

# CSE408 Longest Common Sub Sequence

Lecture # 25

#### Dynamic programming



- It is used, when the solution can be recursively described in terms of solutions to subproblems (*optimal substructure*)
- Algorithm finds solutions to subproblems and stores them in memory for later use
- More efficient than "brute-force methods", which solve the same subproblems over and over again

# Longest Common Subsequence (LC)

Application: comparison of two DNA strings Ex: X= {ABCBDAB}, Y= {BDCABA}

Longest Common Subsequence:

$$X = AB$$
  $C$   $BDAB$ 

$$Y = BDCABA$$

Brute force algorithm would compare each subsequence of X with the symbols in Y

#### LCS Algorithm



- if |X| = m, |Y| = n, then there are  $2^m$  subsequences of x; we must compare each with Y (n comparisons)
- So the running time of the brute-force algorithm is O(n 2<sup>m</sup>)
- Notice that the LCS problem has *optimal substructure*: solutions of subproblems are parts of the final solution.
- Subproblems: "find LCS of pairs of *prefixes* of X and Y"

# 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]

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

# 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

3/16/2020

# 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

3/16/2020

# 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  $LCS(X_{i-1}, Y_i)$

Why not just take the length of LCS( $X_{i-1}$ ,  $Y_{j-1}$ )?

# LCS Length Algorithm



```
LCS-Length(X, Y)
1. m = length(X) // get the # of symbols in X
2. n = length(Y) // get the # of symbols in Y
3. for i = 1 to m c[i,0] = 0 // special case: Y_0
4. for j = 1 to n c[0,j] = 0 // special case: X_0
5. for i = 1 to m
                               // for all X<sub>i</sub>
6. for j = 1 to n
                                      // for all Y<sub>i</sub>
7.
            if (X_i == Y_i)
                   c[i,j] = c[i-1,j-1] + 1
8.
```

9. else c[i,j] = max(c[i-1,j], c[i,j-1])

10. return c

#### LCS Example



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

- X = ABCB
- Y = BDCAB

What is the Longest Common Subsequence of X and Y?

#### LCS Example (0)



	j	0	1 2	3	4 5		CACAS INSUIT
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[5,4]

# LCS Example (1)



j 0 1 2 3 4 !	i	0	1	2	3	4	5
---------------	---	---	---	---	---	---	---

Υj В D

В Α

Xi 

В 

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

# LCS Example (2)



	j	0 1	. 2	3	4 5		CACAS (DED)
i		Yj	B	D	С	Α	В
0	Xi	0	0	0	0	0	0
1	A	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  $c[i,j] = max(c[i-1,j],c[i,j-1])$ 

# LCS Example (3)



В

j 0 1 2 3 4 5	i	0	1	2	3	4	5
---------------	---	---	---	---	---	---	---

i Yj B D C A

1 A 0 0 0 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 (4)



	j	0 1	. 2	3	<b>4</b> 5		CALLAS (DECIN
i		Yj	В	D	С	A	В
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	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  $c[i,j] = max(c[i-1,j],c[i,j-1])$ 

# LCS Example (5)



	j	0 1	L 2	3	4 5		CALLAS IDUIN
i		Yj	В	D	С	Α	$\bigcirc$ B
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1 _	<b>1</b>
2	В	0					
3	С	0					
4	В	0					

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

# LCS Example (6)



	j	0 1	. 2	3	4 5		CACIAS (DUD)
i		Yj	В	D	С	A	В
0	Xi	0	0	0	0	0	0
1	A	0	0	0	0	1	1
2	В	0	1				
3	С	0					
4	В	0					

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

# LCS Example (7)



	j	0 1	2	3	4 5		CAKING TORDS
i		Yj	В	D	С	A	<b>B</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					

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 (8)



	j	0 1	. 2	3	4 5		CACLAS (MICH
i		Yj	В	D	С	Α	(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	С	0					
4	В	0					

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

# LCS Example (10)



	j	0 1	<b>2</b>	3	4 5		CANAS INCO
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			
4	В	0					

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

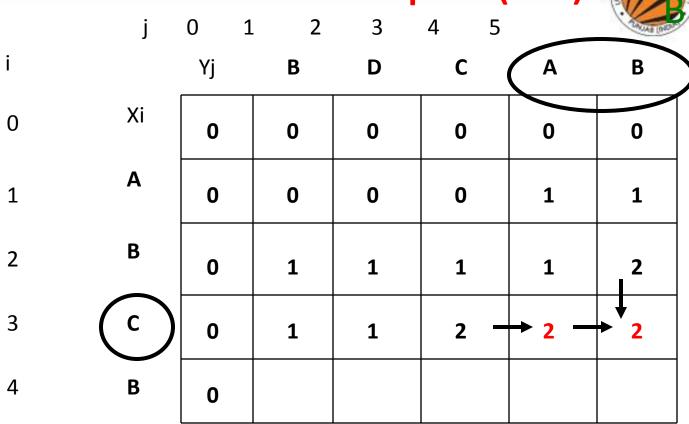
# LCS Example (11)



	j	0	1 2	3	4 5		CACIAS (DED)
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  $c[i,j] = max(c[i-1,j],c[i,j-1])$ 

# LCS Example (12)



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)



	j	0 1	2	3	4 5		CACIAS (DUDA
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	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)



	j	0 1	<b>2</b>	3	<b>4</b> 5		CALLAS INSID
i		Yj	В	D	С	A	<b>S</b> B
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 -	<b>1</b>	2 -	<b>2</b>	

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)



	j	0 2	L 2	3	<b>4 5</b>		SACIAS (DESS)
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	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(m\*n)

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

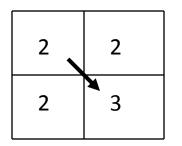
# How to find actual LCS



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

#### How to find actual LCS - continued



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-1, j-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

3/16/2020

# Finding LCS

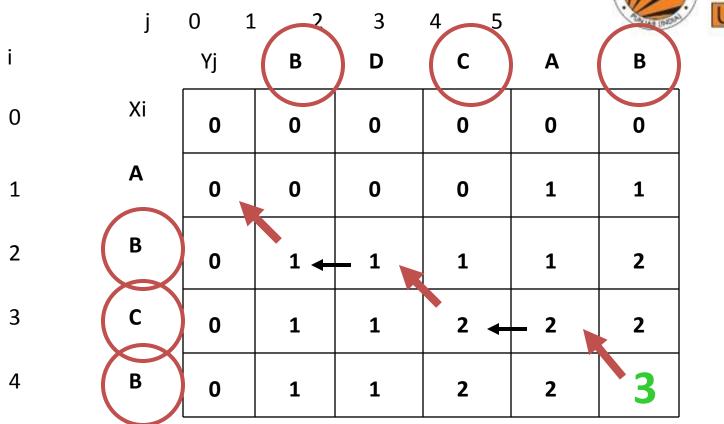




		j	0 2	1 2	3	4 5		CACAS (BESA)
i			Yj	В	D	C	Α	В
(	0	Xi	0	0	0	0	0	0
	1	Α	0	0	0	0	1	1
	2	В	0	1 🛧	- 1	1	1	2
	3	С	0	1	1	2 🛨	_ 2	2
4	4	В	0	1	1	2	2	3

# Finding LCS (2)





LCS (reversed order): B C B

LCS (straight order):

B C B

(this string turned out to be a palindrome)



# Thank You !!!