Longest Common Subsequence

Longest Common Subsequence (LCS)

A DNA sequence is composed of 4 letters A, C, G, T

Example: X= {A G G G C T}

A **subsequence** of a sequence is the same sequence with 0 or more elements left out (deleted)

Subsequences of X = A C , G G G , G C T, G T, …..

Substring is different from subsequence, substring is consecutive string.

For example:

G T is subsequence of X but it is not substring of X.

Longest Common Subsequence (LCS)

A DNA sequence is composed of 4 letters A, C, G, T

Example: X= {A G G G C T}

A **subsequence** is of a sequence is the same sequence with 0 or more elements left out (deleted)

Subsequences of X = A C , G G G , G C T, G T, …..

Question: Which of the following are subsequences of above sequence X?

a) A G
b) G A
c) G C T
d) A T
e) T A

Longest Common Subsequence (LCS)

A DNA sequence is composed of 4 letters A, C, G, T

Example: X= {A G G G C T}

A **subsequence** is of a sequence is the same sequence with 0 or more elements left out (deleted)

Subsequences of X = A C , G G G , G C T, G T, …..

Question: Which of the following are subsequences of above sequence X?

a) A G

b) G A

c) G C T

d) A T

e) T A

Correct Answer: a) , c), d)

Longest Common Subsequence (LCS)

Common Subsequence: A common subsequence of 2 DNA sequences is a subsequence present in both sequences

X = A G C G T A G

Y = G T C A G A

Common subsequences of X and Y = GT, GTA, G A, A G, G C A, …….

Longest Common subsequence is the longest sequence among common subsequences.

X = A **G** **C** **G** T **A** G

Y = **G** T **C** A **G** **A**

Longest is GCGA

Application

Comparison of two DNA strings in evolutionary tree

Brute Force Algorithm

- Brute force algorithm would compute all subsequences of both sequences and find the common and print the longest.

OR

- Compute all subsequences of one sequence and check if it is also present in the other sequence. Print the longest common sequence.

- How many subsequences are there in a sequence of n elements?
- Think about the definition of a subsequence
- A subsequence is same sequence with 0 or more elements left out.
- For each of the n elements, we have an option, delete it or keep it.
- 2 possibilities for each of the n elements so total subsequences =
- $2 * 2 * 2 \ldots.. * 2 = 2^n$

Brute Force Algorithm

- if $|X| = m$, $|Y| = n$, then there are 2^m subsequences of x; we must compare each with Y (n comparisons)
- So the running time of the brute-force algorithm is $O(n\ 2^m)$

The brute force algorithm will take exponential time since computing all subsequences of any one sequence will take exponential time.

Counter Example

- Suppose we have following two DNA sequences:

- X = A C G T A
- Y = A T G T T C
- LCS = A G T
- If you run the wrong O(m*n) algorithm on it, it will fail both ways.
- You can try following as well
- X = A T G T T C
- Y = A C G T A
- LCS= A G T

Counter Example

```
1.  The wrong O (m*n) algorithm is following:
2.  i = 1
3.  while (i < m) {
4.    j = 1
5.    while (j < n) {
6.        if (X[i] == Y [j])
7.            Print X[i], i++, j++
8.        else
9.            j++
10.   }
11.   i ++
12. }
```

This algorithm tries to print all common sub sequences but it will fail to print LCS which is A G T

Optimal Substructure in LCS

- LCS problem has *optimal substructure*: solutions of subproblems are parts of the final solution.
- Subproblems: "find LCS of pairs of *prefixes* of X and Y"
- If $X = <x_1, ..., x_m>$ and if $Y = <y_1, ..., y_n>$ are sequences, let $Z = <z_1, ..., z_k>$ be some LCS of x and y.

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

Optimal Substructure in LCS

- If X = <x_1, …,x_m> and if Y = <y_1, …,y_n> are sequences, let Z = <z_1, …,z_k> be some LCS of x and y.

1. If $x_m = y_n$ then $z_k = x_m$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}

$$x_1, x_2, x_3, \ldots\ldots\ldots, x_{m-2}, x_{m-1}, x_m$$

$$y_1, y_2, y_3, \ldots\ldots\ldots, y_{n-2}, y_{n-1}, y_n$$

X = G C G T A G

Y = G T T C A G A G

Z = G C G A G

Optimal Substructure in LCS

- If $X = <x_1, ...,x_m>$ and if $Y = <y_1, ...,y_n>$ are sequences, let $Z = <z_1, ...,z_k>$ be some LCS of x and y.

1. If $x_m = y_n$ then $z_k = x_m$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}

$x_1, x_2, x_3, \ldots\ldots\ldots, x_{m-2}, x_{m-1}, x_m$

$y_1, y_2, y_3, \ldots\ldots\ldots, y_{n-2}, y_{n-1}, y_n$

Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}

$x_1, x_2, x_3, \ldots\ldots\ldots, x_{m-2}, x_{m-1}, x_m$

$y_1, y_2, y_3, \ldots\ldots\ldots, y_{n-2}, y_{n-1}, y_n$

Optimal Substructure in LCS

- If $X = \langle x_1, ..., x_m \rangle$ and if $Y = \langle y_1, ..., y_n \rangle$ are sequences, let $Z = \langle z_1, ..., z_k \rangle$ be some LCS of x and y.

1. If $x_m = y_n$ then $z_k = x_m$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}

Proof by Contradiction:

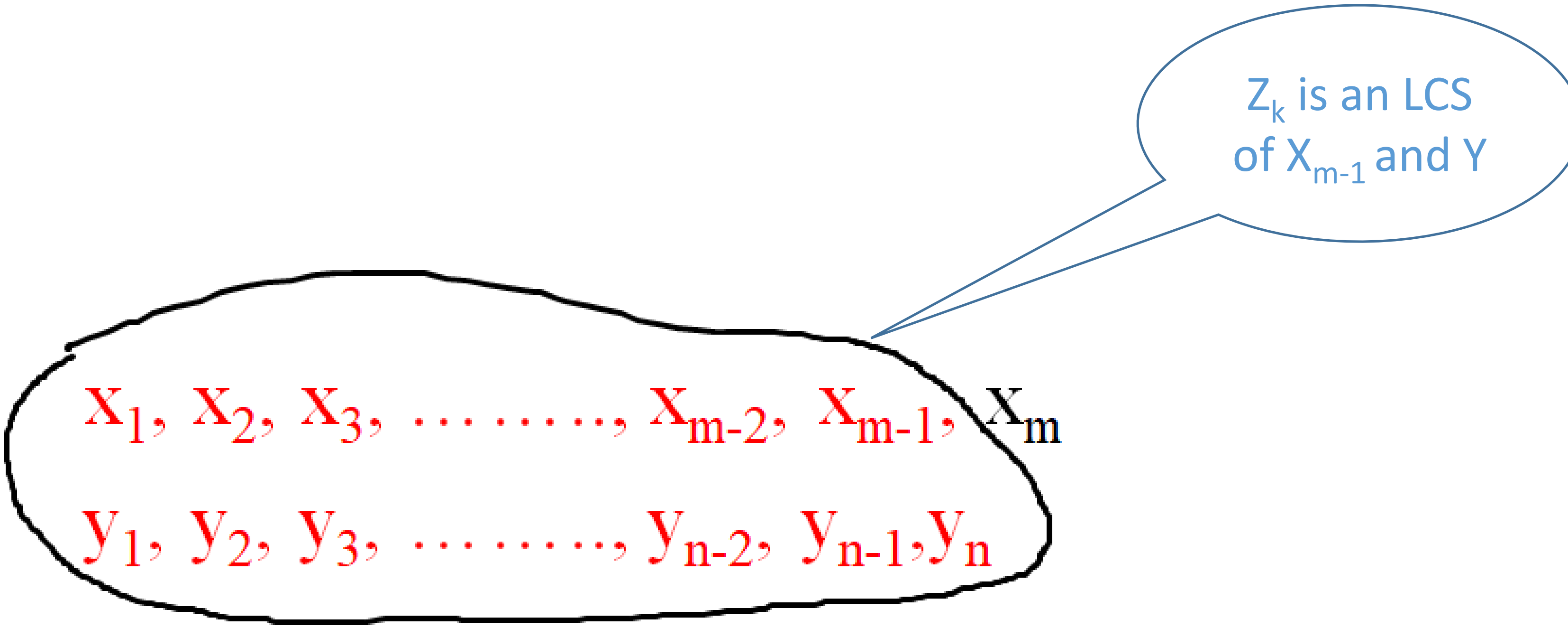
1.If $z_k \neq x_m$ then we could add $x_m = y_n$ to Z to get an LCS of length k + 1. By contradiction it must be that $z_k = x_m = y_n$.

$|z_{k-1}| = k - 1$ and it is an LCS of X_{m-1} and Y_{n-1} .
It is an LCS, if not then suppose W is LCS of X_{m-1} and Y_{n-1} with $| W | > k - 1$ and so by appending $x_m = y_n$ to W we get a LCS of X and Y of length greater than k, a contradiction.

Optimal Substructure in LCS

- If $X = \langle x_1, \ldots, x_m \rangle$ and if $Y = \langle y_1, \ldots, y_n \rangle$ are sequences, let $Z = \langle z_1, \ldots, z_k \rangle$ be some LCS of x and y.

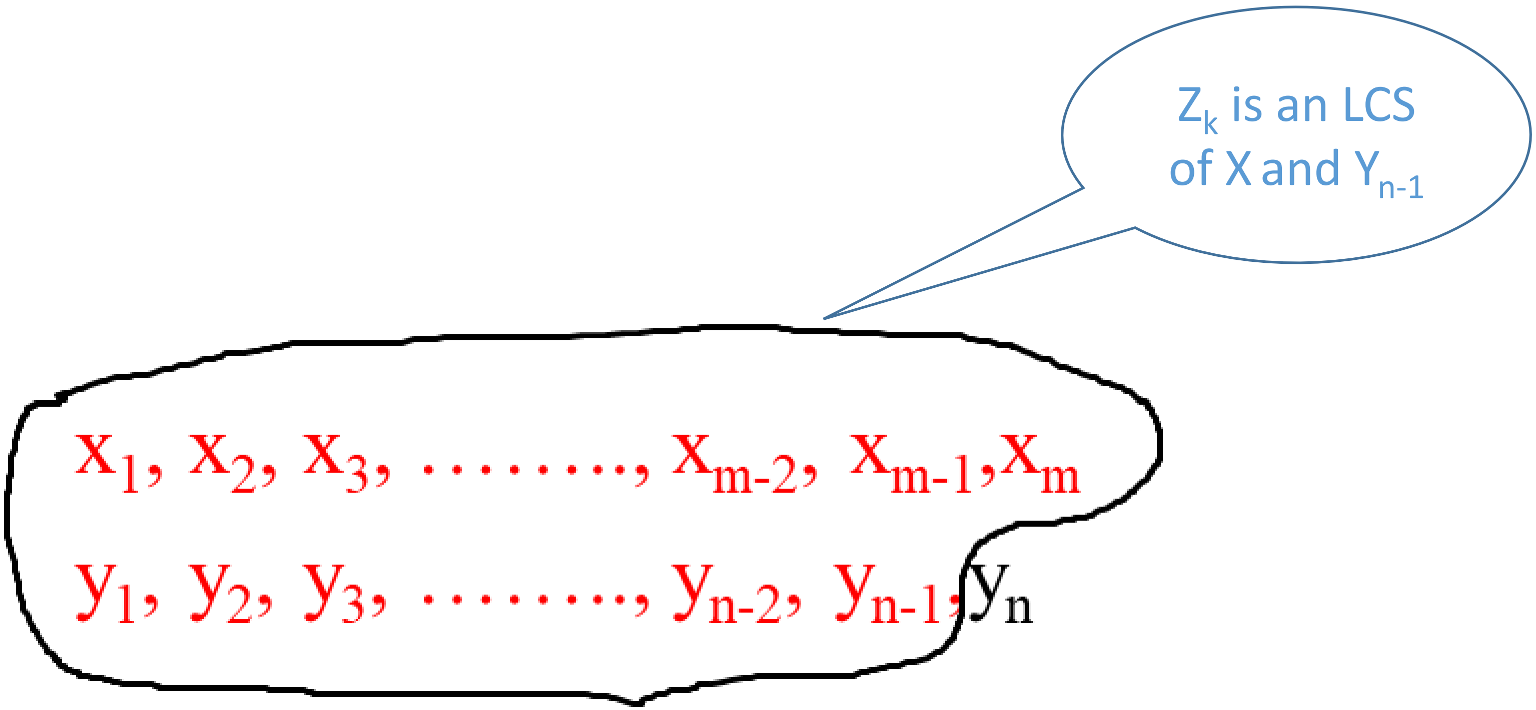
2. If $x_m \neq y_n$ then $z_k \neq x_m$, Z is an LCS of X_{m-1} and Y



Optimal Substructure in LCS

- If $X = <x_1, ...,x_m>$ and if $Y = <y_1, ...,y_n>$ are sequences, let $Z = <z_1, ...,z_k>$ be some LCS of x and y.

3. If $x_m \neq y_n$ then $z_k \neq y_n$ then Z is an LCS of X and Y_{n-1}



Optimal Substructure in LCS

- If $X = \langle x_1, \ldots, x_m \rangle$ and if $Y = \langle y_1, \ldots, y_n \rangle$ are sequences, let $Z = \langle z_1, \ldots, z_k \rangle$ be some LCS of x and y.

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

Z_k is an LCS of X_{m-1} and Y

$$x_1, x_2, x_3, \ldots\ldots, x_{m-2}, x_{m-1}, X_m$$

$$y_1, y_2, y_3, \ldots\ldots, y_{n-2}, y_{n-1}, y_n$$

Proof:

2.If $Z_k \neq X_m$ then Z is a LCS of X_{m-1} and Y.
If Z is not LCS then suppose W is LCS with of X_{m-1} and Y and | W | > k, then W would also be LCS of X and Y, a contradiction

3.Proof by reversing x and y

Optimal Substructure in LCS

- LCS problem has *optimal substructure*: solutions of subproblems are parts of the final solution.
- Subproblems: "find LCS of pairs of *prefixes* of X and Y"
- If X = $<x_1, ..., x_m>$ and if Y = $<y_1, ..., y_n>$ are sequences, let Z = $<z_1, ..., z_k>$ be some LCS of x and y.

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

Recursive Function $O(2^n)$

```
/* Returns length of LCS for X[0..m-1], Y[0..n-1] */
int lcs( char *X, char *Y, int m, int n )
{
  if (m == 0 || n == 0)
    return 0;
  if (X[m-1] == Y[n-1])
    return 1 + lcs(X, Y, m-1, n-1);
  else
    return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}
```

LCS Algorithm

- First we'll find the length of LCS (value of optimal solution). Later we'll modify the algorithm to find LCS (optimal solution) itself.
- Define X_i, Y_j to be the prefixes of X and Y of length *i* and *j* respectively
- Define $c[i,j]$ to be the length of LCS of X_i and Y_j
- Then the length of LCS of X and Y will be $c[m,n]$

LCS Algorithm

- First we'll find the length of LCS (value of optimal solution). Later we'll modify the algorithm to find LCS (optimal solution) itself.
- Define X_i, Y_j to be the prefixes of X and Y of length *i* and *j* respectively
- Define $c[i,j]$ to be the length of LCS of X_i and Y_j
- Then the length of LCS of X and Y will be $c[m,n]$

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

LCS recursive solution

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

- We start with $i = j = 0$ (empty substrings of x and y)
- Since X_0 and Y_0 are empty strings, their LCS is always empty (i.e. $c[0,0] = 0$)
- LCS of empty string and any other string is empty, so for every i and j: $c[0, j] = c[i,0] = 0$

LCS recursive solution

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i]=y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

- When we calculate *c[i,j]*, we consider two cases:
- **First case:** *x[i]=y[j]*: one more symbol in strings X and Y matches, so the length of LCS X_i and Y_j equals to the length of LCS of smaller strings X_{i-1} and Y_{i-1} , plus 1

LCS recursive solution

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

- **Second case:** $x[i] \neq y[j]$
- As symbols don't match, our solution is not improved, and the length of LCS(X_i , Y_j) is the same as before (i.e. maximum of LCS(X_i, Y_{j-1}) and LCS(X_{i-1},Y_j)

LCS Length Algorithm

```
LCS-Length(X, Y)
m = length(X)  // get the # of symbols in X
n  = length(Y) // get the # of symbols in Y
for i = 1 to m      c[i,0] = 0      // special case: Y_0
for j = 1 to n      c[0,j] = 0      // special case: X_0
for i = 1 to m                      // for all X_i
      for j = 1 to n                      // for all Y_j
            if ( X_i == Y_j )
                  c[i,j] = c[i-1,j-1] + 1
            else c[i,j] = max( c[i-1,j], c[i,j-1] )
return c
```

LCS Example

We’ll see how LCS algorithm works on the following example:

- X = ABCB
- Y = BDCAB

What is the Longest Common Subsequence of X and Y?

LCS(X, Y) = BCB

X = A **B** **C** **B**

Y = **B** D **C** A **B**

LCS Example (0)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi						
1	**A**						
2	**B**						
3	**C**						
4	**B**						

X = ABCB; m = |X| = 4
Y = BDCAB; n = |Y| = 5
Allocate array c[5,4]

LCS Example (1)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

for i = 1 to m   c[i,0] = 0
for j = 1 to n   c[0,j] = 0

LCS Example (2)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**				
2	**B**		**0**					
3	**C**		**0**					
4	**B**		**0**					

if (X_i == Y_j)
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (3)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**		
2	**B**		**0**					
3	**C**		**0**					
4	**B**		**0**					

if ($X_i == Y_j$)
$c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (4)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	
2	**B**		**0**					
3	**C**		**0**					
4	**B**		**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (5)

ABCB

BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (6)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**				
3	**C**	**0**					
4	**B**	**0**					

if ($X_i == Y_j$)
$c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (7)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	
3	**C**		**0**					
4	**B**		**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (8)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	**2**
3	**C**		**0**					
4	**B**		**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (10)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	**2**
3	**C**		**0**	**1**	**1**			
4	**B**		**0**					

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (11)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	**2**
3	**C**		**0**	**1**	**1**	**2**		
4	**B**		**0**					

if ($X_i == Y_j$)
$c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (12)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**					

if (X_i == Y_j)
    c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (13)

ABCB

BDCAB

		j	0	1	2	3	4	5
i			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	**2**
3	**C**		**0**	**1**	**1**	**2**	**2**	**2**
4	**B**		**0**	**1**				

if ($X_i == Y_j$)

$c[i,j] = c[i-1,j-1] + 1$

else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Example (14)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Yj	**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1**	**1**	**1**	**1**	**2**
3	**C**		**0**	**1**	**1**	**2**	**2**	**2**
4	**B**		**0**	**1**	**1**	**2**	**2**	

if (X_i == Y_j)
c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

LCS Example (15)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	3

if ($X_i == Y_j$)
$c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

LCS Algorithm Running Time

- LCS algorithm calculates the values of each entry of the array c[m,n]
- So what is the running time?

O(m*n)

since each c[i,j] is calculated in constant time, and there are m*n elements in the array

How to find actual LCS (Optimal Solution)

- So far, we have just found the *length* of LCS, but not LCS itself.
- We want to modify this algorithm to make it output Longest Common Subsequence of X and Y

Each *c[i,j]* depends on *c[i-1,j]* and *c[i,j-1]*

or *c[i-1, j-1]*

For each c[i,j] we can say how it was acquired:

2	2
2	3

For example, here
c[i,j] = c[i-1,j-1] +1 = 2+1=3

How to find actual LCS - continued

- Remember that

$$c[i,j] = \begin{cases} c[i-1, j-1]+1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

- So we can start from *c[m,n]* and go backwards
- Whenever *c[i,j] = c[i-1, j-1]+1*, remember *x[i]* (because *x[i]* is a part of LCS)
- When i=0 or j=0 (i.e. we reached the beginning), output remembered letters in reverse order

Finding LCS

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**0**	**1**	**1**	**2**	**2**	**3**

Finding LCS (2)

i		j	0	1	2	3	4	5
		Yj		**B**	**D**	**C**	**A**	**B**
0	Xi		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1**	**1**
2	**B**		**0**	**1** ←	**1**	**1**	**1**	**2**
3	**C**		**0**	**1**	**1**	**2** ←	**2**	**2**
4	**B**		**0**	**1**	**1**	**2**	**2**	**3**

LCS (reversed order): **B C B**

LCS (straight order): **B C B**

(this string turned out to be a palindrome)

Print LCS using same array

		0	1	2	3	4	5	6	7
		Ø	M	Z	J	A	W	X	U
0	Ø	**0**	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	1	1
2	M	0	**1**	1	1	1	1	1	1
3	J	0	1	1	**2**	2	2	2	2
4	Y	0	1	1	2	2	2	2	2
5	A	0	1	1	2	**3**	3	3	3
6	U	0	1	1	2	3	3	3	**4**
7	Z	0	1	2	2	3	3	3	4

```
// Start from the right-most-bottom-most corner and
  // one by one store characters in lcs[]
  int i = m, j = n;
  while (i > 0 && j > 0)
  {
    // If current character in X[] and Y are same, then
    // current character is part of LCS
    if (X[i-1] == Y[j-1])
    {
        lcs[index-1] = X[i-1]; // Put current character in result
        i- -;  j- -;  index- -;     // reduce values of i, j and index
    }

    // If not same, then find the larger of two and
    // go in the direction of larger value
    else if (L[i-1][j] > L[i][j-1])
       i- -;
    else
       j- -;
  }

  // Print the lcs
  cout << "LCS of " << X << " and " << Y << " is " << lcs;
}
```

Time Complexity = O(m + n)

LCS: Algo

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
     do c[i, 0] ← 0
for j ← 0 to n
     do c[0, j] ← 0
for i ← 1 to m
     do for j ← 1 to n
            do if x_i = y_j
                  then c[i, j] ← c[i − 1, j − 1] + 1
                       b[i, j] ← "↖"
                  else if c[i − 1, j] ≥ c[i, j − 1]
                          then 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
```

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

Time complexity = O (m + n)

We cannot use only last row to print LCS, it will not work on following example

		0	1	2	3	4	5	6	7
		Ø	M	Z	J	A	W	X	U
0	Ø	**0**	0	0	0	0	0	0	0
1	X	0	0	0	0	0	0	1	1
2	M	0	**1**	1	1	1	1	1	1
3	J	0	1	1	**2**	2	2	2	2
4	Y	0	1	1	2	2	2	2	2
5	A	0	1	1	2	**3**	3	3	3
6	U	0	1	1	2	3	3	3	**4**
7	Z	0	1	2	2	3	3	3	4

Practice Problem for Dry Run

i		j: 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