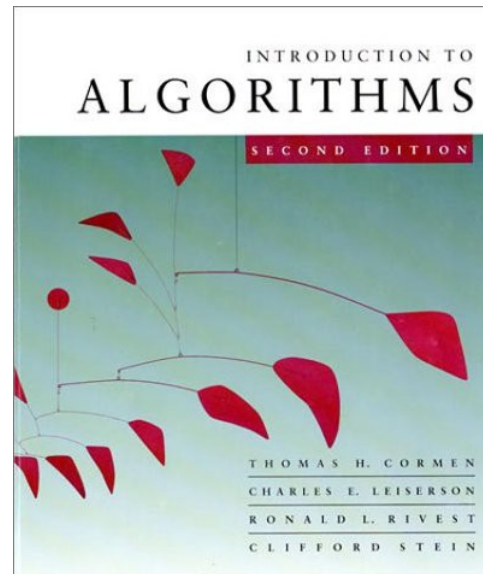


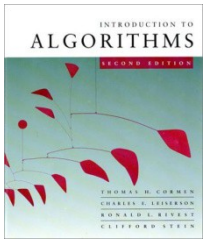
# *Introduction to Algorithms*

## 6.046J/18.401J/SMA5503



## *Lecture 9*

**Based on slides by Prof. Charles E. Leiserson**



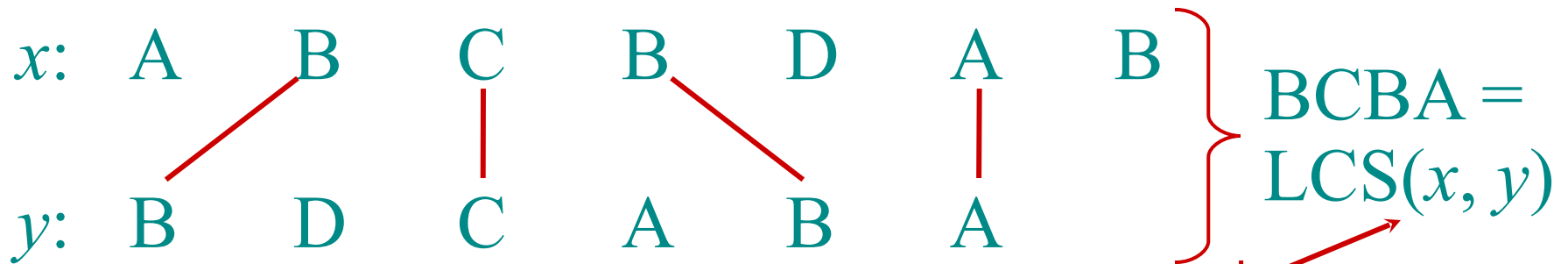
# Dynamic programming

*Design technique, like divide-and-conquer.*

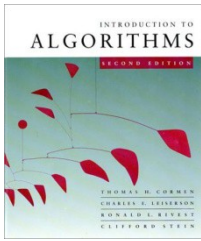
## Example: *Longest Common Subsequence (LCS)*

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

“a” *not* “the”



functional notation,  
but not a function



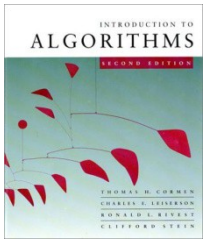
# Brute-force LCS algorithm

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

## 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 =  $O(n2^m)$   
= 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  $y$ .

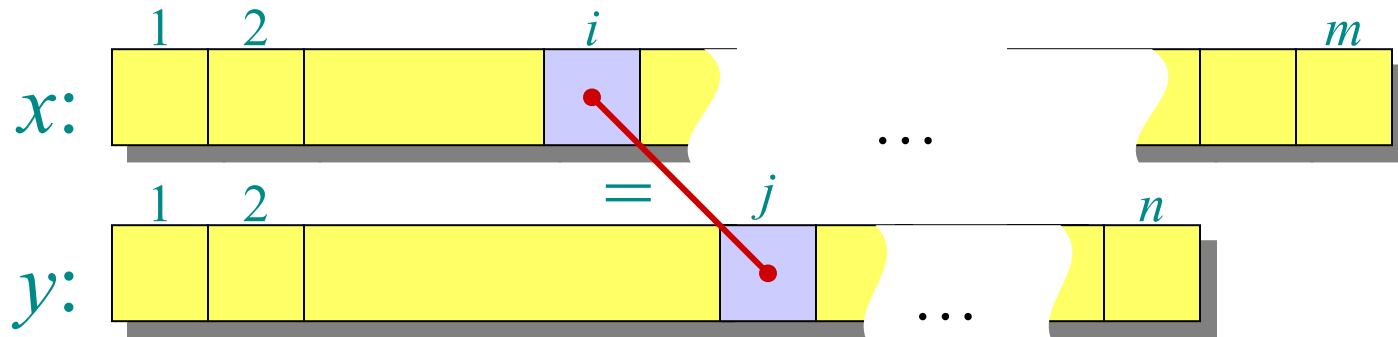
- Define  $c[i, j] = |\text{LCS}(x[1 \dots i], y[1 \dots j])|$ .
- Then,  $c[m, n] = |\text{LCS}(x, y)|$ .

# Recursive formulation

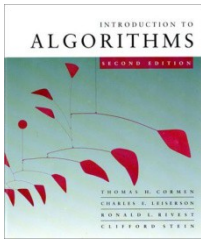
## Theorem.

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

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



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

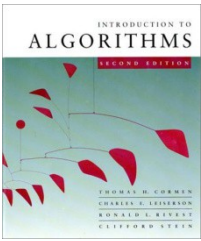


# Proof (continued)

**Claim:**  $z[1 \dots k-1] = \text{LCS}(x[1 \dots i-1], y[1 \dots j-1])$ . Suppose  $w$  is a longer CS of  $x[1 \dots i-1]$  and  $y[1 \dots j-1]$ , that is,  $|w| > k-1$ . Then, **cut and paste:**  $w \parallel z[k]$  ( $w$  concatenated with  $z[k]$ ) is a common subsequence of  $x[1 \dots i]$  and  $y[1 \dots j]$  with  $|w \parallel z[k]| > k$ . Contradiction, proving the 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. □



# Dynamic-programming hallmark #1

## *Optimal substructure*

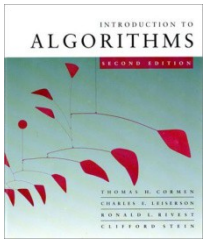
*An optimal solution to a problem (instance) contains optimal solutions to subproblems.*

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



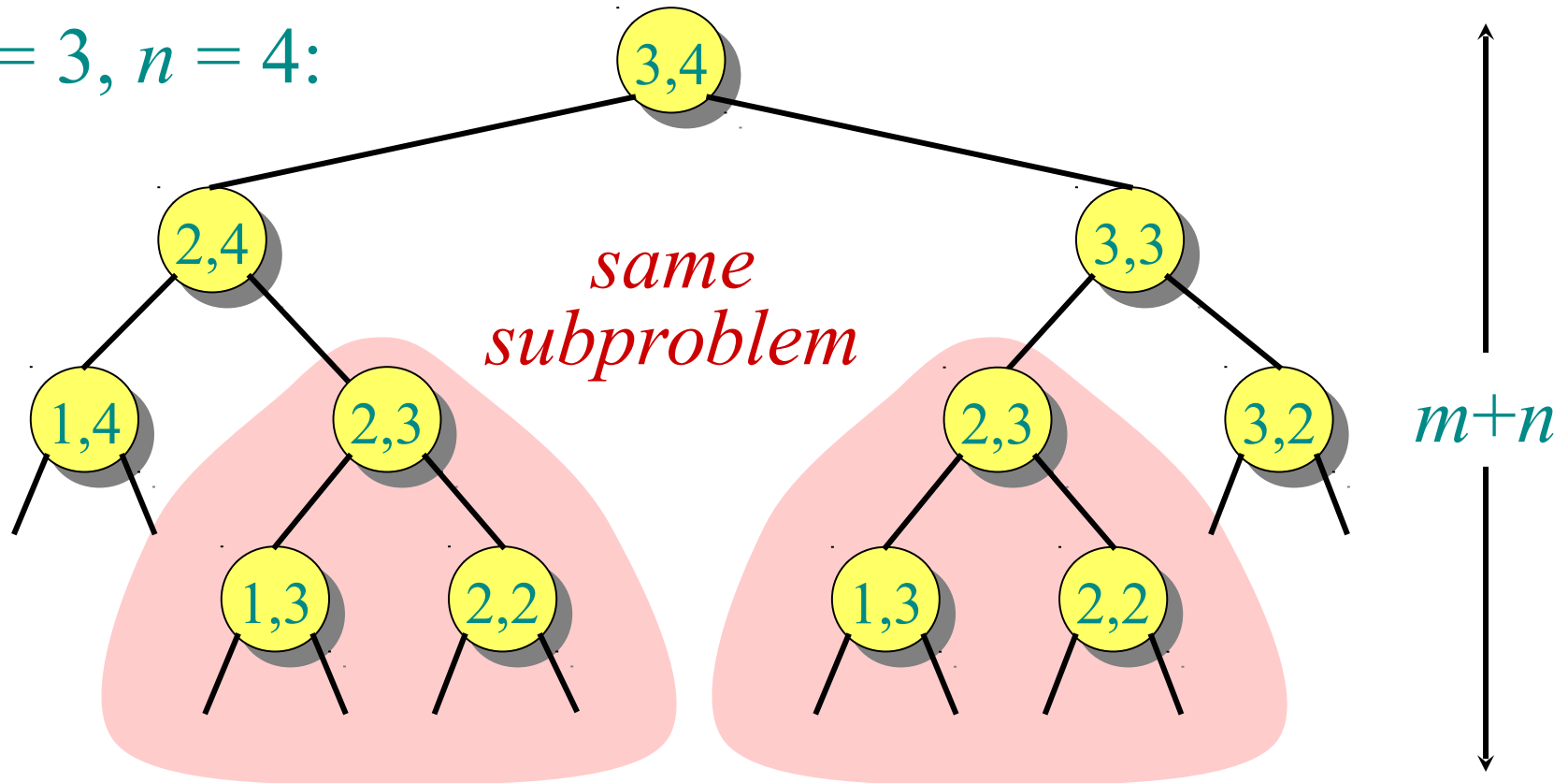
**Worst-case:**  $x[i] \neq y[j]$ , in which case the algorithm evaluates two subproblems, each with only one parameter decremented.



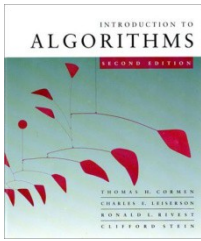


# Recursion tree

$m = 3, n = 4$ :



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

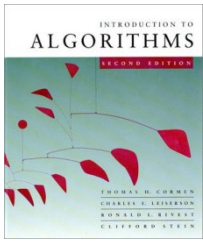


# Dynamic-programming hallmark #2

## *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.

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

if  $c[i, j] = \text{NIL}$

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

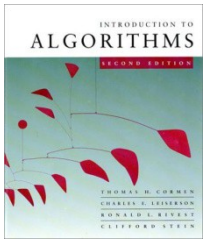
then  $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$

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

} *same  
as  
before*

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

Space =  $\Theta(mn)$ .



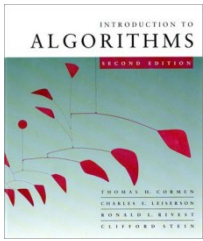
# Dynamic-programming algorithm

## IDEA:

Compute the table bottom-up.

Time =  $\Theta(mn)$ .

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