LONGEST COMMON SUBSEQUENCE
&
DYNAMIC PROGRAMMING

$$X : \overbrace{A\ B\ C\ B\ D\ A\ B}^{n}$$

$$Y : \underbrace{B\ D\ C\ A\ B\ A}_{m \leq n}$$

$$|LCS(X,Y)| = 4$$

Brute force to find LCS:

for every subsequence of Y → $\Theta(2^m)$

check if it exists in X → $O(n)$: easy

$\Theta(2^m)$, $O(n)$ → $O(n \cdot 2^m)$

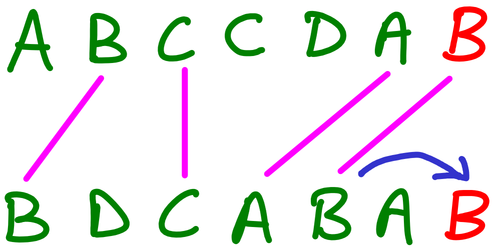
Finding |LCS|

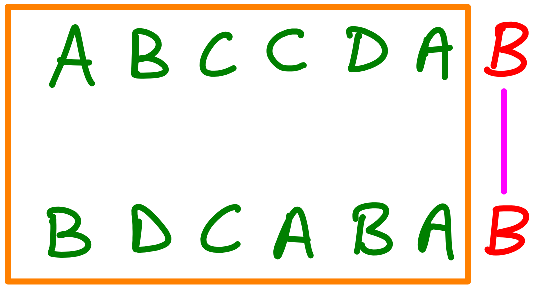
$$c(i,j) = |LCS(X[1\ldots i], Y[1\ldots j])| = \begin{cases} c(i-1,j-1)+1 & \text{if } X[i]=Y[j] \\ & \end{cases}$$

$LCS(X[1\ldots i], Y[1\ldots j])$ must use $X[i]$ or $Y[j]$

otherwise we would improve

Given the above observation...





Slide last match over:
just as good

$c(i-1,j-1)\ +1$

Finding |LCS|

$$c(i,j) = \left|LCS(X[1\ldots i], Y[1\ldots j])\right| = \begin{cases} c(i-1,j-1)+1 & \text{if } X[i]=Y[j] \\ \max\{c(i,j-1), c(i-1,j)\} & \text{otherwise} \end{cases}$$

$LCS(X[1\ldots i], Y[1\ldots j])$

cannot use both $X[i]$ and $Y[j]$

A B C C D A B

B D C A B A

blocked

A B C C D A B

B D C A B A

blocked

Hide each and take best result

$c(i,j-1)$

A B C C D A B

B D C A B ~~A~~

$c(i-1,j)$

A B C C D A ~~B~~

B D C A B A

$$c(i,j) = |LCS(X[1\ldots i], Y[1\ldots j])| = \begin{cases} c(i-1,j-1)+1 & \text{if } X[i]=Y[j] \\ \max\{c(i,j-1), c(i-1,j)\} & \text{otherwise} \end{cases}$$

"optimal substructure" : optimal solutions of subproblems are part of the original problem solution.

LCS(X, Y, i, j)   \\ ignoring base case : if i or j = 0 then $c_{ij} = 0$

if $X_i = Y_j$ then $c_{ij} \leftarrow LCS(X, Y, i-1, j-1) + 1$

else $c_{ij} \leftarrow \max\{LCS(X,Y,i,j-1), LCS(X,Y,i-1,j)\}$

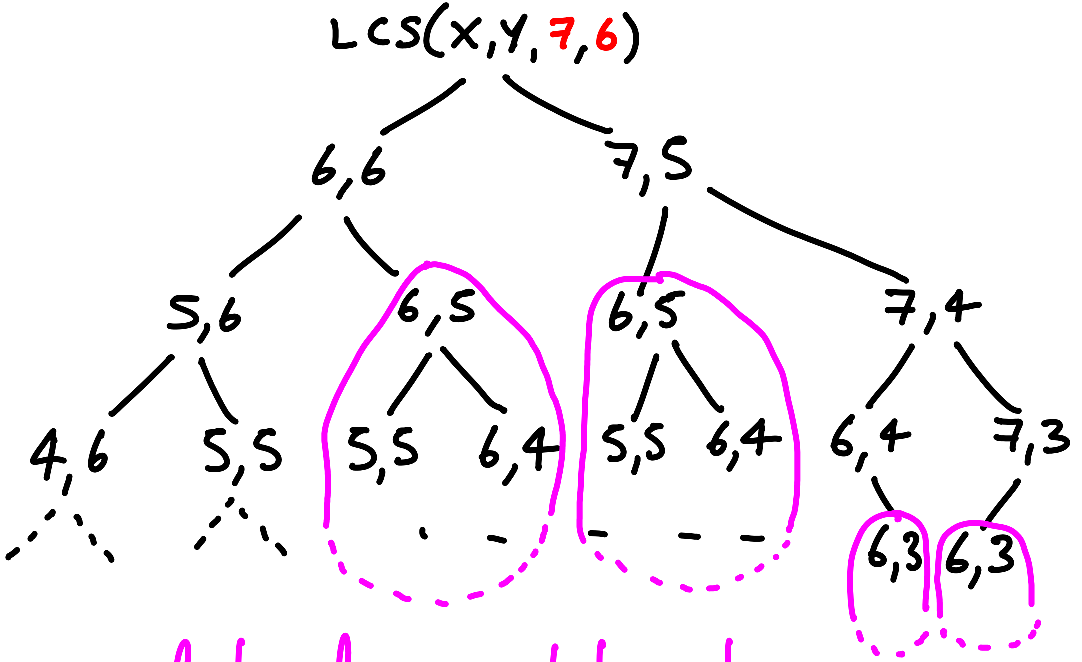
return c_{ij}

LCS(X, Y, i, j)

    if $x_i = y_j$ then $c_{ij} \leftarrow \mathrm{LCS}(X, Y, i-1, j-1) + 1$
    else $c_{ij} \leftarrow \max\{\mathrm{LCS}(X, Y, i, j-1), \mathrm{LCS}(X, Y, i-1, j)\}$
    return c_{ij}

worst case: always get $x_i \neq y_j$

ex: $n = 7$, $m = 6$



lots of repeated work

↳ #distinct subproblems = $m \cdot n$

#full levels $> \min\{m, n\}$

work $= \Omega(2^n)$ if $m \sim n$

Repeated subproblems
+
optimal substructure
↓
try dynamic programming

LCS(X,Y,i,j)

if $X_i = Y_j$ then $c_{ij} \leftarrow$ LCS(X,Y,i-1,j-1) + 1

else $c_{ij} \leftarrow \max\{$LCS(X,Y,i,j-1), LCS(X,Y,i-1,j)$\}$

return c_{ij}

Memoization

Make "memos" of solutions (to subproblems)

Let $c[1...m, 1...n]$ be a $m \times n$ table of -1's.

whenever we need to know c_{ij}

if it's the first time $(c[i,j] = -1)$ then calculate it

else look it up

$\Theta(mn)$ time & space

LCS(X,Y,i,j)

if $\min\{i,j\} = 0$ then return 0

if $c[i,j] = -1$ then \\ first time

if $X_i = Y_j$ then $c[i,j] \leftarrow$ LCS(X,Y,i-1,j-1) + 1

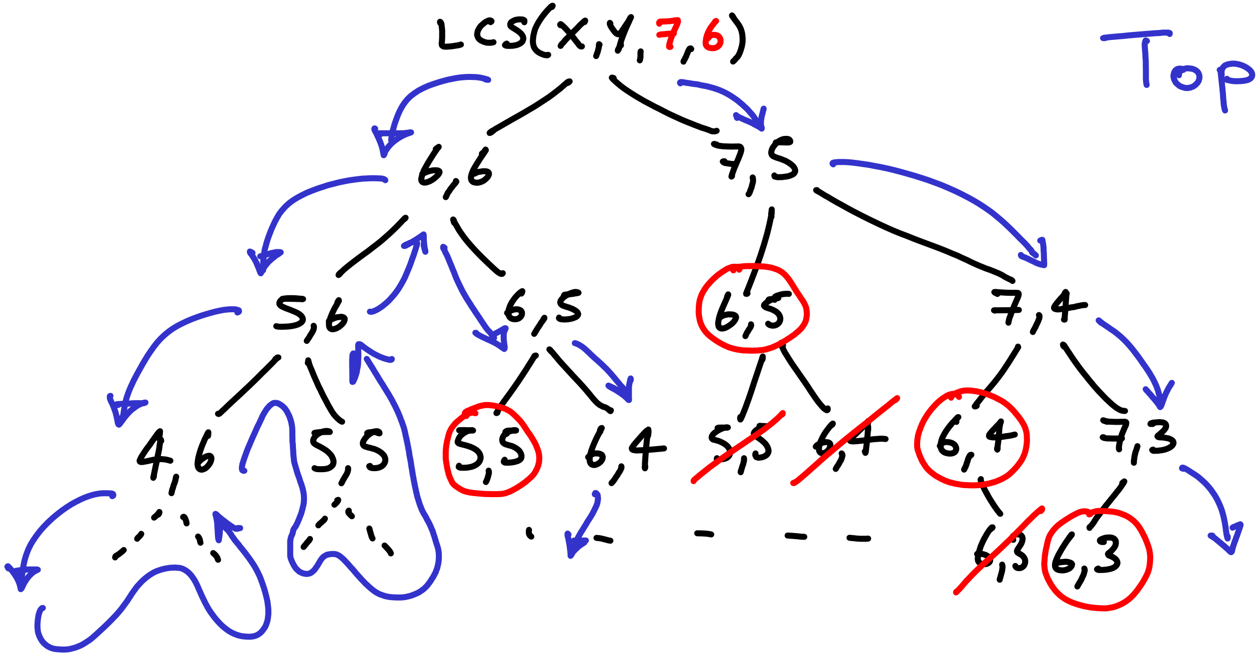
else $c[i,j] \leftarrow \max\{$LCS(X,Y,i,j-1), LCS(X,Y,i-1,j)$\}$

return $c[i,j]$ \\ look up

Memoization

Make "memos" of solutions (to subproblems)

Top-down



DYNAMIC PROGRAMMING

bottom-up

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0							
D	0							
C	0							
A	0							
B	0							
A	0							

base cases

DYNAMIC PROGRAMMING

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

red # : 1 + diag ↖ #

when letters in column & row of # match

green # : max of {above, left}

when letters in column & row of # don't match

DYNAMIC PROGRAMMING

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

BCBA

red # : 1 + diag↖#

when letters in column & row of # match

green # : max of {above, left}

when letters in column & row of # don't match

Trace from C_{mn} to C_{11} to get LCS

DYNAMIC PROGRAMMING

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

BDAB
BCAB

BCBA

red # : 1 + diag↖#

when letters in column & row of # match

green # : max of {above, left}

when letters in column & row of # don't match

Trace from c_{mn} to c_{11} to get LCS

↳ follow mandatory paths ;

optional branches : multiple solutions

$\Theta(mn)$ time & space (+1 trace)

DYNAMIC PROGRAMMING

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D								
C								
A								
B								
A								

red # : 1 + diag↖ #

when letters in column & row of # match

green # : max of {above, left}

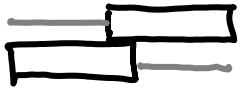
when letters in column & row of # don't match

Trace from c_{mn} to c_{11} to get LCS

↳ follow mandatory paths; optional branches: multiple solutions

$\Theta(mn)$ time & space (+1 trace)

Save space: min{m,n}



DYNAMIC PROGRAMMING

		A	B	C	B	D	A	B
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C								
A								
B								
A								

red # : 1 + diag↖#

when letters in column & row of # match

green # : max of {above, left}

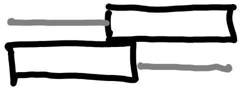
when letters in column & row of # don't match

Trace from c_{mn} to c_{11} to get LCS

↳ follow mandatory paths;
optional branches : multiple solutions

$\Theta(mn)$ time & space (+1 trace)

Save space: $\min\{m,n\}$



DYNAMIC PROGRAMMING

		A	B	C	B	D	A	B
B								
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A								
B								
A								

⋮
etc

red # : 1 + diag↖#

when letters in column & row of # match

green # : max of {above, left}

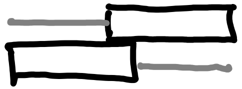
when letters in column & row of # don't match

Trace from C_{mn} to C_{11} to get LCS

↳ follow mandatory paths ; optional branches : multiple solutions

$\Theta(mn)$ time & space (+1 trace)

Save space: min{m,n}



DYNAMIC PROGRAMMING

		A	B	C	B	D	A	B
B								
D								
C					etc			
A					⋮			
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

get |LCS| but not LCS

red # : 1 + diag↖#

when letters in column & row of # match

green # : max of {above, left}

when letters in column & row of # don't match

Trace from C_{mn} to C_{11} to get LCS

↳ follow mandatory paths ; optional branches : multiple solutions

$\Theta(mn)$ time & space (+1 trace)

Save space: min{m,n}