2, return the length of their longest common subsequence

What is a sub-sequence?

Subsequence

A sub-sequence is a sequence that can be obtained by deleting elements from the original sequence without changing order of the elements.

Subsequence

A sub-sequence is a sequence that can be obtained by deleting elements from the original sequence without changing order of the elements.

Let str = ”ABCDE” be a sequence

A B C D E F

Subsequence

A sub-sequence is a sequence that can be obtained by deleting elements from the original sequence without changing order of the elements.

Let str = ”ABCDE” be a sequence

A B C D E F

Then some subsequnce of str are : ABC

Subsequence

A sub-sequence is a sequence that can be obtained by deleting elements from the original sequence without changing order of the elements.

Let str = ”ABCDE” be a sequence

A B C D E F

Then some subsequnce of str are : ABC ACE

Subsequence

A sub-sequence is a sequence that can be obtained by deleting elements from the original sequence without changing order of the elements.

Let str = ”ABCDE” be a sequence

A B C D E F

Then some subsequnce of str are : ABC ACE BEF etc.

Longest Common Subsequence

From the two given sequences, we have to find a subsequence that is present in both of them and is the longest.

$$S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$$
$$S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$$

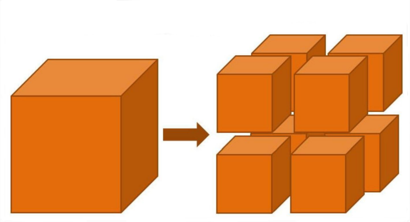
Longest Common Subsequence

$$S_1 = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA$$
$$S_2 = GTCGTTCGGAATGCCGTTGCTCTGTAAA$$

$$S_3 = GTCGTCGGAAGCCGGCCGAA$$

How to approach?

Break it down into smaller subproblems.



Breaking into smaller sub-problems

Suppose we have two sequences

$$X = \text{A B C A D E}$$

$$Y = \text{A C B E}$$

Breaking into smaller sub-problems

Suppose we have two sequences

$$X \quad = \quad A\ B\ C\ A\ D\ \boxed{E}$$

$$Y \quad = \quad A\ C\ B\ \boxed{E}$$

Breaking into smaller sub-problems

Suppose we have two sequences

$$X = A\ B\ C\ A\ D\ \boxed{E}$$

$$Y = A\ C\ B\ \boxed{E}$$

Longest Common Subsequence : ____________E

Breaking into smaller sub-problems

Suppose we have two sequences

$$X = \boxed{A\ B\ C\ A\ D}\ E$$

$$Y = \boxed{A\ C\ B}\ E$$

Longest Common Subsequence : ___________E

Breaking into smaller sub-problems

Suppose we have two sequences

X = A B C A D E

Y = A C B E

Longest Common Subsequence : __________E

Breaking into smaller sub-problems

Suppose we have two sequences

$$X \quad = \quad A\ B\ C\ A\ D\ \boxed{F}$$

$$Y \quad = \quad A\ C\ B\ E$$

Longest Common Subsequence : ___________

Breaking into smaller sub-problems

Suppose we have two sequences

$$X = A\ B\ C\ A\ D\ \boxed{F}$$

$$Y = A\ C\ B\ \boxed{E}$$

Longest Common Subsequence : ___________

Breaking into smaller sub-problems

Suppose we have two sequences

$$X \quad = \quad A \; B \; C \; A \; D \; \boxed{F}$$

$$Y \quad = \quad A \; C \; B \; \boxed{E}$$

Longest Common Subsequence : ___________

Breaking into smaller sub-problems

Suppose we have two sequences

$$X = \boxed{A\ B\ C\ A\ D\ F}$$

$$Y = \boxed{A\ C\ B}\ E$$

Longest Common Subsequence : ___________

Breaking into smaller sub-problems

Suppose we have two sequences

$$X = \boxed{A\ B\ C\ A\ D}\ F$$

$$Y = \boxed{A\ C\ B\ E}$$

Longest Common Subsequence : ___________

Recursive Formulation

Recursion

If we have two sequences

$$X = [\ 1 \cdot \cdot \cdot \cdot \cdot \cdot m\ ] \qquad Y = [\ 1 \cdot \cdot \cdot \cdot \cdot \cdot n\ ]$$

Recursion

If we have two sequences

$$X = [\underline{1 \cdot \cdot \cdot a} \cdot \cdot m] \qquad Y = [\underline{1 \cdot \cdot b} \cdot \cdot \cdot n]$$

Let $LCS(a, b)$ denote the longest common subsequence of $x_1 x_2 \ldots x_a$ and $y_1 y_2 \ldots y_a$.

Recursion

If we have two sequences

$$X = [\underline{1 \cdot \cdot \cdot \cdot \cdot \cdot m}] \qquad Y = [\underline{1 \cdot \cdot \cdot \cdot \cdot \cdot n}]$$

Let $LCS(a, b)$ denote the longest common subsequence of $x_1 x_2 \ldots x_a$ and $y_1 y_2 \ldots y_a$.

Our goal is to find $LCS(m, n)$

Recursive Formulation

$X = [\ 1\ 2\ 3\ \cdots\ m\ ]$ $Y = [\ 1\ 2\ 3 \cdots\ n\ ]$

Recursive Formulation

$X = [\ 1\ 2\ 3\ \cdots \boxed{m}\ ]$ $\quad$ $Y = [\ 1\ 2\ 3 \cdots \boxed{n}\ ]$

When X[m] == Y[n]

Recursive Formulation

$X = [\ \boxed{1\ 2\ 3\ \cdots}\ m\ ]$ $Y = [\ \boxed{1\ 2\ 3\cdots}\ n\ ]$

When X[m] == Y[n]

$$LCS(m, n) = LCS(m-1, n-1) + 1$$

Recursive Formulation

$$X = [\ 1\ \ 2\ \ 3\ \cdots \boxed{m}\ ] \qquad Y = [\ 1\ \ 2\ \ 3 \cdots \boxed{n}\ ]$$

When X[m] != Y[n]

Recursive Formulation

$X = [\ \boxed{1\ \ 2\ \ 3\ \ \cdots\ \ m}\ ]$ $\quad Y = [\ \boxed{1\ \ 2\ \ 3 \cdots}\ n\ ]$

When X[m] != Y[n]

$$LCS(m, n) = max\{LCS(m, n-1),$$

Recursive Formulation

$$X = [\ \boxed{1\ \ 2\ \ 3\ \ \cdots}\ m\ ] \qquad Y = [\ \boxed{1\ \ 2\ \ 3 \cdots\ \ n}\ ]$$

When X[m] != Y[n]

$$LCS(m, n) = max\{LCS(m, n-1), LCS(m-1, n)\}$$

Recursive Formulation

X = [1 2 3 · · · m] Y = [1 2 3· · · n]

The recursion solution:

$$LCS(m,n) = \begin{cases} LCS(m-1, n-1) + 1 & X[m] == Y[n] \\ max\{LCS(m-1, n), LCS(m, n-1)\} & X[m]! = Y[n] \end{cases}$$

What about the base case?

Recursive Formulation

If m == 0 then LCS(m,n) = 0

If n == 0 then LCS(m,n) = 0

Recursive Formulation

The final recursion solution:

$$LCS(m,n) = \begin{cases} 0 & m == 0 || n == 0 \\ LCS(m-1, n-1) + 1 & X[m] == Y[n] \\ max\{LCS(m-1, n), LCS(m, n-1)\} & X[m] != Y[n] \end{cases}$$

Recursion

$$LCS(m,n) = \begin{cases} 0 & \text{if } \; m = 0 \quad or \quad n = 0 \\ LCS(m-1, n-1) + 1 & \text{if } \; X[m] = Y[n] \\ MAX(LCS(m-1,n), LCS(m, n-1)) & \text{if } \; X[m]! = Y[n] \end{cases}$$

```
def LCS(X[1..m], Y[1..n]) :
    if m = 0 or n = 0
        return 0
```

Recursion

$$LCS(m,n) = \begin{cases} 0 & \text{if } \ m = 0 \quad or \quad n = 0 \\ LCS(m-1, n-1) + 1 & \text{if } \ X[m] = Y[n] \\ MAX(LCS(m-1, n), LCS(m, n-1)) & \text{if } \ X[m]! = Y[n] \end{cases}$$

```
def LCS(X[1..m], Y[1..n]) :
    if m = 0 or n = 0
        return 0

    if X[m] = Y[n] :
        result = LCS(m-1, n-1) + 1
```

Recursion

$$LCS(m,n) = \begin{cases} 0 & \text{if } \; m = 0 \quad or \quad n = 0 \\ LCS(m-1, n-1) + 1 & \text{if } \; X[m] = Y[n] \\ MAX(LCS(m-1, n), LCS(m, n-1)) & \text{if } \; X[m]! = Y[n] \end{cases}$$

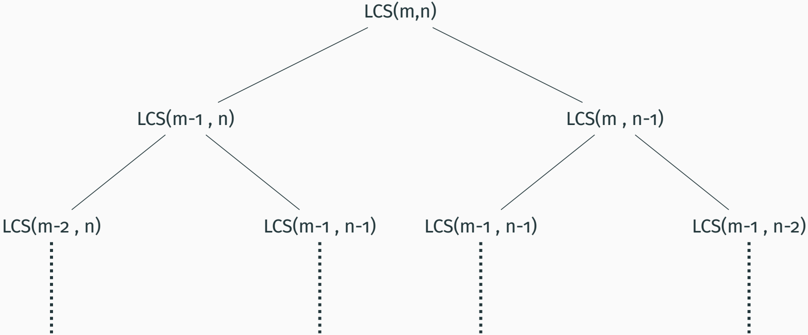
```
def LCS(X[1..m], Y[1..n]) :
    if m = 0 or n = 0
        return 0

    if X[m] = Y[n] :
        result = LCS(m-1, n-1) + 1

    else :
        result = max(LCS(m-1, n), LCS(m, n-1))
```

Recursion

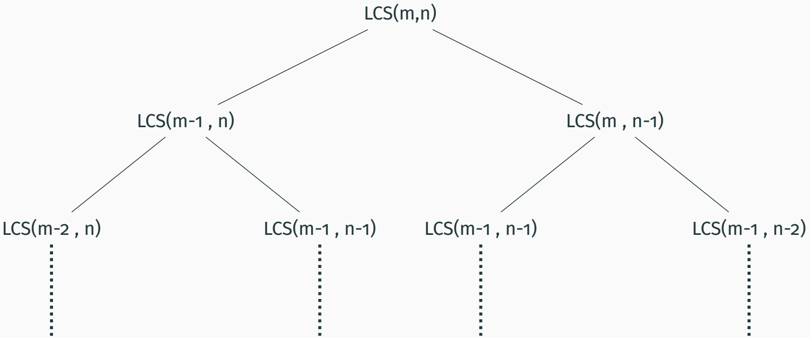
$$LCS(m,n) = \begin{cases} 0 & \text{if } m = 0 \quad or \quad n = 0 \\ LCS(m-1, n-1) + 1 & \text{if } X[m] = Y[n] \\ MAX(LCS(m-1, n), LCS(m, n-1)) & \text{if } X[m]! = Y[n] \end{cases}$$

```
def LCS(X[1..m], Y[1..n]) :
    if m = 0 or n = 0
        return 0

    if X[m] = Y[n] :
        result = LCS(m-1, n-1) + 1

    else :
        result = max(LCS(m-1, n), LCS(m, n-1))

    return result
```

Frame Title



Frame Title



So the time complexity is $O(2^n)$

Memoized Algorithm

$$LCS(m,n) = \begin{cases} 0 & \text{if } m = 0 \quad or \quad n = 0 \\ LCS(m-1, n-1) + 1 & \text{if } X[m] = Y[n] \\ MAX(LCS(m-1,n), LCS(m,n-1)) & \text{if } X[m]! = Y[n] \end{cases}$$

```
def LCS(X[1..m], Y[1..n]) :
    if m = 0 or n = 0
        return 0
    if LCS_memo[m][n] != -INF:
        return LCS_memo[m][n]
    else if X[m] = Y[n] :
        result = LCS(m-1, n-1) + 1
    else :
        result = max(LCS(m-1, n), LCS(m, n-1))
    LCS_memo[m][n] = result
    return result
```

Memoized Algorithm

$$LCS(m,n) = \begin{cases} 0 & \text{if } \ m = 0 \quad or \quad n = 0 \\ LCS(m-1, n-1) + 1 & \text{if } \ X[m] = Y[n] \\ MAX(LCS(m-1,n), LCS(m, n-1)) & \text{if } \ X[m]! = Y[n] \end{cases}$$

```
def LCS(X[1..m], Y[1..n]) :
    if m = 0 or n = 0
        return 0
    if LCS_memo[m][n] != -INF:
        return LCS_memo[m][n]
    else if X[m] = Y[n] :
        result = LCS(m-1, n-1) + 1
    else :
        result = max(LCS(m-1, n), LCS(m, n-1))
    LCS_memo[m][n] = result
    return result
```

Time complexity is $O(mn)$

Using Dynamic Programming

Dynamic Programming

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i							
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		y_j	*B*	*D*	*C*	*A*	*B*	*A*
0	x_i	0	0	0	0	0	0	0
1	*A*	0	0					
2	*B*	0						
3	*C*	0						
4	*B*	0						
5	*D*	0						
6	*A*	0						
7	*B*	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0				
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0				
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	0	0	0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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		
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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	
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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	
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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	
3	*C*	0						
4	*B*	0						
5	*D*	0						
6	*A*	0						
7	*B*	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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				
4	B	0						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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						
5	D	0						
6	A	0						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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						
6	A	0						
7	B	0						

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else

    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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						
7	B	0						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
	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						

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Dynamic Programming

		0	1	2	3	4	5	6
		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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else

LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Simulated Annealing (SA)

		0	1	2	3	4	5	6
	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

```
def dp_lcs(x[1..m], y[1..n]):

    for i = 0 to m :
        L[i, 0] = 0

    for j = 0 to n :
        L[0, j] = 0

    for i = 1 to m :
        for j = 1 to n :
            if x[i] == y[j] :
                L[i, j] = L[i-1, j-1] + 1
            else if L[i-1, j] >= L[i, j-1] :
                L[i, j] = L[i-1, j]
            else :
                L[i, j] = L[i, j-1]

    return L[m, n]
```

What if we want to find the subsequence too?

Going back

		0	1	2	3	4	5	6
	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

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else
    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
	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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else
LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
		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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else
LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
		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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else
LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
		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

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else
    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
		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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else
LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
		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

If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else
    LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
	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

```
If m=0 or n=0
    LCS(m,n) = 0

Else if X[m] = Y[n]
    LCS(m,n) = LCS(m-1, n-1) + 1

Else
    LCS(m,n) = max( LCS(m-1,n), LCS(m, n-1))
```

Going back

		0	1	2	3	4	5	6
	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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else
LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Going back

		0	1	2	3	4	5	6
	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

If m=0 or n=0
LCS(m,n) = 0

Else if X[m] = Y[n]
LCS(m,n) = LCS(m-1, n-1) + 1

Else
LCS(m,n) = max(LCS(m-1,n), LCS(m, n-1))

Find LCS String

Function `LCS-Genarate-String`($X[1 \ldots m], Y[1 \ldots n]$):

Initialization

for $i \leftarrow 0$ *to* m **do**
    $L[i][0] \leftarrow 0$ `/* Initialization */`
end

for $j \leftarrow 0$ *to* n **do**
    $L[0][j] \leftarrow 0$ `/* Initialization */`
end

Find LCS String

Function `LCS-Genarate-String`(*X[1...m], Y[1...n]*):

Matching Characters

for $i \leftarrow 1$ *to* m **do**
  for $j \leftarrow 1$ *to* n **do**
    `/* If current characters match, add 1 to LCS */`

    if $X[i] = Y[j]$ **then**
      $L[i][j] \leftarrow L[i-1][j-1] + 1$

      $D[i][j] = 1$;

    end

    else
      `/* If they don't match, take the maximum of the LCS */`

      if $L[i\text{-}1][j] \geq L[i][j\text{-}1]$ **then**
        $L[i][j] \leftarrow L[i-1][j]$

        $D[i][j] = 2$;

      end

      else
        $L[i][j] \leftarrow L[i][j-1]$

        $D[i][j] = 3$;

      end

    end

  end

THANK YOU