Dynamic Programming, Longest Common Subsequence

Dynamic programming

Design technique, like divide-and-conquer.

Example: ***Longest Common Subsequence (LCS)***

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

"a" *not* "the"

x: A B C B D A B

y: B D C A B A

BCBA = LCS(x, y)

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

- Checking $= O(n)$ time per subsequence.
- 2^m subsequences of x (each bit-vector of length m determines a distinct subsequence of x).

Worst-case running time $= O(n2^m)$
$= \text{exponential time.}$

Towards a better algorithm

Simplification:

1. Look at the *length* of a longest-common subsequence.
2. Extend the algorithm to find the LCS itself.

Notation: Denote the length of a sequence s by $|s|$.

Strategy: Consider ***prefixes*** of x and y.

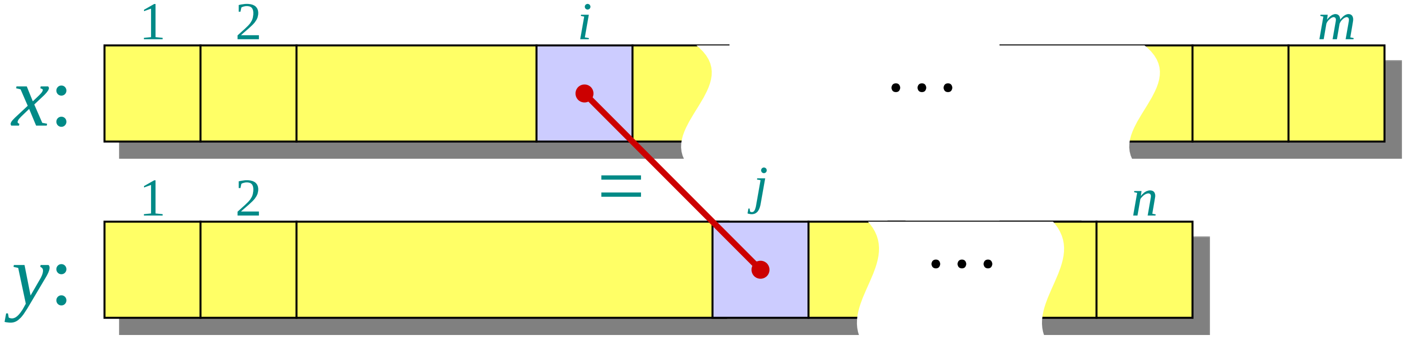
- Define $c[i, j] = |\mathrm{LCS}(x[1\,.\,.\,i], y[1\,.\,.\,j])|$.
- Then, $c[m, n] = |\mathrm{LCS}(x, y)|$.

Recursive formulation

Theorem.

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

Proof. Case $x[i] = y[j]$:



Let $z[1 \,.\,.\, k] = \mathrm{LCS}(x[1 \,.\,.\, i], y[1 \,.\,.\, j])$, where $c[i, j] = k$. Then, $z[k] = x[i]$, or else z could be extended. Thus, $z[i \,.\,.\, k-1]$ is CS of $x[1 \,.\,.\, i-1]$ and $y[1 \,.\,.\, j-1]$.

Proof (continued)

Claim: $z[1\,.\,.\,k-1] = \mathrm{LCS}(x[1\,.\,.\,i-1], y[1\,.\,.\,j-1])$.

Suppose w is a longer CS of $x[1\,.\,.\,i-1]$ and $y[1\,.\,.\,j-1]$, that is, $|w| > k-1$. Then, ***cut and paste***: $w \,\|\, z[k]$ (w concatenated with $z[k]$) is a common subsequence of x and y with $\big|w \,\|\, z[k]\big| > k$. Contradiction, proving claim.

Thus, $c[i-1, j-1] = k-1$, which implies that $c[i, j] = c[i-1, j-1] + 1$.

Other cases are similar. $\square$

Dynamic-programming hallmark #1

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

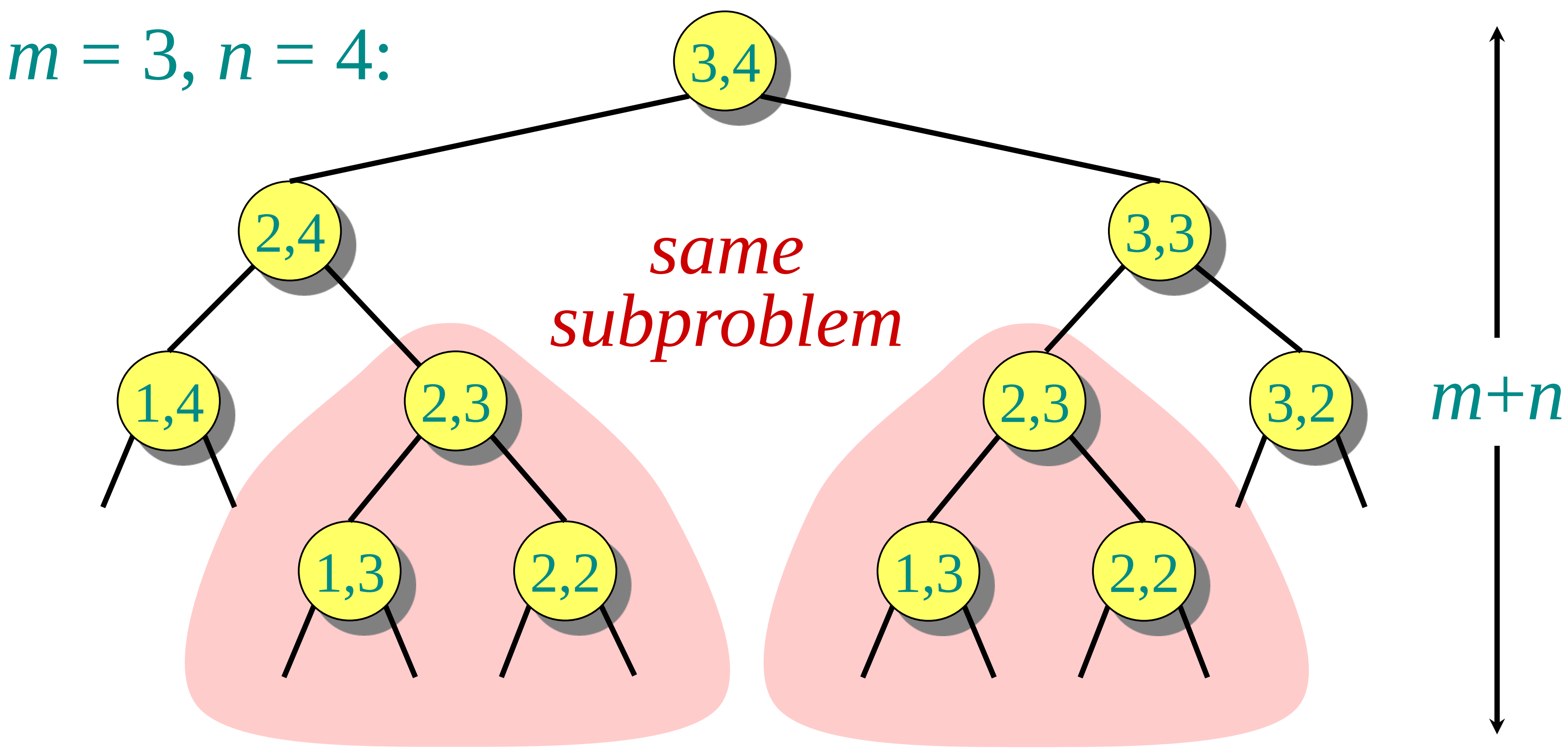
If $z = \text{LCS}(x, y)$, then any prefix of z is an LCS of a prefix of x and a prefix of y.

Recursive algorithm for LCS

$\text{LCS}(x, y, i, j)$
  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



Height = $m + n \Rightarrow$ work potentially exponential, but we're solving subproblems already solved!

Dynamic-programming hallmark #2

Overlapping subproblems
A recursive solution contains a "small" number of distinct subproblems repeated many times.

The number of distinct LCS subproblems for two strings of lengths m and n is only mn.

Memoization algorithm

Memoization: After computing a solution to a subproblem, store it in a table. Subsequent calls check the table to avoid redoing work.

$$
\left.
\begin{array}{l}
\text{LCS}(x, y, i, j) \\
\quad \textbf{if } c[i, j] = \text{NIL} \\
\quad\quad \textbf{then if } x[i] = y[j] \\
\quad\quad\quad \textbf{then } c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1 \\
\quad\quad\quad \textbf{else } c[i, j] \leftarrow \max\left\{ \begin{array}{l} \text{LCS}(x, y, i-1, j), \\ \text{LCS}(x, y, i, j-1) \end{array} \right\}
\end{array}
\right\} \textit{same as before}
$$

Time $= \Theta(mn) =$ constant work per table entry.
Space $= \Theta(mn)$.

LCS Example

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

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

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**				
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

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

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**	
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

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

ABCB

BDCAB

LCS Example

	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

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

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**	
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

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

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**			
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

ABCB
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

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

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

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

ABCB
BDCAB

i	Xi	j: 0 Yj	1 B	2 D	3 C	4 A	5 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 $c[i,j] = \max(c[i-1,j], c[i,j-1])$

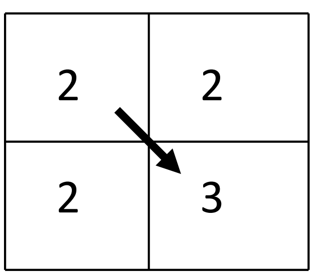
Algorithm design

How to find actual LCS

- So far, we have just found the *length* of LCS, but not LCS itself.
- We want to modify this algorithm to make it output LCS 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:



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

Algorithm design

- Remember that

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

So we can start from *c[m,n]* and go backwards

Whenever $c[i,j]=c[i-1,j-1]+1$, record $x[i]$

When i=0 or j=0 (i.e. we reached the beginning), output recorded 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**

To construct LCS, start in the bottom right corner and follow the arrows. ↖ indicates a matching character

Finding LCS

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**

Dynamic-programming algorithm

IDEA:

Compute the table bottom-up.

Time $= \Theta(mn)$.

		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

Dynamic-programming algorithm

IDEA:

Compute the table bottom-up.

Time $= \Theta(mn)$.

Reconstruct LCS by tracing backwards.

Space $= \Theta(mn)$.

Exercise: $O(\min\{m, n\})$.

		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