

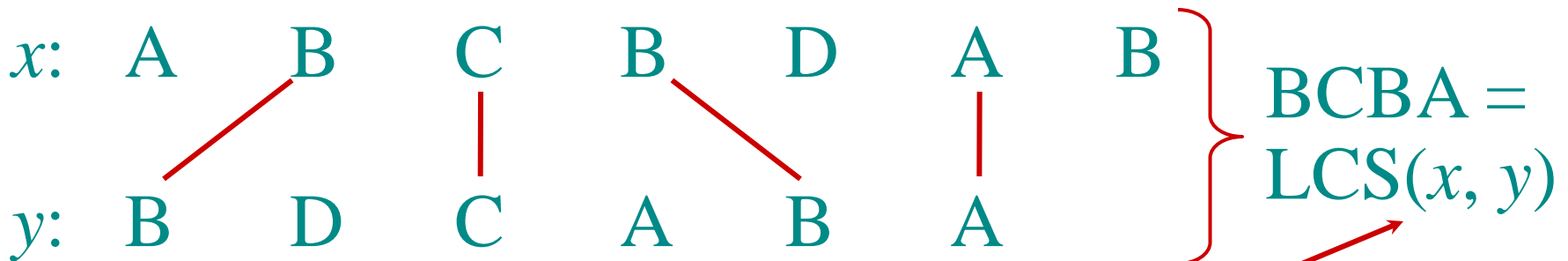
# Dynamic programming

*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”



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,  $c[m, n] = |\text{LCS}(x, y)|$ .

# 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**  $x[i] = y[j]$

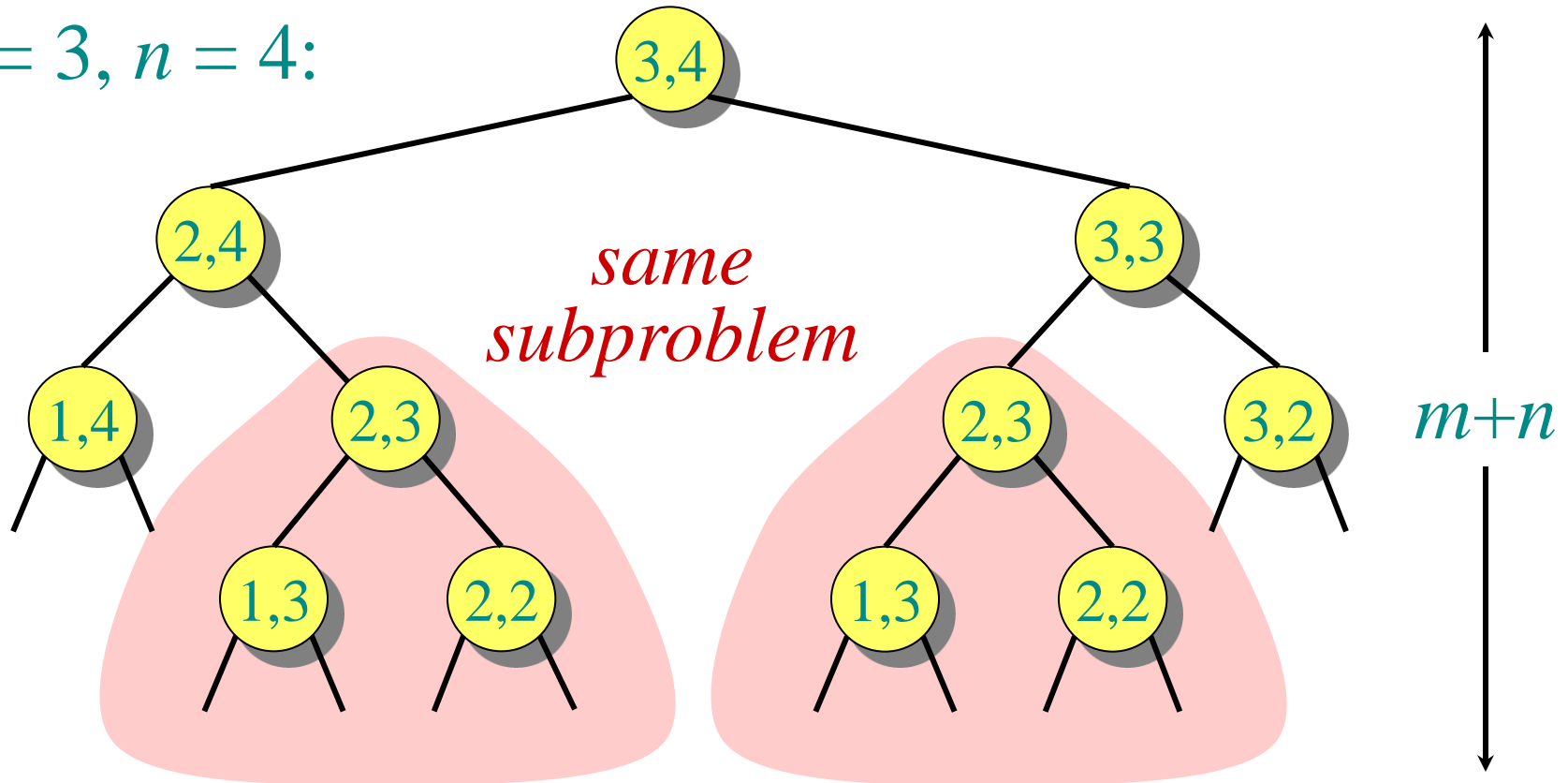
**then**  $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$

**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

$m = 3, n = 4$ :



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.

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