LONGEST COMMON SUBSEQUENCE

&

DYNAMIC PROGRAMMING

LONGEST COMMON SUBSEQUENCE

&

DYNAMIC PROGRAMMING

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

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

LONGEST COMMON SUBSEQUENCE

&

DYNAMIC PROGRAMMING

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

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

$$|LCS(x,y)| = 4$$

LONGEST COMMON SUBSEQUENCE

&

DYNAMIC PROGRAMMING

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

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

$$|LCS(x,y)| = 4$$

LONGEST COMMON SUBSEQUENCE

&

DYNAMIC PROGRAMMING

$$\left.\begin{array}{l} X : \overbrace{A\ B\ C\ B\ D\ A\ B}^{n} \\[2em] Y : \underbrace{B\ D\ C\ A\ B\ A}_{m \leq n} \end{array}\right\} |LCS(x,y)| = 4$$

LONGEST COMMON SUBSEQUENCE

&

DYNAMIC PROGRAMMING

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

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

$$|LCS(x,y)| = 4$$

Brute force to find LCS:

for every subsequence of Y

$$\Theta(2^m)$$

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

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

$O(n \cdot 2^m)$

Finding |LCS|

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

Finding |LCS|

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

Finding |LCS|

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

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

why?

A B C C D A B

B D C A B A B

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

A B C C D A B

B D C A B A B

Finding |LCS|

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

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

otherwise we would improve

A B C C D A B

B D C A B A B

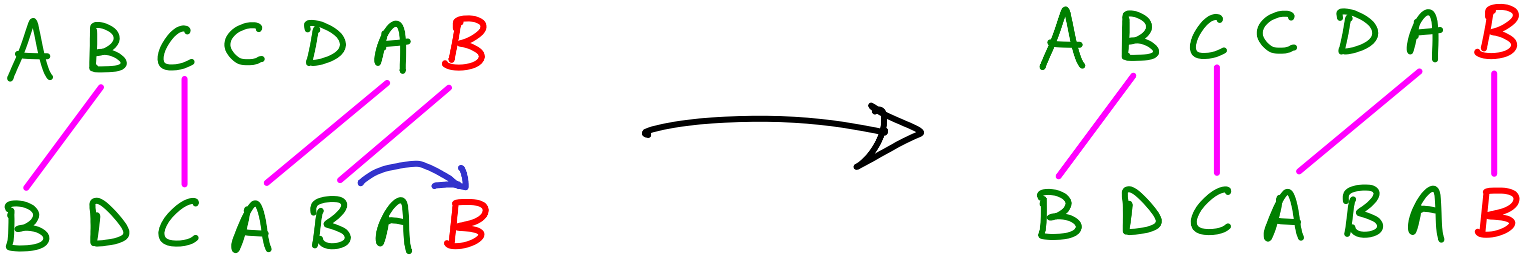
Slide last match over:
just as good

Finding |LCS|

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

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

otherwise we would improve



Slide last match over:
just as good

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

A B C C D A B

B D C A B A B

Slide last match over:
just as good

A B C C D A B

B D C A B A B

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

Finding |LCS|

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

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

blocked

B D C A B A

A B C C D A B

blocked

B D C A B A

Finding |LCS|

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

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

A B C C D A B

B D C A B ~~A~~

A B C C D A ~~B~~

B D C A B A

Finding |LCS|

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

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

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

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

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

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

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

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

LCS(X, Y, i, j)

return c_{ij}

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

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

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

    return c_{ij}

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

"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 ← LCS(X,Y,i-1,j-1) + 1
    else  c_ij ← 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 $c_{ij} \leftarrow LCS(X, Y, i-1, j-1) + 1$

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

return c_{ij}

worst case : ?

LCS(X, Y, i, j)

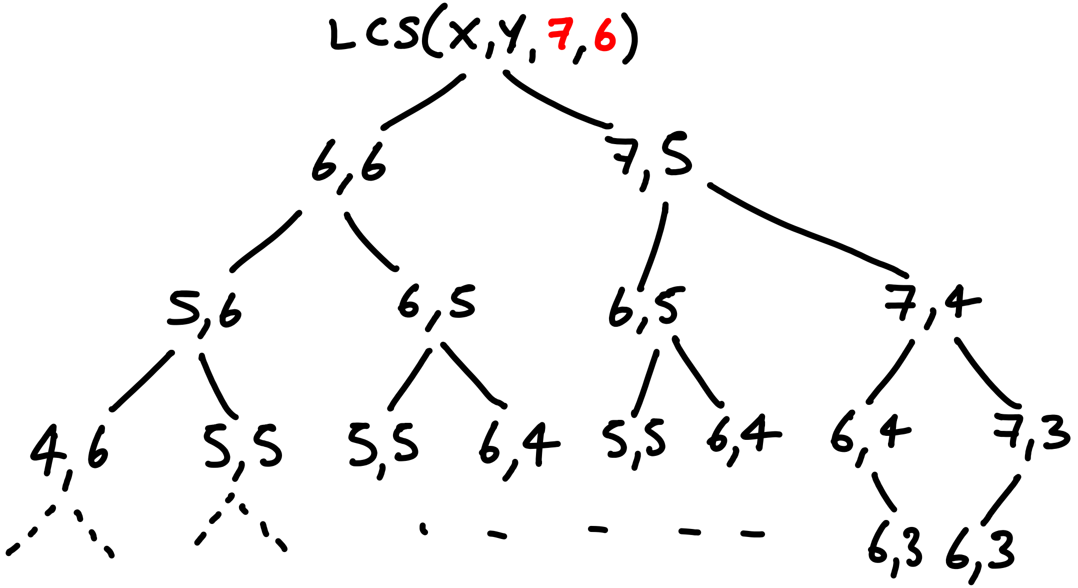
  if $x_i = y_j$ then $c_{ij} \leftarrow$ LCS(X, Y, i−1, j−1) + 1

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

  return c_{ij}

worst case: always get $x_i \neq y_j$

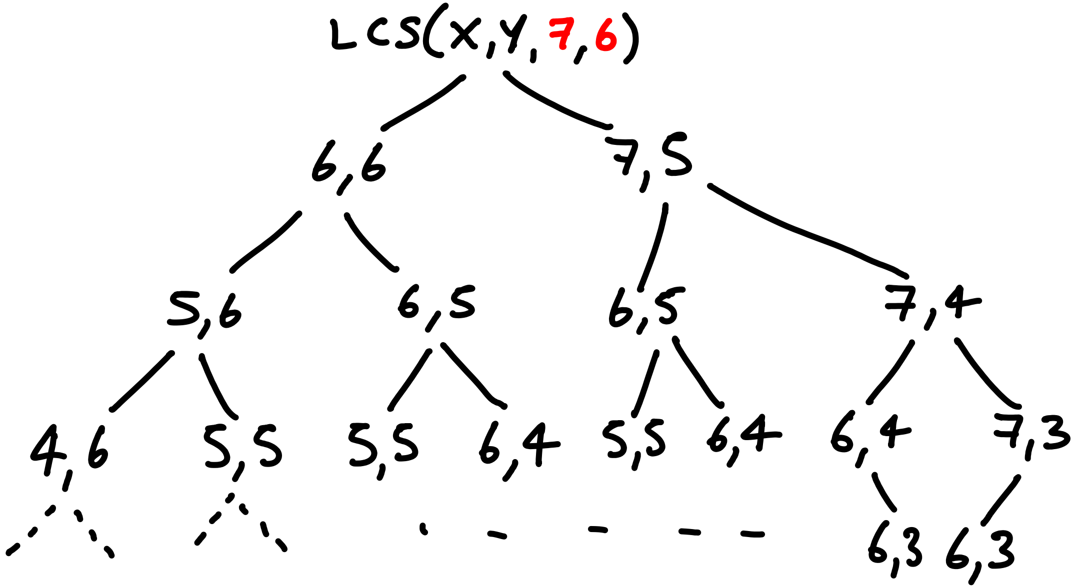
ex: n = 7, m = 6



```
LCS(X,Y,i,j)
    if X_i = Y_j then c_ij ← LCS(X,Y,i-1,j-1) + 1
    else c_ij ← max{LCS(X,Y,i,j-1), LCS(X,Y,i-1,j)}
    return c_ij
```

worst case : always get $x_i \neq y_j$

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



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

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

$LCS(X, Y, i, j)$

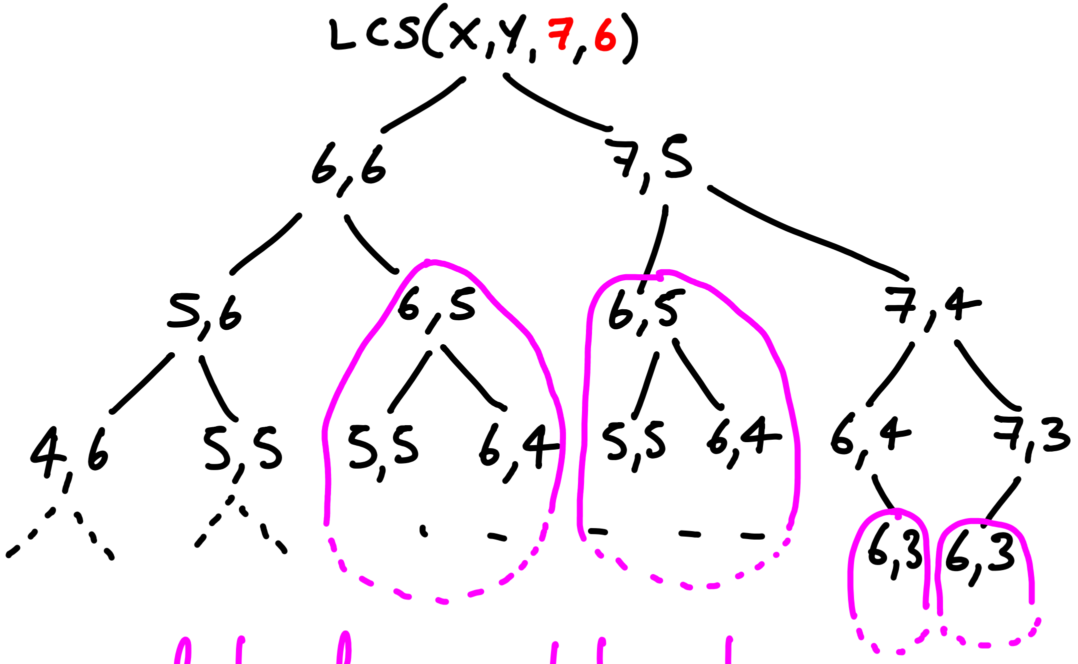
if $x_i = y_j$ then $c_{ij} \leftarrow LCS(X, Y, i-1, j-1) + 1$

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

return c_{ij}

worst case : always get $x_i \neq y_j$

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



lots of repeated work

↳ #distinct subproblems = ?

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

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

LCS(X, Y, i, j)

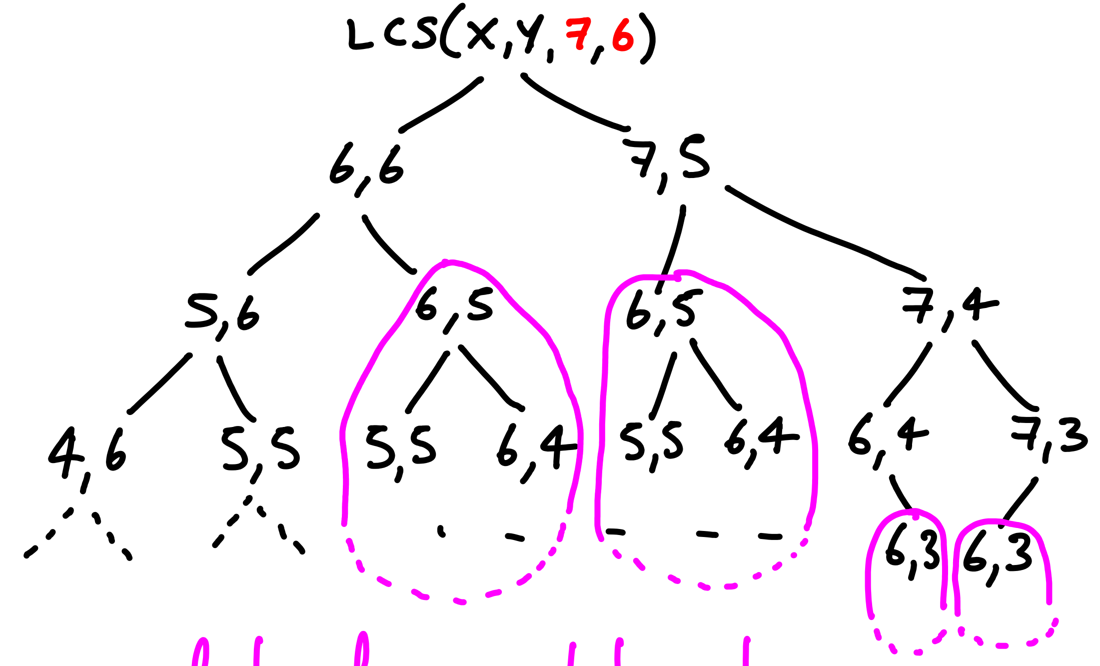
    if $x_i = y_j$ then $c_{ij} \leftarrow$ LCS(X, Y, i−1, j−1) + 1

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

    return c_{ij}

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 $m \sim n$

LCS(X,Y,i,j)

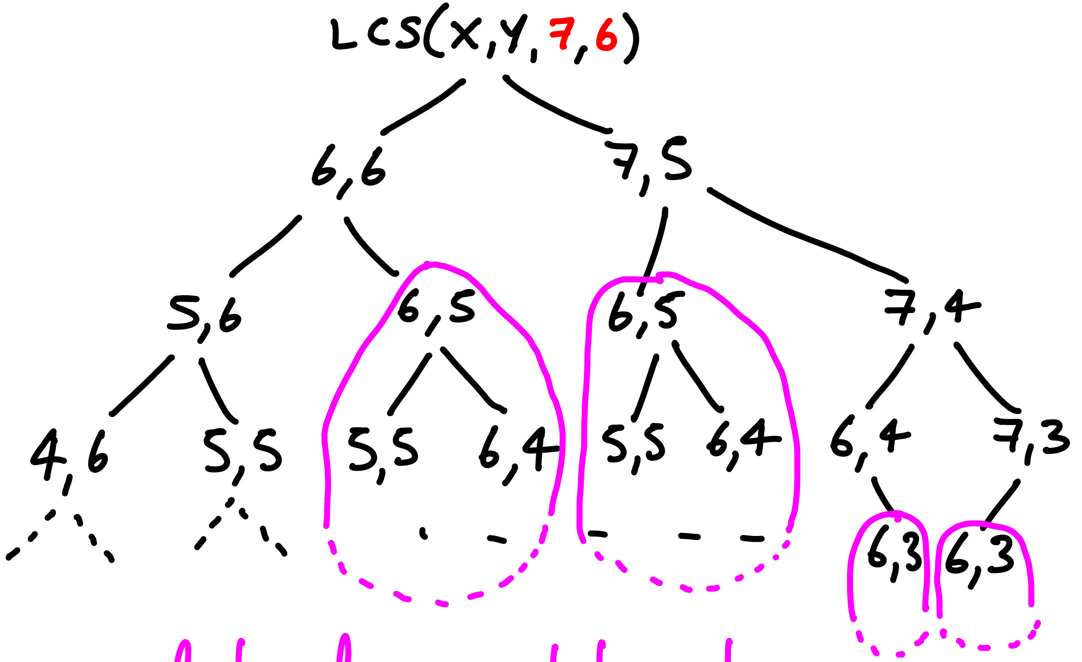
  if $x_i = y_j$ then $c_{ij} \leftarrow \text{LCS}(X,Y,i-1,j-1)+1$

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

  return c_{ij}

worst case : always get $x_i \neq y_j$

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



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

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

Repeated subproblems

+

optimal substructure

↓

try dynamic programming

lots of repeated work

↳ #distinct subproblems $= m \cdot n$

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}

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)

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

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

whenever we need to know c_{ij}

if it's the first time . . . . then calculate it

$LCS(X,Y,i,j)$

  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

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

LCS(X, Y, i, j)

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

$LCS(X,Y,i,j)$

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 $c[1\ldots m, 1\ldots n]$ 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

$LCS(X,Y,i,j)$

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

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

$LCS(X,Y,i,j)$

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 $c[1\ldots m, 1\ldots n]$ 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

$LCS(X,Y,i,j)$

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

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 × 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

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

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)

time ?

Let c[1...m, 1...n] be a m × 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

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

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 $c[1 \ldots m, 1 \ldots n]$ 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

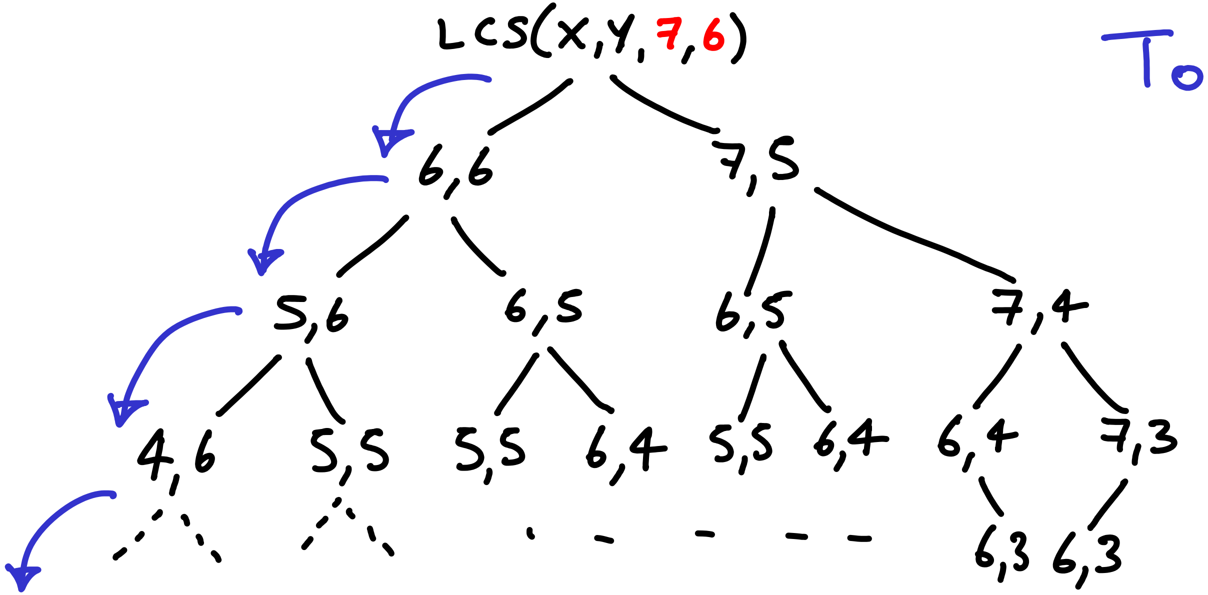
else $c[i,j] \leftarrow \max\{$LCS(X,Y,i,j-1), LCS(X,Y,i-1,j)$\}$

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

Memoization

Make "memos" of solutions (to subproblems)

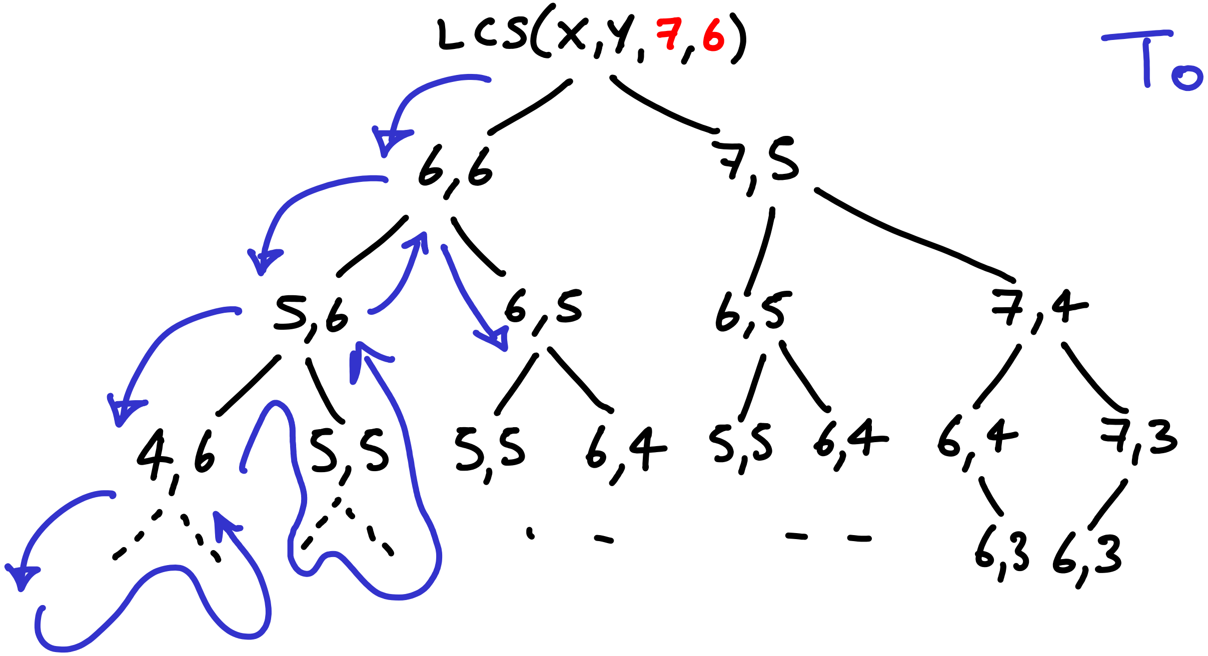
Top-down



Memoization

Make "memos" of solutions (to subproblems)

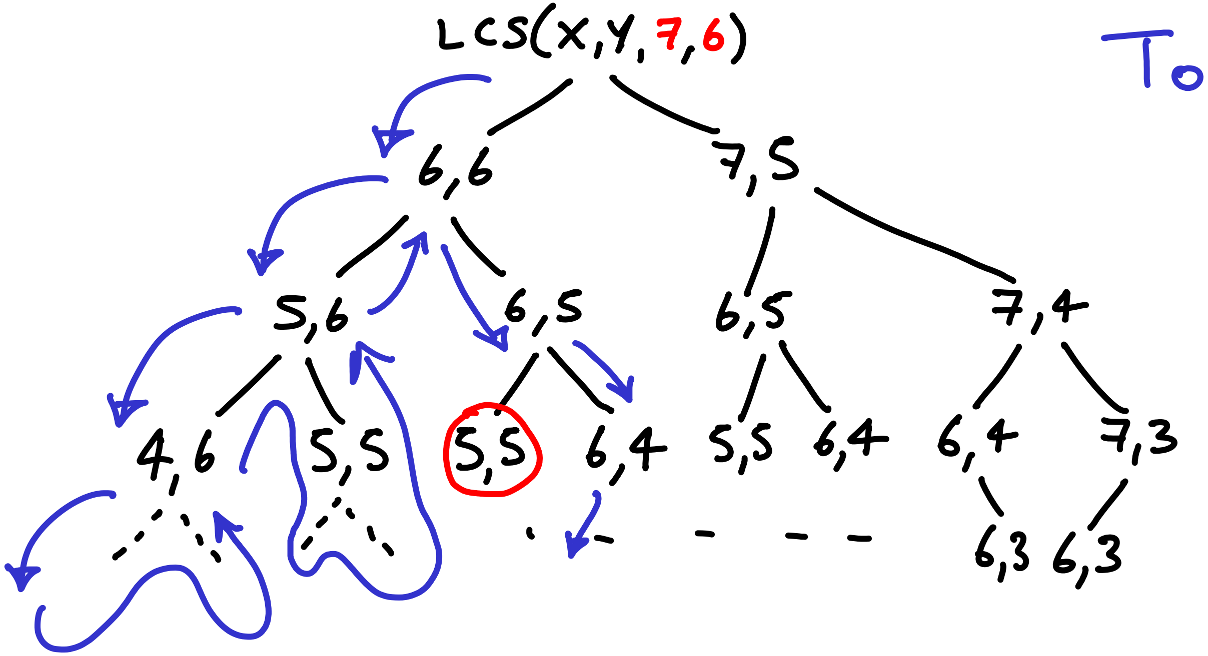
Top-down



Memoization

Make "memos" of solutions
(to subproblems)

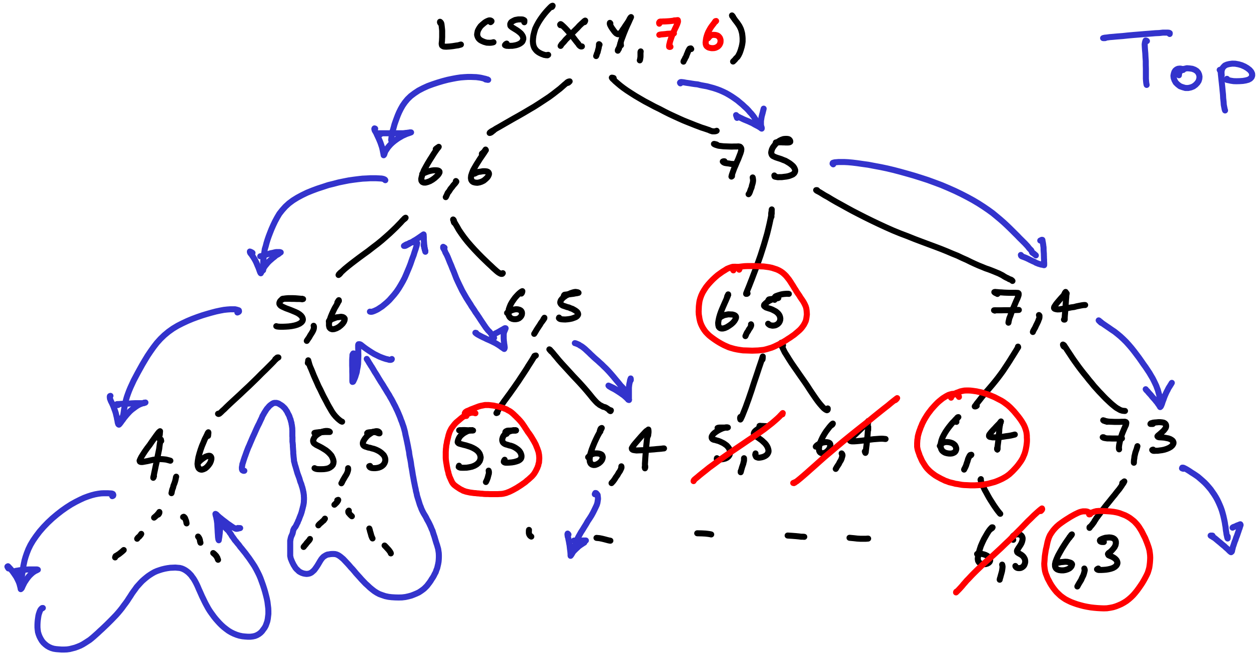
Top-down



Memoization

Make "memos" of solutions (to subproblems)

Top-down



DYNAMIC PROGRAMMING

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						
D	0							
C	0							
A	0							
B	0							
A	0							

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					
D	0							
C	0							
A	0							
B	0							
A	0							

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				
D	0							
C	0							
A	0							
B	0							
A	0							

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			
D	0							
C	0							
A	0							
B	0							
A	0							

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	
D	0							
C	0							
A	0							
B	0							
A	0							

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							
C	0							
A	0							
B	0							
A	0							

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							
A	0							
B	0							
A	0							

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						
A	0	1						
B	0	1						
A	0	1						

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				
A	0	1	1					
B	0	1						
A	0	1						

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			
A	0	1	1	2				
B	0	1	2					
A	0	1						

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

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 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

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

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}

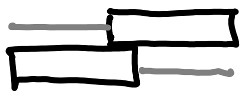
when letters in column & row of # don't match

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}

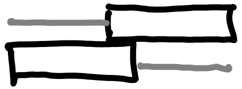
when letters in column & row of # don't match

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

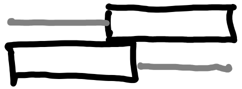
green # : max of {above, left}

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								
A								
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

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

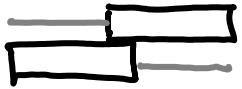
Trace from c_{mn} to c_{11} to get LCS

↳ follow mandatory paths ;

optional branches : multiple solutions

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

Save space: min{m,n}



get |LCS| but not LCS