DP Examples

- Fibonacci
- Binomial Coefficients
- Longest Common Subsequence
- Longest Increasing Subsequence
- Knapsack
- Shortest Path
- Edit Distance
- Rod Cutting
- Optimal BST

Longest Common Subsequence

Given two sequences x[1..m] and y[1..n]

$$X = \langle x_1, x_2, ..., x_m \rangle$$
$$Y = \langle y_1, y_2, ..., y_n \rangle$$

find a maximum length common subsequence (LCS) of X and Y

Example

$$X = \langle A, B, C, B, D, A, B \rangle$$

- Subsequences of X:
 - A subset of elements from the sequence taken in order
 (A, B, D), (B, C, D, B), etc.

Longest Common Subsequence (LCS)

Application: Comparison of two DNA strings

Ex: $X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$

Longest Common Subsequence:

X = A BCBDAB

Y = BDCABA

(B, D, B) is a common subsequence with length 3 but is it the longest?

LCS is not unique

$$X = \langle A, B, C, B, D, A, B \rangle$$
 $X = \langle A, B, C, B, D, A, B \rangle$
 $Y = \langle B, D, C, A, B, A \rangle$ $Y = \langle B, D, C, A, B, A \rangle$

- (B, C, B, A) and (B, D, A, B) are longest common subsequences of X and Y (length = 4)
- (B, C, A) is a CS of X and Y but not the longest

Brute-Force Solution

- For every subsequence of X, check whether it's a subsequence of Y
- There are 2^m subsequences of X to check
- Each subsequence takes Θ(n) time to check
 - scan Y for first letter, from there scan for second, and so on
- Running time: Θ(n2^m)

Steps in Dynamic Programming

- 1. Characterize structure of an optimal solution.
- 2. Define value of optimal solution recursively.
- 3. Compute optimal solution values bottom-up in a table.
- Construct an optimal solution from computed values.

We'll study these with the help of examples.

Notations

• Given a sequence $X = \langle x_1, x_2, ..., x_m \rangle$ we define the i-th prefix of X, for i = 0, 1, 2, ..., m

$$X_i = \langle x_1, x_2, ..., x_i \rangle$$
 or $x[1,...,i]$

$$Y_j = \langle y_1, y_2, ..., y_j \rangle$$
 or y[1,...,j]

c[i, j] = the length of a LCS of the sequences

$$X_{i} = \langle x_{1}, x_{2}, ..., x_{i} \rangle$$
 and $Y_{j} = \langle y_{1}, y_{2}, ..., y_{j} \rangle$

Making the choice

$$X = \langle A, B, D, E \rangle$$

 $Y = \langle Z, B, E \rangle$

 Choice: include one element into the common sequence (E) and solve the resulting subproblem

$$X = \langle A, B, D, G \rangle$$

 $Y = \langle Z, B, D \rangle$

 Choice: exclude an element from a string and solve the resulting subproblem

A Recursive Solution

Case 1:
$$x_i = y_j$$

$$X_i = \langle A, B, D, E \rangle$$

$$Y_j = \langle Z, B, E \rangle$$

$$c[i, j] = c[i-1, j-1] + 1$$

- Append $x_i = y_j$ to the LCS of X_{i-1} and Y_{j-1}
- Must find a LCS of X_{i-1} and $Y_{j-1} \Rightarrow$ optimal solution to a problem includes optimal solutions to subproblems

A Recursive Solution

```
Case 2: x_i \neq y_j
X_i = \langle A, B, D, G \rangle
Y_j = \langle Z, B, D \rangle
c[i, j] = \max \{ c[i-1, j], c[i, j-1] \}
```

- Must solve two problems
 - find a LCS of X_{i-1} and Y_i : $X_{i-1} = \langle A, B, D \rangle$ and $Y_j = \langle Z, B, D \rangle$
 - find a LCS of X_i and Y_{j-1} : $X_i = \langle A, B, D, G \rangle$ and $Y_j = \langle Z, B \rangle$
- Optimal solution to a problem includes optimal solutions to subproblems

Overlapping Subproblems

- To find a LCS of X and Y
 - we may need to find the LCS between X and Y_{n-1} and that of X_{m-1} and Y
 - Both the above subproblems has the subproblem of finding the LCS of X_{m-1} and Y_{n-1}
- Subproblems share subsubproblems

LCS Algorithm

- First we'll find the length of LCS. Later we'll modify the algorithm to find LCS itself.
- Define X_i, Y_j to be the prefixes of X and Y of length i and j respectively
- Define c[i,j] to be the length of LCS of X_i and Y_j
- Then the length of LCS of X and Y will be c[m,n]

$$c[i, j] = \begin{cases} 0 & i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } x[i] = y[j], \\ \max(c[i, j-1], c[i-1, j]) & \text{otherwise} \end{cases}$$

LCS recursive solution

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

- We start with i = j = 0 (empty substrings of x and y)
- Since X_0 and Y_0 are empty strings, their LCS is always empty (i.e. c[0,0]=0)
- LCS of empty string and any other string is empty, so for every i and j: c[0, j] = c[i, 0] = 0

LCS recursive solution

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

- When we calculate c[i,j], we consider two cases:
- **First case:** x[i]=y[j]: one more symbol in strings X and Y matches, so the length of LCS X_i and Y_j equals to the length of LCS of smaller strings X_{i-1} and Y_{i-1} , plus 1

LCS recursive solution

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

- Second case: x[i] != y[j]
- As symbols don't match, our solution is not improved, and the length of $LCS(X_i, Y_j)$ is the same as before (i.e. maximum of $LCS(X_i, Y_{j-1})$ and $LCS(X_{i-1}, Y_i)$

LCS Length Algorithm

```
LCS-Length(X, Y)
   m = length(X) // get the # of symbols in X
   n = length(Y) // get the # of symbols in Y
   for i = 1 to m
        c[i,0] = 0 // special case: Y_0
   for j = 1 to n
        c[0,j] = 0 // special case: X_0
   for i = 1 to m
                                         // for all X<sub>i</sub>
        for j = 1 to n
                                         // for all Y<sub>i</sub>
                if (X_i == Y_i)
                         c[i,j] = c[i-1,j-1] + 1
                else c[i,j] = max(c[i-1,j], c[i,j-1])
   return c
```

We'll see how LCS algorithm works on the following example:

```
X = ABCB
```

Y = BDCAB

LCS Example (0)

i		Yj	В	D	С	Α	В
0	Xi						
1	Α						
2	В						
3	С						
4	В						

$$X = ABCB$$
; $m = |X| = 4$
 $Y = BDCAB$; $n = |Y| = 5$
Allocate array $c[4,5]$

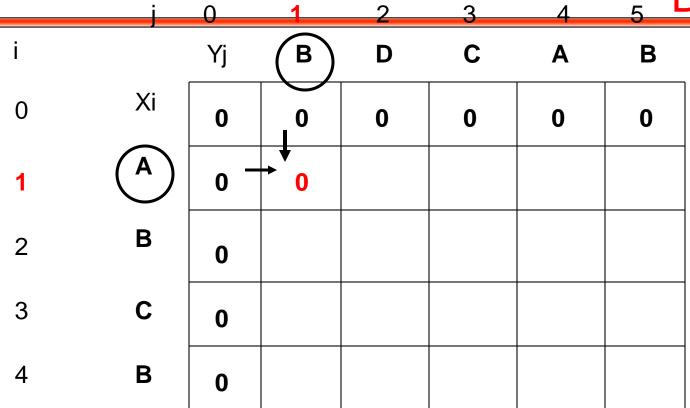
LCS Example (1)

	i_	_0	1	2	3	4	5
i		Yj	В	D	С	Α	В
0	Xi	0	0	0	0	0	0
1	Α	0					
2	В	0					
3	С	0					
4	В	0					

for
$$i = 1$$
 to m $c[i,0] = 0$
for $j = 1$ to n $c[0,j] = 0$

LCS Example (2)





if
$$(X_i == Y_j)$$

 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$

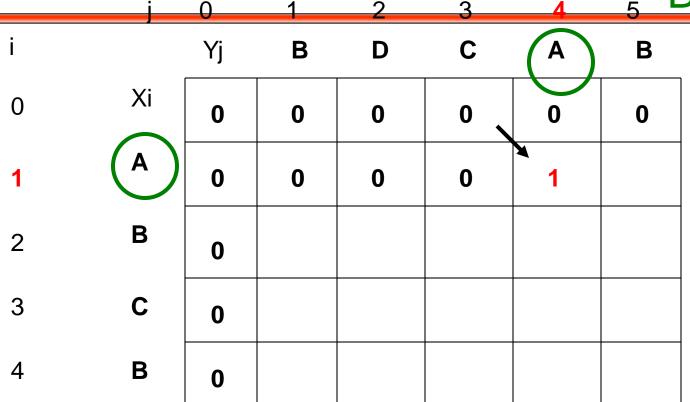


	i	0	_1	2	3	4	_5
i		Yj	В	D	С	Α	В
0	Xi	0	0	0	0	0	0
1	Α	0	0	0	0		
2	В	0					
3	С	0					
4	В	0					

if
$$(X_i == Y_j)$$

 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$





if
$$(X_i == Y_j)$$

 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$

	i	0	1	2	3	4	5	טטט
i		Yj	В	D	С	Α	В	
0	Xi	0	0	0	0	0	0	
1	A	0	0	0	0	1 -	1	
2	В	0						
3	С	0						
4	В	0						

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

		0	1	2	3	4	_5
i		Yj	B	D	С	Α	В
0	Xi	0	0	0	0	0	0
1	Α	0	0	0	0	1	1
2	В	0	1				
3	С	0					
4	В	0					

$$\begin{array}{c} \text{if (} X_i == Y_j \text{)} \\ c[i,j] = c[i\text{-}1,j\text{-}1] + 1 \\ \text{else c[i,j]} = \max(\text{ c[i\text{-}1,j], c[i,j\text{-}1] }) \end{array}$$

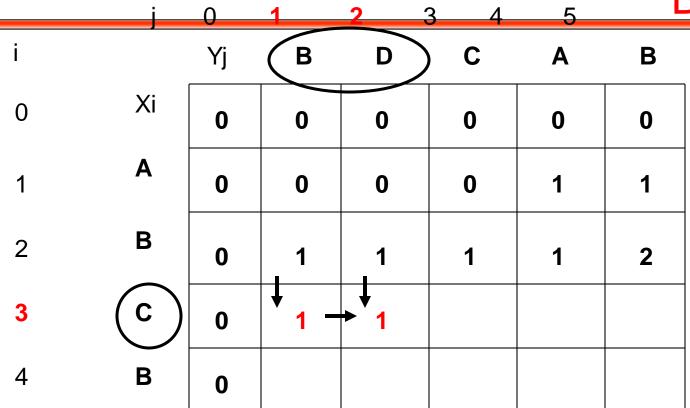
		0	_1	2	3	4	_5
i	•	Yj	В	D	С	A) B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	B	0	1 -	1 -	1	1	
3	С	0					
4	В	0					

$$\begin{aligned} &\text{if (} X_i == Y_j \text{)} \\ & c[i,j] = c[i\text{-}1,j\text{-}1] + 1 \\ &\text{else c}[i,j] = \max(\ c[i\text{-}1,j],\ c[i,j\text{-}1]\) \end{aligned}$$

		_0	_1	2	3 4	5_	
i		Yj	В	D	С	Α	В
0	Xi	0	0	0	0	0	0
1	Α	0	0	0	0	1 ,	1
2	В	0	1	1	1	1	2
3	С	0					
4	В	0					

$$\begin{array}{c} \text{if (} X_i == Y_j \text{)} \\ c[i,j] = c[i\text{-}1,j\text{-}1] + 1 \\ \text{else c[i,j]} = \max(\text{ c[i\text{-}1,j], c[i,j\text{-}1] }) \end{array}$$





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



		0	1	2	3 4	5	
i		Yj	В	D	(c)	Α	В
0	Xi	0	0	0	0	0	0
1	Α	0	0	0	0	1	1
2	В	0	1	1	1	1	2
3	C	0	1	1	2		
4	В	0					

if
$$(X_i == Y_j)$$

 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$



	i	0	1	2 3	3 4	5		<u>ر</u>
i		Yj	В	D	С	A	В)
0	Xi	0	0	0	0	0	0	
1	Α	0	0	0	0	1	1	
2	В	0	1	1	1	1	2	
3	C	0	1	1	2 -	→ 2 −	2	
4	В	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 (13)



		0		2	34	5	
i		Yj	В	D	С	Α	В
0	Xi	0)	0	0	0	0
1	Α	0	0	0	0	1	1
2	В	0	1	1	1	1	2
3	С	0	1	1	2	2	2
4	В	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 (14)



	<u> </u>	0	1	2	3 4	5_	
i		Yj	В	D	С	A	В
0	Xi	0	0	0	0	0	0
1	Α	0	0	0	0	1	1
2	В	0	1	1	1	1	2
3	С	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 (15)



			1	2	3 4		
i		Yj	В	D	С	Α	В
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	В	0	1	1	1	1	2
3	С	0	1	1	2	2	2
4	В	0	1	1	2	2	3

if
$$(X_i == Y_j)$$

 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = max(c[i-1,j], c[i,j-1])$

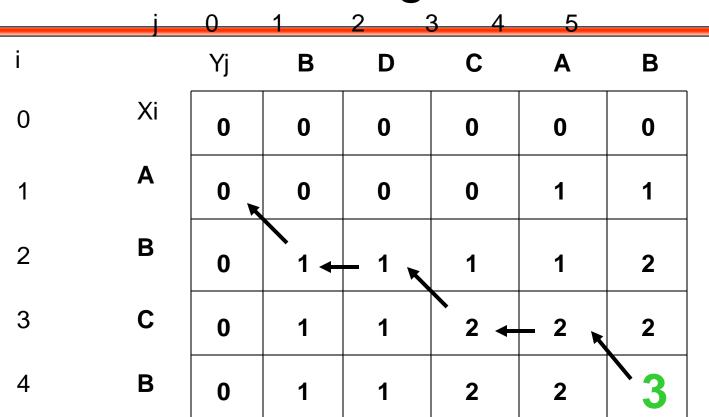
LCS Algorithm Running Time

- LCS algorithm calculates the values of each entry of the array c[m,n]
- So what is the running time?

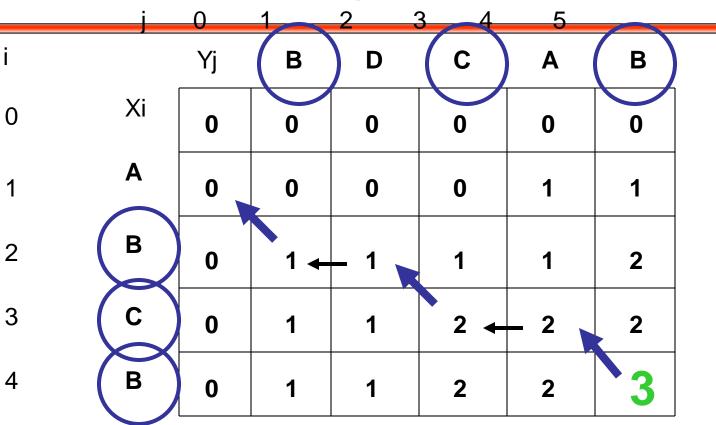
O(mn)

since each c[i,j] is calculated in constant time, and there are m*n elements in the array

Finding LCS



Finding LCS (2)



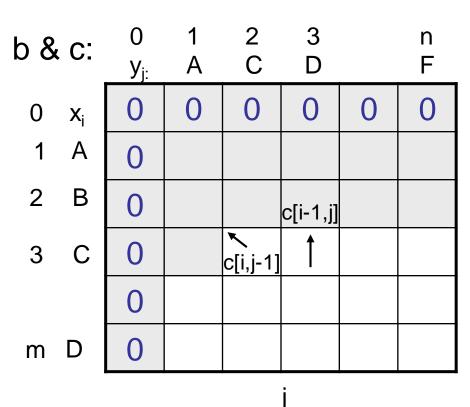
LCS (reversed order): B C B

LCS (straight order):

B C B

Additional Information

$$c[i, j] = \begin{cases} 0 & \text{if } i, j = 0 \\ c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ max(c[i, j-1], c[i-1, j]) & \text{if } x_i \neq y_j \end{cases}$$



A matrix b[i, j]:

- For a subproblem [i, j] it tells us what choice was made to obtain the optimal value
- If $x_i = y_j$ b[i, j] = " "
- Else, if c[i 1, j] ≥ c[i, j-1]
 b[i, j] = "↑"

else

$$b[i, j] = " \leftarrow "$$

Example

В

Constructing a LCS

- Start at b[m, n] and follow the arrows
- When we encounter a "

 "

 " in b[i, j] ⇒ x_i = y_j is an element of the LCS

		0	1	2	3	4	5	6
	_	y _i	В	D	С	Α	В	Α
0	X _i	0	0	0	0	0	0	0
1	Α	0	← 0	← 0	←0	× ~	← 1	1
2	В	0	1	(1)	←1	↑ 1	2	←2
3	С	0	1→	_→(2	€2	↑ 2	↑ 2
4	В	0	× ~	← 1) ←2) ←2	<u>(3</u>	← 3
5	D	0	←←	× 2	↑ 2	← 2	<(∞	↑ 3
6	Α	0	←1	↑ 2	↑ 2	√ [∞])←თ	4
7	В	0	1	↑ 2	↑ 2	↑ 3	4	4

PRINT-LCS(b, X, i, j)

```
if (i = 0 or j = 0)
                                        Running time: \Theta(m + n)
  return
if (b[i, j] = ^{*} ") {
        PRINT-LCS(b, X, i - 1, j - 1)
        print x<sub>i</sub>
elseif ( b[i, j] = "↑") {
        PRINT-LCS(b, X, i - 1, j)
} else {
        PRINT-LCS(b, X, i, j - 1)
Initial call: PRINT-LCS(b, X, length[X], length[Y])
```

Improving the Code

- If we only need the length of the LCS
 - LCS-LENGTH works only on two rows of c at a time.
 The row being computed and the previous row
 - We can reduce the asymptotic space requirements by storing only these two rows