Dynamic Programming Longest Common Subsequence

ANINDITA KUNDU
LECTURER
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MILITARY INSTITUTE OF SCIENCE AND TECHNOLOGY

Longest Common Subsequence

A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous.

For example:

X = “abcdefg”

“abc”, “abg”,

“bdf”, “aeg”,

”acefg”, .. etc are subsequences of X.

Longest Common Subsequence

Given two sequences

$X = \langle x_1, x_2, \ldots, x_m \rangle$

$Y = \langle y_1, y_2, \ldots, y_n \rangle$

A subsequence of a given sequence is just the given sequence with zero or more elements left out.

A common subsequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ of X *and* Y

- Z is a subsequence of both X and Y

Example:

X = A **B** **C** **B** D **A** B

Y = **B** D **C** A **B** **A**

Goal: Find the Longest Common Subsequence (LCS)

Develop a Recursive Solution

- Define $c[i, j]$ to be the length of an LCS of the sequences X_i and Y_j .
  - Goal: Find $c[m, n]$
  - Basis: $c[i, j] = 0$ if either $i = 0$ or $j = 0$
  - Recursion: How to define $c[i, j]$ recursively ?

- Finding an LCS of $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$
  - If $x_m = y_n$, then we must find an LCS of X_{m-1} and Y_{n-1}.
    - Appending $x_m = y_n$ to this LCS yields an LCS of X and Y.
  - If $x_m \neq y_n$, then we must solve two subproblems:
    - Finding an LCS of X_{m-1} and Y
    - Finding an LCS of X and Y_{n-1}
    - Whichever of these two LCSs is longer is an LCS of X and Y.

Develop a Recursive Solution

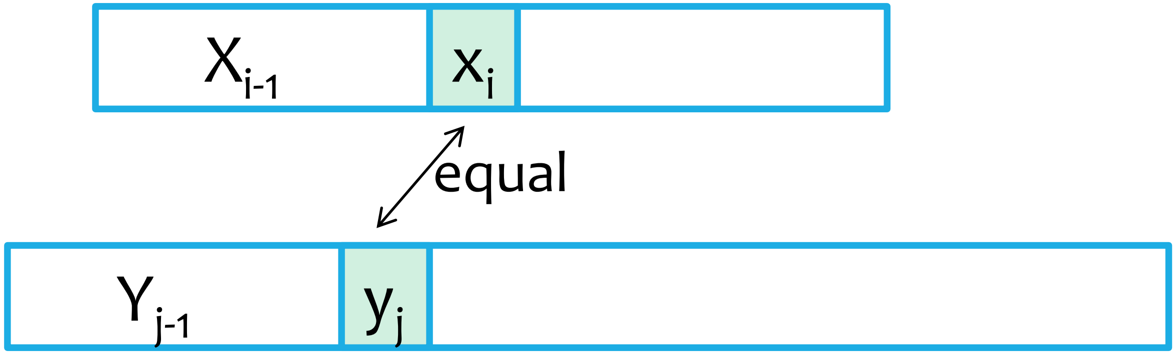
- The recursive formula is

$$c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x[i] = y[j], \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i, j > 0 \text{ and } x[i] \neq y[j] \end{cases}$$

Develop a Recursive Solution

Case 1: $x_i = y_j$

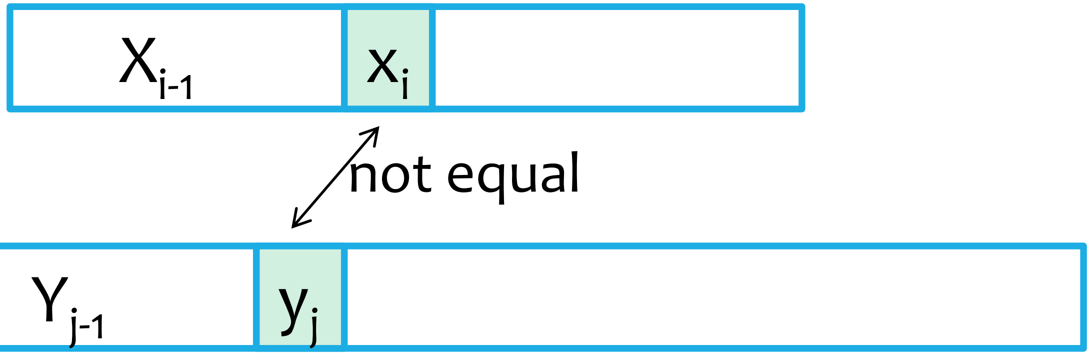
- Recursively find LCS of X_{i-1} and Y_{j-1} and append x_i
- So $c[i, j] = c[i-1, j-1] + 1$ if $i, j > 0$, and $x_i = y_j$



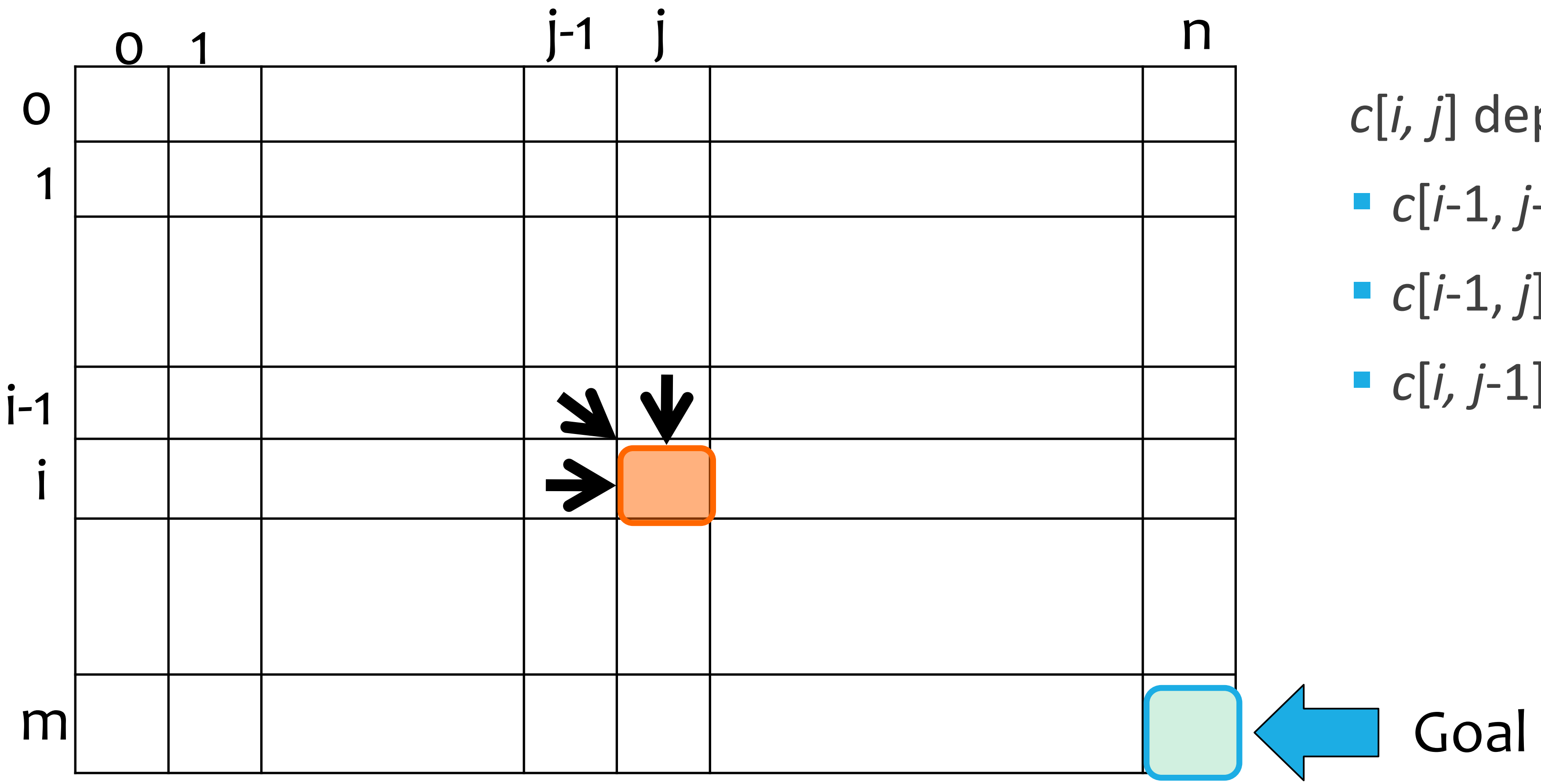
Develop a Recursive Solution

Case 2: $x_i \neq y_j$

- Recursively find LCS of X_{i-1} and Y_j
- Recursively find LCS of X_i and Y_{j-1}
- Take the longer one
- So $c[i, j] = \max\{c[i, j-1], c[i-1, j]\}$ if $i, j > 0$, and $x_i \neq y_j$



Dependencies among Sub-problems



$c[i, j]$ depends on

- $c[i\text{-}1, j\text{-}1]$,
- $c[i\text{-}1, j]$, and
- $c[i, j\text{-}1]$

- An order for solving the sub-problems (*i.e.*, filling in the array) that respects the dependencies is row major order:
  - do the rows from top to bottom
  - inside each row, go from left to right

Develop a Recursive Solution

```
LCS-LENGTH(X, Y)
m ← length[X]
n ← length[Y]
for i ← 1 to m
        do c[i, 0] ← 0
for j ← 0 to n
        do c[0, j] ← 0
for i ← 1 to m
        do for j ← 1 to n
                do if x_i = y_j
                        then c[i, j] ← c[i − 1, j − 1] + 1
                             b[i, j] ← "↖"
                        else if c[i − 1, j] ≥ c[i, j − 1]
                                then c[i, j] ← c[i − 1, j]
                                     b[i, j] ← "↑"
                                else c[i, j] ← c[i, j − 1]
                                     b[i, j] ← "←"
return c and b
```

- The algorithm calculates the values of each entry of the array $c[m, n]$.
- Each $c[i, j]$ is calculated in constant time, and there are $m \cdot n$ elements in the array.
- So the running time is $O(m \cdot n)$.

LCS Example

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

X = ABCG

Y = BDCAG

LCS(X, Y) = BCG

X = A **B** **C** **G**

Y = **B** D **C** A **G**

LCS Example

ABCG
BDCAG

i		j	0	1	2	3	4	5
		y_j		**B**	**D**	**C**	**A**	**G**
0	x_i							
1	**A**							
2	**B**							
3	**C**							
4	**G**							

X = ABCG; $m = |X| = 4$

Y = BDCAG; $n = |Y| = 5$

Allocate array: $c[5, 4]$

LCS Example

ABCG
BDCAG

i		j	0	1	2	3	4	5
		y_j		**B**	**D**	**C**	**A**	**G**
0	x_i		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**					
2	**B**		**0**					
3	**C**		**0**					
4	**G**		**0**					

for $i = 0$ to m $\quad c[i, 0] = 0$

for $j = 1$ to n $\quad c[0, j] = 0$

ABCG

BDCAG

LCS Example

i		j	0	1	2	3	4	5
		y_j		**B**	**D**	**C**	**A**	**G**
0	x_i		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**				
2	**B**		**0**					
3	**C**		**0**					
4	**G**		**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

ABCG
BDCAG

	j	0	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**G**
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**		
2	**B**	**0**					
3	**C**	**0**					
4	**G**	**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

ABCG

BDCAG

	j	0	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**G**
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	
2	**B**	**0**					
3	**C**	**0**					
4	**G**	**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

ABCG
BDCAG

i		j	0	1	2	3	4	5
		y_j		**B**	**D**	**C**	**A**	**G**
0	x_i		**0**	**0**	**0**	**0**	**0**	**0**
1	**A**		**0**	**0**	**0**	**0**	**1** →	**1**
2	**B**		**0**					
3	**C**		**0**					
4	**G**		**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

ABCG
BDCAG

		j	0	1	2	3	4	5
i			y_j	B	D	C	A	G
0	x_i		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1				
3	C		0					
4	G		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

ABCG
BDCAG

	j	0	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**G**
0	x_i	**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	**G**	**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

ABCG

BDCAG

i	j	0	1	2	3	4	5
		y_j	**B**	**D**	**C**	**A**	**G**
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**1**
3	**C**	**0**					
4	**G**	**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

ABCG
BDCAG

		j	0	1	2	3	4	5
i		y_j		**B**	**D**	**C**	**A**	**G**
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**			
4	**G**		**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

ABCG
BDCAG

i	j	0	1	2	3	4	5
		y_j	**B**	**D**	**C**	**A**	**G**
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	**G**	**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

ABCG

BDCAG

	j	0	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**G**
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**1**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**G**	**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

ABCG
BDCAG

	j	0	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**G**
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**1**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**G**	**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])

ABCG
BDCAG

LCS Example

i	j	0	1	2	3	4	5
	y_j		**B**	**D**	**C**	**A**	**G**
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**1**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**G**	**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

ABCG
BDCAG

i	j	0	1	2	3	4	5
	y_j		**B**	**D**	**C**	**A**	**G**
0	x_i	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**	**1**	**1**
2	**B**	**0**	**1**	**1**	**1**	**1**	**1**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**G**	**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])

Another LCS Example

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

How to Find Actual LCS

So far, we have just found the *length* of LCS, but not LCS itself.

We can modify this algorithm to make it output an LCS of X and Y.

Each $c[i, j]$ depends on $c[i-1, j-1]$, or $c[i-1, j]$ and $c[i, j-1]$.

For each $c[i, j]$ we can say how it was acquired.

2	2
2	3

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

How to Find Actual LCS

Remember that

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

- So we can start from $c[m, n]$ and go backwards
- Whenever $c[i, j] = c[i\text{-}1, j\text{-}1]+1$, remember $x[i]$, because $x[i]$ is a part of LCS
- When i=0 or j=0 (*i.e.* we reached the beginning), output remembered letters in reverse order

Finding LCS: Example

	j	0	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**G**
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**	**2**	**2**
4	**G**	**0**	**1**	**1**	**2**	**2**	**3**

Finding LCS: Example

	j	0	1	2	3	4	5
i		y_j	**B**	**D**	**C**	**A**	**G**
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**	**2**	**2**
4	**G**	**0**	**1**	**1**	**2**	**2**	**3**

LCS (reversed order): **G C B**

LCS (straight order): **B C G**

Finding LCS: Algorithm

```
PRINT-LCS(b, X, i, j)
    if i = 0 or j = 0
        then return
    if b[i, j] = "↖"
        then PRINT-LCS(b, X, i − 1, j − 1)
            print x_i
    elseif b[i, j] = "↑"
        then PRINT-LCS(b, X, i − 1, j)
    else PRINT-LCS(b, X, i, j − 1)
```

Trace backwards from $b[m, n]$

Thank You