Design and Analysis of Algorithms

- ***Course Code***: BCSE304L
- ***Course Type***: Theory (ETH)
- ***Slot***: A1+TA1 & & A2+TA2
- ***Class ID***: VL2023240500901
  VL2023240500902

A1+TA1

Day	Start	End
Monday	08:00	08:50
Wednesday	09:00	09:50
Friday	10:00	10:50

A2+TA2

Day	Start	End
Monday	14:00	14:50
Wednesday	15:00	15:50
Friday	16:00	16:50

Syllabus- Module 2

Module:2	Design Paradigms: Dynamic Programming, Backtracking and Branch & Bound Techniques	10 hours

Dynamic programming: Assembly Line Scheduling, Matrix Chain Multiplication, Longest Common Subsequence, 0-1 Knapsack, TSP-
Backtracking: N-Queens problem, Subset Sum, Graph Coloring-
Branch & Bound: LIFO-BB and FIFO BB methods: Job Selection problem, 0-1 Knapsack Problem

Longest Common Subsequence

- The Longest Common Subsequence (LCS) problem involves finding the longest subsequence present in two given sequences while maintaining the same relative order.
- A subsequence is a sequence that appears in the same order, but not necessarily consecutively.

Input:

X = "AGGTAB"

Y = "GATBAYB"

Output:GTAB

Input:

X = "AGGTAB"

Y = "GATBAYB"

Output:GTAB

Longest Common Subsequence

- **Problem:** Given sequences x[1..m] and y[1..n], find a **longest common subsequence** of both.

Example:

x=ABCBDAB and y=BDCABA,

BCA is a common subsequence and

BCBA and BDAB are two LCSs

Substring Vs Subsequence

Substring means considering contiguously.

Example: str: Phanikrishna
- Pha is substring
- Krish is substring

That is substring should be contiguous.

A **subsequence** is a sequence that can be derived from another sequence by zero or more elements, without changing the order of the remaining elements.

K	R	I	S	H	N	A
1	2	3	4	5	6	7

Choose some of the indices, and all those indices are should be in the increasing order and considering those characters or letters, that is nothing but a subsequence

Examples of Subsequence:
- 1356==KIHN
- 267=RNA

213=RKI is not subsequence

If there is string with m characters then how many subsequence are possible?

Total possible subsequence's are 2^m for string of length m.

This includes both non-empty subsequences and the empty sequence.

Application of LCS

- **DNA Sequence Alignment**: One real-time application of the Longest Common Subsequence (LCS) is in the field of genomics, particularly in the study of DNA and genes to determine the similarity between animals or species. A Specie has DNA that is a string or stand of some molecules.

- **Molecular Biology**: In molecular biology, LCS can be used to analyze and compare gene expression patterns. Identifying common subsequences in genes helps understand the regulation of genes and their functions.

- **Identification of Functional Elements**: By comparing DNA sequences, scientists can identify functional elements like genes, promoters, and regulatory regions. LCS aids in locating these elements and understanding their conservation across species.

DNA (Deoxyribonucleic acid)

Actually there are four molecules and these are represented with (A, C, G, T) four characters.

Note: *Adenine (A), Cytosine (C), Guanine (G), and Thymine (T)*

Example: DNA of one specie D1= AGCCTCAGT; DNA of other specie D2= GCCT

- It they are similar both DNA's are equal. Means both from same parent
- If there are very much similar one DNA is substring of other DNA

If the two DNA's are not substring of each other then we would like to know how similar they are.

The similarity depends on what is the longest common subsequence.

Means if **D1 has string of length m** and D2 has string of length n then

- D1 has total 2^m
- D2 has total 2^n

We find the subsequence's which is present in both of them and which is longest. That is called Longest Common sub-sequences.

The DNA's similarity depends on how long the LCS

Brute force approach for Longest Common Subsequence

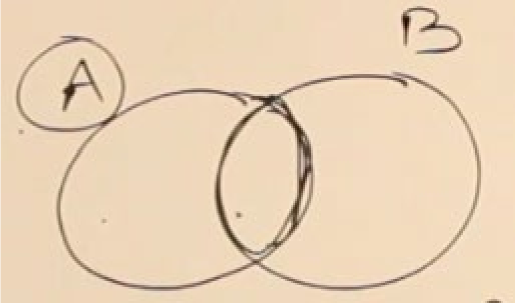
If **A is string of length M**, and **B is string of length N**

- S1: Find all subsequence of A. Its time complexity is 2^M
- S2: Find for each subsequence of A, whether it is a subsequence of B. Its time complexity is N*2^M
  - How:
    - For each subsequence of A, check whether it is a subsequence of B.
    - Time complexity for each check: N
    - Since there are 2^M subsequences of A, the overall time complexity for this step is N×2^M.
- S3: Find all subsequence of B. Its time complexity is 2^N

So final Time complexity is: (2^M)+(N*2^M)+(2^N)==>O(N*2^M) Exponential time.

Simply, If |A| = m, |B| = n, then there are 2^m subsequences of A; we must compare each with B (n comparisons), So the running time of the brute-force algorithm is $O(n\ 2^m)$

If A and B are two string



If A and B are two string

$$C[i][j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + C[i-1][j-1] & \text{if } X[i] = Y[j] \\ \max(C[i-1][j], C[i][j-1]) & \text{if } X[i] \neq Y[j] \end{cases}$$

Recurrence relation of LCS in dynamic programming

Three cases commonly used in dynamic programming to determine the length of the Longest Common Subsequence (LCS) between two strings X and Y.

For finding the LCS between X and Y, where $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_m$, considering $i = 0$ to n and $j = 0$ to m:

- **Case 1**: If $i = 0$ or $j = 0$, it means one of the strings is empty, and the length of the LCS is 0.
- **Case 2**: If x_n is equal to y_m, the last characters of X and Y are the same. In this case, the length of the LCS is one plus the length of the LCS for the previous characters.
  - Symbolically, if $x_n = y_m$: $\mathbf{C[i][j] = 1 + C[i-1][j-1]}$
- **Case 3**: If x_n is not equal to y_m, the last characters of X and Y are different. In this case, the length of the LCS is the maximum length obtained by excluding one character from either string.
  - Symbolically, if $x_n \neq y_m$: $\mathbf{C[i][j] = \max(C[i-1][j], C[i][j-1])}$

This recurrence relation is at the core of the dynamic programming solution for the LCS problem.

X_i and Y_j end with $x_i = y_j$

X_i	$x_1\ x_2\ \dots\ x_{i-1}$	x_i
Y_j	$y_1\ y_2\ \dots\ y_{j-1}$	$y_j = x_i$
Z_k	$z_1\ z_2 \dots z_{k-1}$	$z_k = y_j = x_i$

Z_k is Z_{k-1} followed by $z_k = y_j = x_i$ where
Z_{k-1} is an LCS of X_{i-1} and Y_{j-1} and
$LenLCS(i, j) = LenLCS(i-1, j-1) + 1$

X_i and Y_j end with $x_i \neq y_j$

X_i: $x_1\ x_2\ \ldots\ x_{i-1}\ \ x_i$

Y_j: $y_1\ y_2\ \ldots\ y_{j-1}\ \vert\ y_j$

Z_k: $z_1\ z_2 \ldots z_{k-1}\ \vert\ z_k \neq y_j$

X_i: $x_1\ x_2\ \ldots\ x_{i-1}\ \vert\ x_i$

Y_j: $y_j\ y_1\ y_2\ \ldots y_{j-1}\ \ y_j$

Z_k: $z_1\ z_2 \ldots z_{k-1}\ \vert\ z_k \neq x_i$

Z_k is an LCS of X_i and Y_{j-1}

Z_k is an LCS of X_{i-1} and Y_j

$$LenLCS(i, j) = \max\{LenLCS(i, j-1),\ LenLCS(i-1, j)\}$$

Recurrence relation of LCS in dynamic programming

$$C[i][j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ 1 + C[i-1][j-1] & \text{if } X[i] = Y[j] \\ \max(C[i-1][j], C[i][j-1]) & \text{if } X[i] \neq Y[j] \end{cases}$$

- **Case 1**: If $i = 0$ or $j = 0$, it means one of the strings is empty, and the length of the LCS is 0.
- **Case 2**: If x_n is equal to y_m, the last characters of X and Y are the same. In this case, the length of the LCS is one plus the length of the LCS for the previous characters.
  - Symbolically, if $x_n = y_m$: $\mathbf{C[i][j] = 1 + C[i-1][j-1]}$
- **Case 3**: If x_n is not equal to y_m, the last characters of X and Y are different. In this case, the length of the LCS is the maximum length obtained by excluding one character from either string.
  - Symbolically, if $x_n \neq y_m$: $\mathbf{C[i][j] = \max(C[i-1][j], C[i][j-1])}$

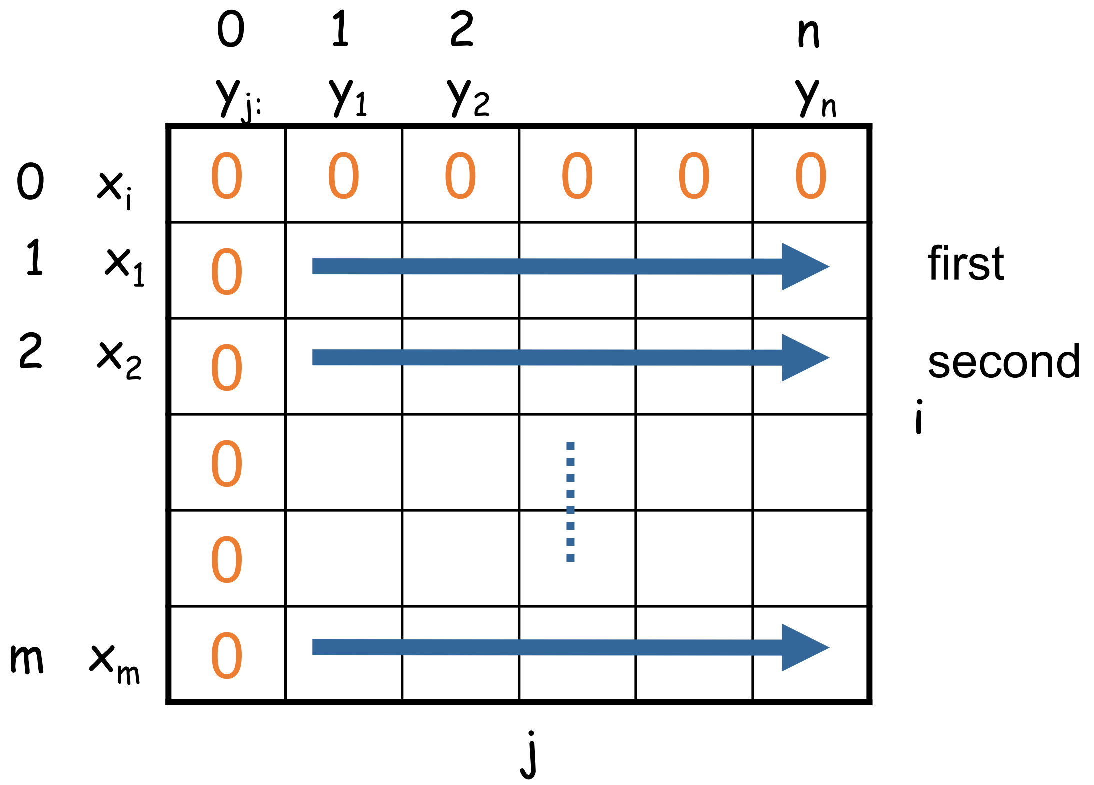
This recurrence relation is at the core of the dynamic programming solution for the LCS problem.

The dynamic programming solution

- Initialize the **first row and the first column of the matrix LenLCS to ‘0’**
- Calculate LenLCS (1, j) **for j = 1,…, n**
- Then the LenLCS (2, j) **for j = 1,…, n, etc.**
- Store also in a table an arrow pointing to the array element that was used in the computation.
- It is easy to see that the computation is **O(mn)**

3. Computing the Length of the LCS

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



Additional Information

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

b & c:

		0	1	2	3		n
		y_j:	A	C	D		F
0	x_i	0	0	0	0	0	0
1	A	0					
2	B	0			c[i-1,j]		
3	C	0		c[i,j-1]			
		0					
m	D	0					

i

j

A matrix b[i, j]:

- For a subproblem [i, j] it tells us what choice was made to obtain the optimal value
- If $x_i = y_j$

  $b[i, j] =$ " $\nwarrow$ "
- Else, if

  $c[i - 1, j] \geq c[i, j-1]$

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

  else

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

Example

	y_j	B	D	C	A
x_j	0	0	0	0	0
A	0	↑ 0	↑ 0	↑ 0	↖ 1
B	0	↖ 1	← 1	← 1	↑ 1
C	0	↑ 1	↑ 1	↖ 2	← 2
B	0	↖ 1	↑ 1	↑ 2	↑ 2

To find an LCS follow the arrows, for each diagonal arrow there is a member of the LCS

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(X, Y) = BCB

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

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

Longest Common Subsequence Procedure

- Let X=<x_1,x_2,x_3....,x_m> and Y=<y_1,y_2,y_3....,y_m> be the sequences. To compute the length of an element the following algorithm is used.
- **Step 1** − Construct an empty adjacency table with the size, n × m, where n = size of sequence **X** and m = size of sequence **Y**. The rows in the table represent the elements in sequence X and columns represent the elements in sequence Y.
- **Step 2** − The zeroeth rows and columns must be filled with zeroes. And the remaining values are filled in based on different cases, by maintaining a counter value.
- **Case 1** − If the counter encounters common element in both X and Y sequences, increment the counter by 1.
- **Case 2** − If the counter does not encounter common elements in X and Y sequences at C[i, j], find the **maximum value between C[i-1, j] and C[i, j-1] to fill it in C[i, j]**.
- **Step3** − Once the table is filled, backtrack from the last value in the table. Backtracking here is done by tracing the path where the counter incremented first.
- **Step4** − The longest common subsequence obtained by noting the elements in the traced path.

LCS Example (0)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi						
1	**A**						
2	**B**						
3	**C**						
4	**B**						

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

LCS Example (1)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**					
2	**B**	**0**					
3	**C**	**0**					
4	**B**	**0**					

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

LCS Example (2)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0				
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 (3)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**A**	**B**
0	Xi	**0**	**0**	**0**	**0**	**0**	**0**
1	**A**	**0**	**0**	**0**	**0**		
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)

ABCB
BDCAB

i		j	0	1	2	3	4	5
			Yj	B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	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 (5)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	A	B
0	Xi	0	0	0	0	0	0
1	A	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 (6)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	A	B
0	Xi		0	0	0	0	0	0
1	A		0	0	0	0	1	1
2	B		0	1				
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 (7)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	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	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 (8)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj		B	D	C	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	2
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 (10)

ABCB
BDCAB

		j	0	1	2	3	4	5
i			Yj	B	D	C	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	2
3	C		0	1	1			
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 (11)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	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	2
3	C	0	1	1	2		
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 (12)

ABCB
BDCAB

j		0	1	2	3	4	5
i		Yj	B	D	C	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	2
3	C	0	1	1	2	2	2
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 (13)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	**B**	**D**	**C**	**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**	**2**
3	**C**	**0**	**1**	**1**	**2**	**2**	**2**
4	**B**	**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)

ABCB
BDCAB

i		j	0	1	2	3	4	5
		Yj	B	D	C	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	2
3	C	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)

ABCB
BDCAB

	j	0	1	2	3	4	5
i		Yj	B	D	C	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	2
3	C	0	1	1	2	2	2
4	B	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-Length(X, Y)

```
m ← length[X}
n ← length[Y]
for i ← 1 to m do
    c[i, 0] ← 0
for j ← 1 to n do
    c[0, j] ← 0
```

```
for i ← 1 to m do
   for j ← 1 to n do
      if x_i = y_j
         c[i, j] ← c[i-1, j-1]+1
         b[i, j] ← "D"
      else
         if c[i-1, j] ≥ c[i, j-1]
            c[i, j] ← c[i-1, j]
            b[i, j] ← "U"
         else
            c[i, j] ← c[i, j-1]
            b[i, j] ← "L"
return c and b
```

Note: 'D' – Diagonal arrow
'U' – UP arrow
'L' – LEFT arrow

Finding LCS (2)

i		j	0	1	2	3	4	5
			Yj	B	D	C	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	2
3	C		0	1	1	2	2	2
4	B		0	1	1	2	2	3

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

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

Running time: $\Theta(m + n)$

```
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)
```

Initial call: PRINT-LCS(b, X, length[X], length[Y])

Example

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

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

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

If $x_i = y_j$

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

Else if

$c[i - 1, j] \geq c[i, j-1]$

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

else

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

		0	1	2	3	4	5	6
	y_j		B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$1	$\leftarrow$1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

4. Constructing a LCS

- Start at b[m, n] and follow the arrows
- When we encounter a "$\nwarrow$" in b[i, j] $\Rightarrow$ $x_i = y_j$ is an element of the LCS

		0	1	2	3	4	5	6
		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

$X = \{A, B, C, B, D, A, B\}$

$Y = \{B, D, C, A, B, A\}$

		Y	B	D	C	A	B	A
0	X	0	0	0	0	0	0	0
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

$X = \{A, B, C, B, D, A, B\}$

$Y = \{B, D, C, A, B, A\}$

		Y	B	D	C	A	B	A
0	X	0	0	0	0	0	0	0
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

$X = \{A, B, C, B, D, A, B\}$

$Y = \{B, D, C, A, B, A\}$

		Y	B	D	C	A	B	A
0	X	0	0	0	0	0	0	0
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

```
Algorithm DP_LCS(X,Y)
{
m=X.length
n=Y.length
Let C[1….m, 1….n] be a new table
For i=1 to m  {    C[i, 0]=0  }
For j=1 to n   {     C[0, j]=0}
For i=1 to m {
     For j=1 to n {
            If xi==yi then do C[i, j]= C[i-1, j-1]+1
            else C[i, j]= max( C[i-1, j], C[i, j-1])
}
}
return C
return C[m,n] i.e., final LCS value.
}
```

Any

Question


Thank You!


FOR YOUR
ATTENTION