Welcome To My Presentation

My Presentation Topic is

Longest Common Subsequence

Presented By

Tushar Sarkar

Student ID : 18CSE35

Second Year Second Semester

Department of CSE, BSMRSTU.

Common Subsequence

- A subsequence of a string is the string with zero or more chars left out.
- A common subsequence of two strings:
  - A subsequence of both strings
  - Example :

x = {A B C B D A B }

y = {B D C A B A}

Here,

{B C} and {A A} are both common subsequences of x and y.

Longest Common Subsequence

- Given two sequences *x*[1 . . *m*] and *y*[1 . . *n*], find a longest subsequence common to them both.

x: A B C B D A B

y: B D C A B A

LCS(x,y)=BCBA

functional notation,
but not a function

Brute-force LCS algorithm

- Check every subsequence of $x[1 \,.\,.\, m]$ to see if it is also a subsequence of $y[1 \,.\,.\, n]$.

Analysis

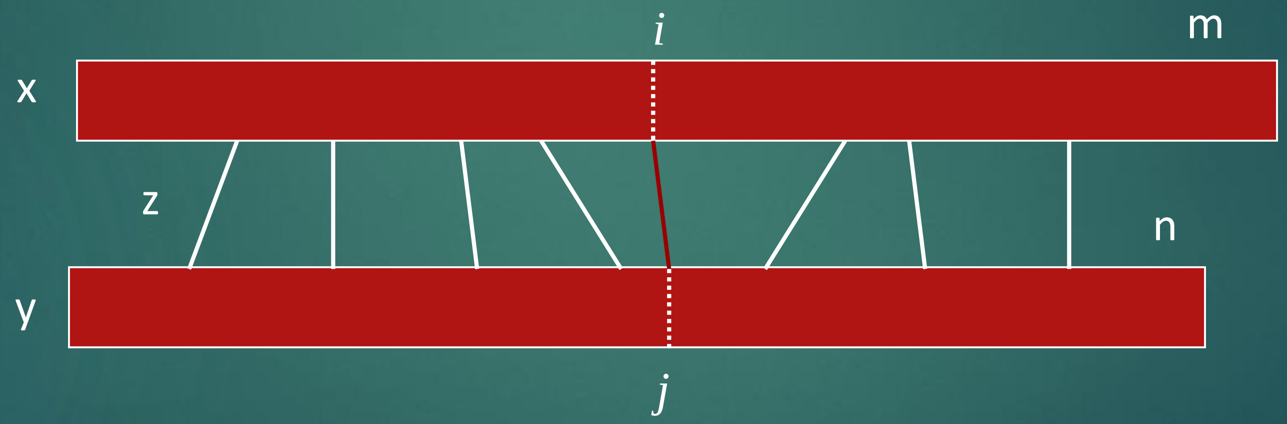
- 2^m subsequences of x (each bit-vector of length m determines a distinct subsequence of x).
- Hence, the runtime would be exponential !

Towards a better algorithm: a DP strategy

- Key: optimal substructure and overlapping sub-problems
- First we'll find the length of LCS. Later we'll modify the algorithm to find LCS itself.

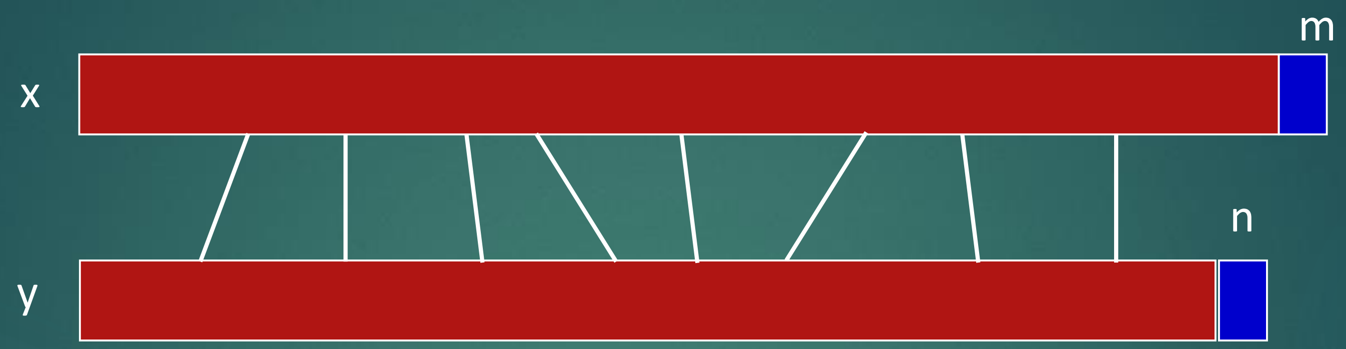
Optimal Substructure

- Notice that the LCS problem has *optimal substructure*: parts of the final solution are solutions of subproblems.
  - If z = LCS(x, y), then any prefix of z is an LCS of a prefix of x and a prefix of y.



- Subproblems: "find LCS of pairs of *prefixes* of x and y"

Recursive Thinking



- Case 1: $x[m]=y[n]$. There is **an** optimal LCS that matches $x[m]$ with $y[n]$.

  $\longrightarrow$ Find out LCS $(x[1..m-1], y[1..n-1])$

- Case 2: $x[m] \neq y[n]$. At most one of them is in LCS.
  - Case 2.1: *x[m] not in LCS* $\longrightarrow$ Find out LCS $(x[1..m-1], y[1..n])$
  - Case 2.2: *y[n] not in LCS* $\longrightarrow$ Find out LCS $(x[1..m], y[1..n-1])$

Recursive Thinking

- Case 1: $x[m]=y[n]$
  - $LCS(x, y) = LCS(x[1..m-1], y[1..n-1]) \,||\, x[m]$

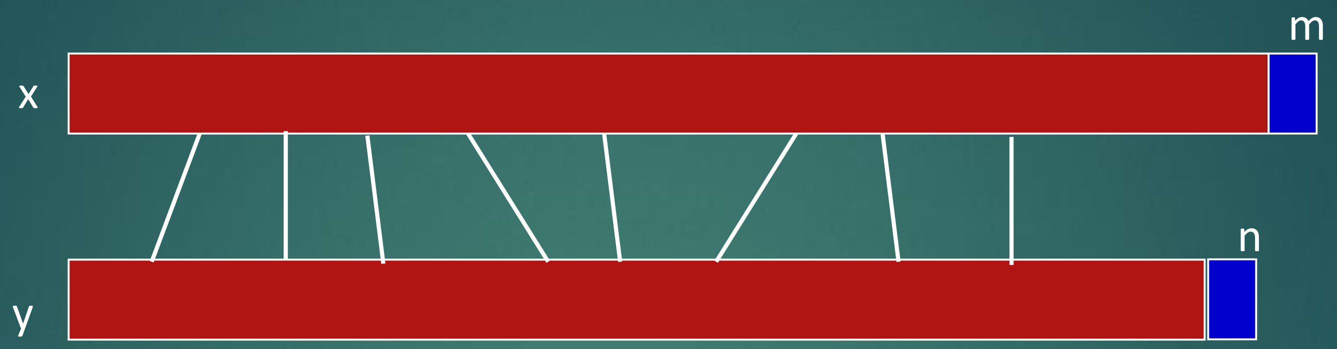
Reduce both sequences by 1 char

concatenate

- Case 2: $x[m] \neq y[n]$
  - $LCS(x, y) = LCS(x[1..m-1], y[1..n])$ or
  - $LCS(x[1..m], y[1..n-1])$, whichever is longer

Reduce either sequence by 1 char

Finding Length of LCS



- Let $c[i, j]$ be the length of LCS($x[1..i], y[1..j]$)

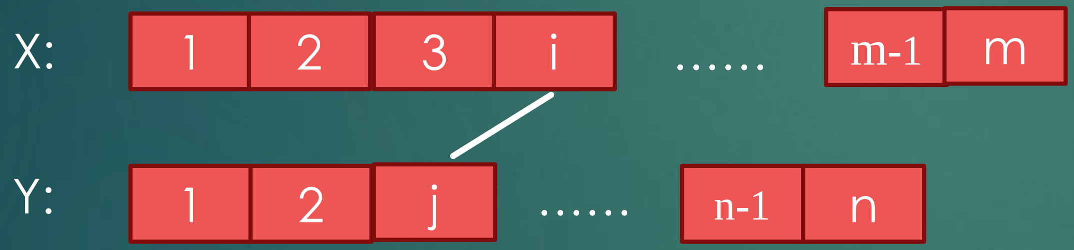
  => $c[m, n]$ is the length of LCS(x, y)
- If $x[m] = y[n]$

  $c[m, n] = c[m\text{-}1, n\text{-}1] + 1$
- If $x[m] \mathrel{!}= y[n]$

  $c[m, n] = max \{ c[m\text{-}1, n], c[m, n\text{-}1] \}$

Generalize: Recursive Formulation

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



Recursive algorithm for LCS

LCS (x, y, i, j)

if $x[i] = y[j]$

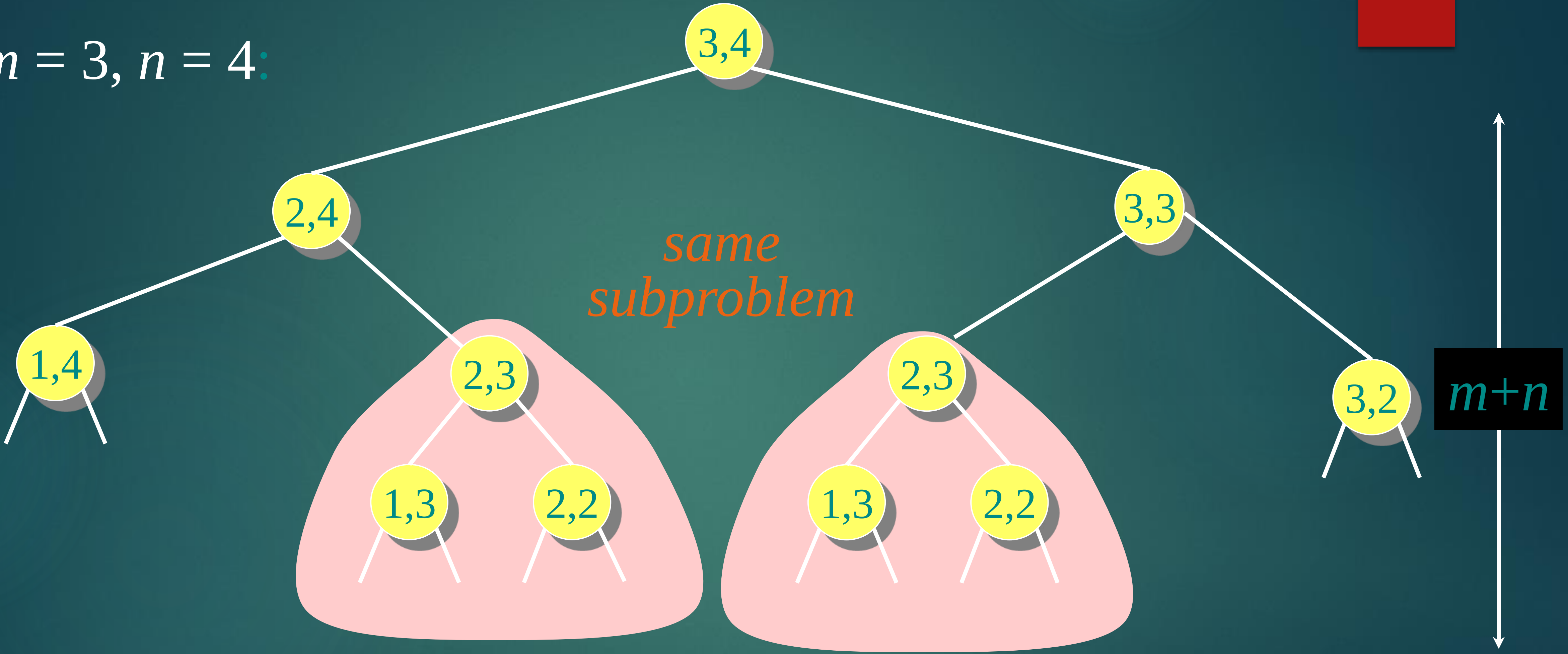
then $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$

else $c[i, j] \leftarrow \max\{\text{LCS}(x, y, i-1, j), \text{LCS}(x, y, i, j-1)\}$

Worst-Case: $x[i] \neq y[j]$, in which case the algorithm evaluates two subproblems, each with only one parameter decremented.

Recursion Tree

$m = 3$, $n = 4$:



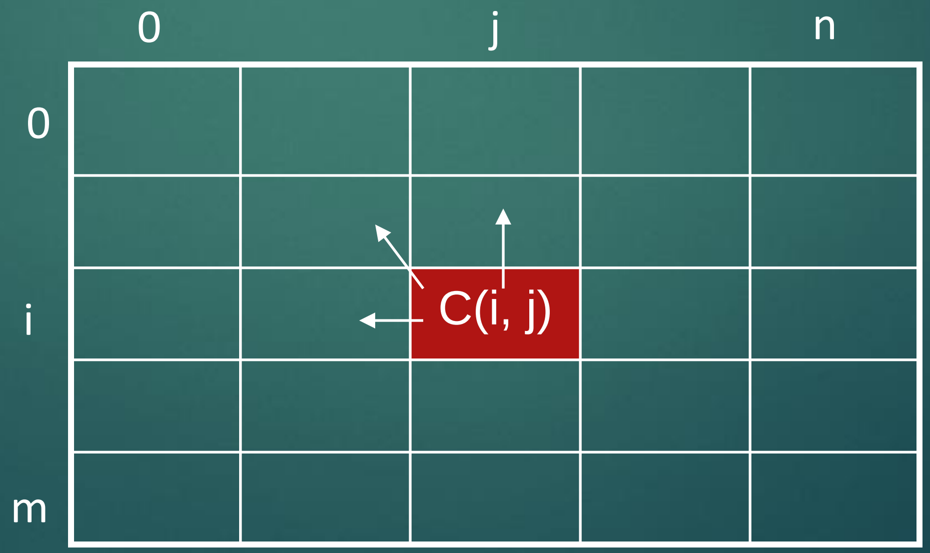
Height $= m + n \Rightarrow$ work potentially exponential.

but we're solving subproblems already solved!

DP Algorithm

- Key: find out the correct order to solve the sub-problems
- Total number of sub-problems: $m * n$

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



DP 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 LCS of X and Y?

LCS(X, Y) = BCB
X = A B C B
Y = B D C A B

Computing the Length of the LCS

$$c[i, j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max(c[i-1, j], c[i, j-1]) & \text{if } x_i \neq y_j \end{cases}$$

		0	1	2	3		n
		Y[j]	1	2	3		Y_n
0	X[i]	**0**	**0**	**0**	**0**	**0**	**0**
1	1	**0**					
2	2	**0**					
3	3	**0**			.		
		0			.		
m	X_m	**0**					

LCS Example (0)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]						
1	**A**						
2	**B**						
3	**C**						
4	**B**						

X = ABCB; m = |X| = 4
Y = BDCAB; n = |Y| = 5
Allocate array c[5][6]

LCS Example (1)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**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)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	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)

X=ABCB

Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**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)

X=ABCB

Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	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)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	**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)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	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)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	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)

X=ABCB
Y=BDCAB

i		j: 0	1	2	3	4	5
		Y[j]	B	D	C	A	B
0	X[i]	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 (9)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**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 (10)

X=ABCB
Y=BDCAB

		j	0	1	2	3	4	5
i			Y[j]	B	D	C	A	B
0	X[i]		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 (11)

X=ABCB
Y=BDCAB

i		j	0	1	2	3	4	5
			Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]		**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 (12)

X=ABCB
Y=BDCAB

i	j X[i]	0 Y[j]	1 B	2 D	3 C	4 A	5 B
0		**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 (13)

X=ABCB
Y=BDCAB

i		j	0	1	2	3	4	5
		Y[j]		B	D	C	A	B
0	X[i]		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 (14)

X=ABCB
Y=BDCAB

	j	0	1	2	3	4	5
i		Y[j]	B	D	C	A	B
0	X[i]	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

- The algorithm just found the *length* of LCS, but not LCS itself.
- How to find the actual LCS?
- For each c[i,j] we know how it was acquired:

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

- A match happens only when the first equation is taken
- So we can start from *c[m,n]* and go backwards, remember *x[i]* whenever *c[i,j] = c[i-1, j-1]+1*.

2	2
2	3

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

Finding LCS

	j	0	1	2	3	4	5
i		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**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**

Time for trace back: $O(m+n)$.

Finding LCS (2)

i	j	0	1	2	3	4	5
		Y[j]	**B**	**D**	**C**	**A**	**B**
0	X[i]	**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)

Dynamic Programming Solution

```
LCS-Length(X, Y)
    m = X.length()
    n = Y.length()
    for i = 1 to m do → c[i,0] = 0
    for j = 0 to n do  → c[0,j] = 0                    O(nm)
    for i = 1 to m do          // row
        for j = 1 to n do          // cloumn
            if x_i = =y_i 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] = “<”
```

Thank You