ADVANCED ANALYSIS OF ALGORITHMS
CPS 5440

UNIT 10: DYNAMIC PROGRAMMING. LONGEST SUBSEQUENCE PROBLEM.

REVIEW: DYNAMIC PROGRAMMING

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

REVIEW: DYNAMIC PROGRAMMING

- DP is a method for solving certain kinds of problems
- DP can be applied when the solution of a problem includes solutions to subproblems
- We need to find a recursive formula for the solution
- We can recursively solve subproblems,
  Starting from the trivial case, and saving their solutions in memory
- In the end, we'll get the solution to the whole problem

- **Properties of a problem that can be solved with dynamic programming**

- Simple Subproblems

  Break the original problem into smaller subproblems that have the same structure
- Optimal Substructure

  The solution to the problem must be a composition of subproblem solutions
- Subproblems Overlap
  - Optimal subproblems to unrelated problems can contain subproblems in common

General Strategy of Dynamic Programming

1. Structure:
   What's the structure of an optimal solution in terms of solutions to its subproblems?
2. Recursion:
   Give a recursive definition of an optimal solution in terms of optimal solutions to smaller problems
3. Memory:
   Use a data structure (often a table) to store smaller solutions
   The optimal value found in the table
4. Reconstruction:
   Reconstruct the optimal solution
   what produced the optimal value

Application: comparison of two DNA strings

- Example: X =ABCBDAB and Y=BDCABA,
  - BCA is a common subsequence and
  - BCBA and BDAB are two LCSs

- **Solution:** For every subsequence of X, check whether it is a subsequence of Y, and record it if it is longer than the longest previously found.
- $|X| = m, |Y| = n$
- Analysis :
  - There are 2^m subsequences of X to check.
  - For each subsequence, scan Y for the first letter. From there scan for the second letter, etc., up to the n letters of Y.
  - Each check takes $\theta(n)$ time
  - The worst-case running time is $\theta(n * 2^m)$.

exponential time complexity !!!

LCS: BRUTE FORCE ALGORITHM | EXAMPLE

$X =$ ABCB $\quad Y =$ ABDC

- Enumerate all subsequence of X

	Subsequence		Subsequence
0 0 0 0	“”	1 0 0 0	A
0 0 0 1	B	1 0 0 1	AB
0 0 1 0	C	1 0 1 0	AC
0 0 1 1	CB	1 0 1 1	ACB
0 1 0 0	B	1 1 0 0	AB
0 1 0 1	BB	1 1 0 1	ABB
0 1 1 0	BC	1 1 1 0	ABC
0 1 1 1	BCB	1 1 1 1	ABCB

LCS: BRUTE FORCE ALGORITHM | EXAMPLE

$X =$ ABCB $\quad Y =$ ABDC

- Check if each subsequence of X exists in Y

Subsequence	Exists?	Subsequence	Exists?
“”	F	A	T
B	T	AB	T
C	T	AC	T
CB	F	ACB	F
B	T	AB	T
BB	F	ABB	F
BC	T	ABC	T
BCB	F	ABCB	F

LCS: BRUTE FORCE ALGORITHM | EXAMPLE

$X =$ ABCB $\qquad Y =$ ABDC

- Select the existing subsequence with the maximum length (Multiple optimal solutions may exist)

Subsequence	Exists?	Subsequence	Exists?
“”	F	A	T
B	T	AB	T
C	T	AC	T
CB	F	ACB	F
B	T	AB	T
BB	F	ABB	F
BC	T	ABC	T
BCB	F	ABCB	F

LCS WRITING THE RECURRENCE EQUATION

- Let X_i denote the $i - th\ prefix\ x[1..i]$ of $x[1..m]$, and
- X_0 denotes an empty prefix
- We will first compute the *length of an LCS of* X_m *and* Y_n, $LenLCS(m, n)$, and then use information saved during the computation for finding the actual subsequence
- We need a recursive formula for computing $LenLCS(i, j)$.

LCS WRITING THE RECURRENCE EQUATION

- If X_i and Y_j end with the same character $x_i = y_j$,
  the LCS must include the character.
  If it did not,
  we could get a longer LCS by adding the common character.

- If X_i and Y_j do not end with the same character,
  there are two possibilities:
  - Either the LCS does not end with x_i
  - Or it does not end with y_j

- Let Z_k denote an LCS of X_i and Y_j

LCS WRITING THE RECURRENCE EQUATION

- X_i and Y_j end with $x_i = y_j$

$$\boldsymbol{X_i} \quad \boxed{\boldsymbol{x_1 \; x_2 \cdots x_{i-1}}}\boxed{\boldsymbol{x_i}}$$

$$\boldsymbol{Y_j} \quad \boxed{\boldsymbol{y_1 \, y_2 \cdots y_{j-1}}}\boxed{\boldsymbol{y_j}}$$

$$\boldsymbol{Z_k} \quad \boxed{\boldsymbol{z_1 \, z_2 \cdots z_{k-1}}}\boxed{\boldsymbol{z_k = y_j = x_i}}$$

$\boldsymbol{Z_k}$ **is** $\boldsymbol{Z_{k-1}}$ **followed by** $\boldsymbol{z_k = y_j = x_i}$ ***where***

$\boldsymbol{Z_{k-1}}$ **is an LCS of** $\boldsymbol{X_{i-1}}$ **and** $\boldsymbol{Y_{j-1}}$ ***and***

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

- X_i and Y_j end with $x_i = y_j$

$$X_i \quad \boxed{x_1\ x_2 \cdots x_{i-1}}\boxed{x_i} \qquad\qquad X_i \quad \boxed{A\ B\ C}\boxed{D}$$

$$Y_j \quad \boxed{y_1\ y_2 \cdots y_{j-1}}\boxed{y_j} \qquad\qquad Y_j \quad \boxed{A\ C\ B}\boxed{D}$$

$$Z_k \quad \boxed{z_1\ z_2 \cdots z_{k-1}}\boxed{z_k = y_j = x_i} \qquad\qquad Z_k \quad \boxed{A\ C}\boxed{D}$$

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*

$$LenLCS(i,j) = LenLCS(i-1, j-1) + 1$$

LCS WRITING THE RECURRENCE EQUATION

- X_i and Y_j end with $x_i \neq \ y_j$

$$\boldsymbol{X_i} \ \boxed{\boldsymbol{x_1 \ x_2 \cdots x_{i-1} \ x_i}}$$

$$\boldsymbol{Y_j} \ \boxed{\boldsymbol{y_1 \, y_2 \cdots y_{j-1}}}\boxed{\boldsymbol{y_j}}$$

$$\boldsymbol{Z_k} \ \boxed{\boldsymbol{z_1 \, z_2 \cdots z_{k-1}}}\boxed{\boldsymbol{z_k \ \neq y_j}}$$

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

$$\boldsymbol{X_i} \ \boxed{\boldsymbol{x_1 \ x_2 \cdots x_{i-1}}}\boxed{\boldsymbol{x_i}}$$

$$\boldsymbol{Y_j} \ \boxed{\boldsymbol{y_j \, y_1 \cdots y_{j-1} \, y_j}}$$

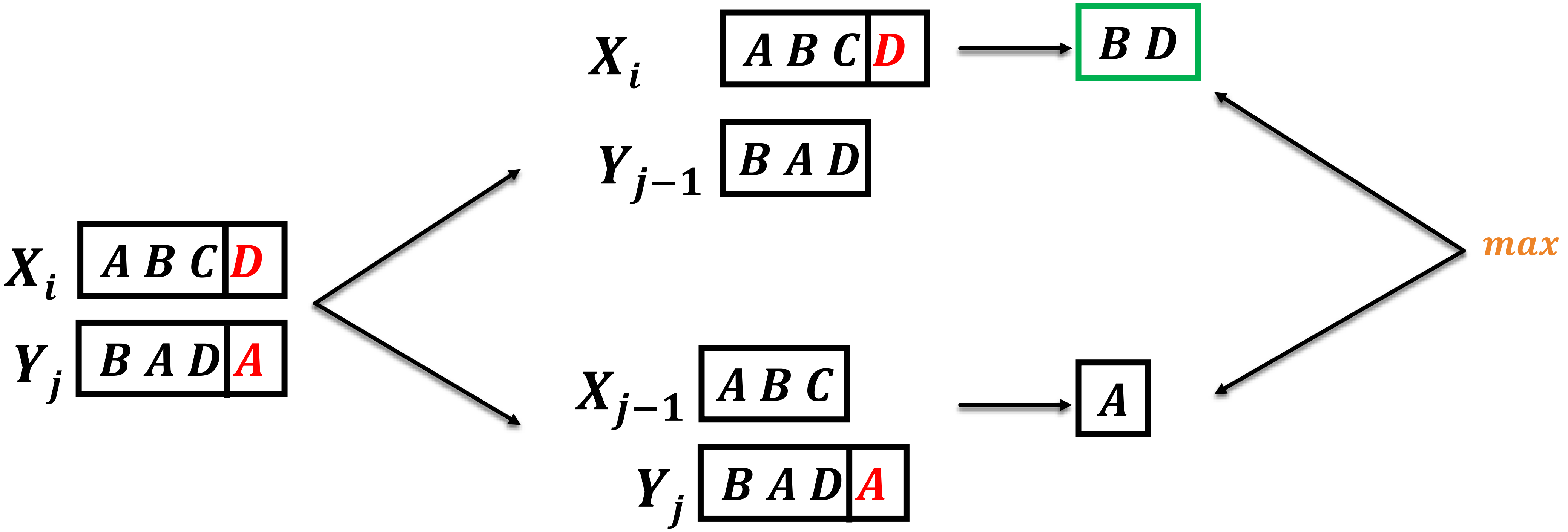
$$\boldsymbol{Z_k} \ \boxed{\boldsymbol{z_1 \, z_2 \cdots z_{k-1}}}\boxed{\boldsymbol{z_k \neq x_i}}$$

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

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

LCS WRITING THE RECURRENCE EQUATION

- X_i and Y_j end with $x_i \neq y_j$

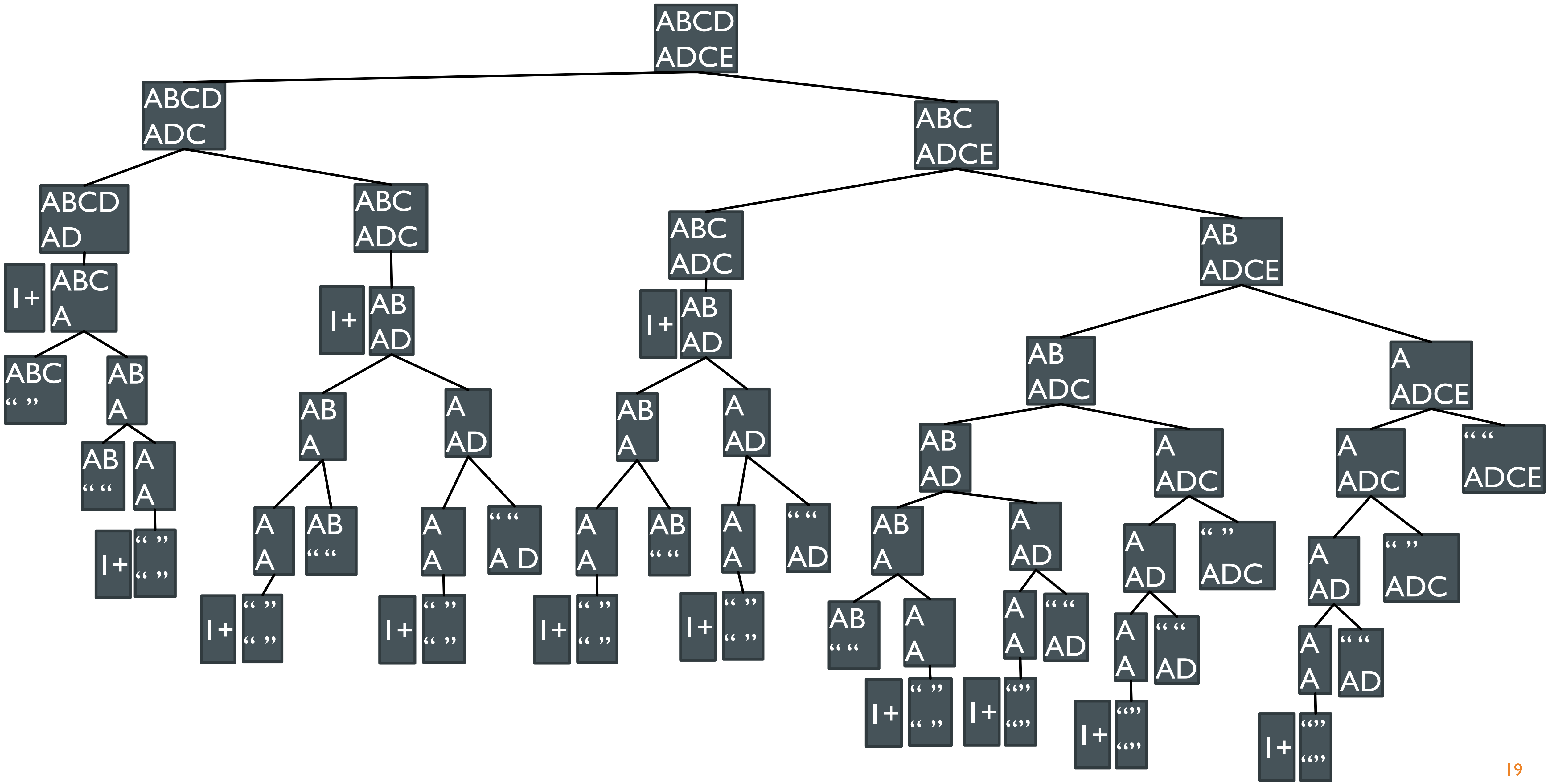


$$LenLCS(i,j) = max\{LenLCS(i,j-1), LenLCS(i-1,j)\}$$

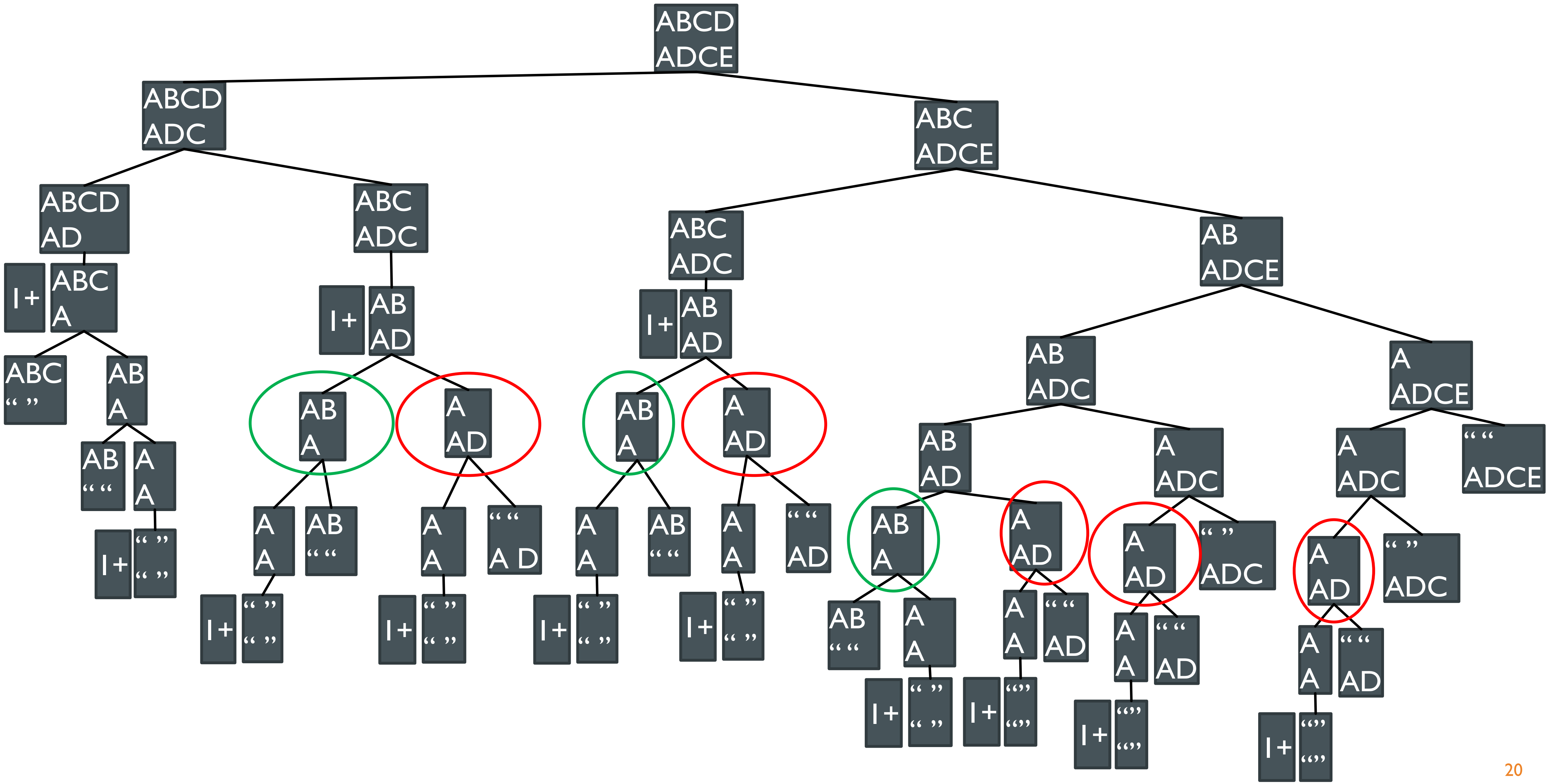
- ***The recurrence equations***

$$lenLCS(i,j) = \begin{cases} 0 & if\ i = 0, or\ j = 0 \\ lenLCS(i-1, j-1) + 1 & if\ i,j > 0\ and\ x_i = y_j \\ \max\{lenLCS(i-1,j), lenLCS(i,j-1)\} & otherwise \end{cases}$$

LCS RECURSIVE ALGORITHMS | EXAMPLE



LCS RECURSIVE ALGORITHMS | EXAMPLE



LCS DYNAMIC PROGRAMMING TOP DOWN

- Store the solutions of subproblems in a look-up table
- Check if the subproblem has already been solved
- If so, retrieve the solution from the table
  - Do not recompute the same subproblem multiple times
- If not, compute the subproblem and update the table

LCS DYNAMIC PROGRAMMING FRAMEWORKS

- Top-Down DP
  - Recursive Algorithm + Explicit memo ($i.e.$, lock-up) table
- Bottom-Up DP
  - Iterative Algorithm + Implicit memo ($i.e.$, lock-up) table

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_i

  6\. for j = 1 to n   // for all Y_j

  7\.   if (X_i == Y_j)

  8\.   c[i,j] = c[i-1,j-1] + 1

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

10\. return c[m,n]   // return LCS length for X and Y

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

LCS EXAMPLE

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

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

LCS EXAMPLE (0)

ABCB
BDCAB

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_i
   6. for j = 1 to n // for all Y_j
   7. if (X_i == Y_j)
   8. c[i,j] = c[i-1,j-1] + 1
   9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

LCS EXAMPLE (1)

ABCB
BDCAB

	j	0	1	2	3	4	5	n
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**						
m								

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

```
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_i
    6. for j = 1 to n               // for all Y_j
    7.     if ( X_i == Y_j )
    8.         c[i,j] = c[i-1,j-1] + 1
    9.     else c[i,j] = max( c[i-1,j], c[i,j-1] )
10. return c[m,n]   // return LCS length for X and Y
```

LCS EXAMPLE

LCS EXAMPLE (2)

ABCB
BDCAB

		j	0	1	2	3	4	5 n
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**					
m								

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)
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_i
   6. for j = 1 to n   // for all Y_j
   7. if ($X_i == Y_j$)
   8. c[i,j] = c[i-1,j-1] + 1
   9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

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**		
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

```
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: Y0
4. for j = 1 to n     c[0,j] = 0   // special case: X0
5. for i = 1 to m                  // for all Xi
   6. for j = 1 to n                 // for all Yj
   7.     if ( Xi == Yj )
   8.         c[i,j] = c[i-1,j-1] + 1
   9.     else c[i,j] = max( c[i-1,j], c[i,j-1] )
10. return c[m,n]   // return LCS length for X and Y
```

LCS EXAMPLE

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 (4)

ABCB
BDCAB

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_i
   6. for j = 1 to n // for all Y_j
   7. if (X_i == Y_j)
   8. c[i,j] = c[i-1,j-1] + 1
   9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

LCS EXAMPLE

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**					
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

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_i
   6. for j = 1 to n   // for all Y_j
   7. if (X_i == Y_j)
   8. c[i,j] = c[i-1,j-1] + 1
   9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n]   // return LCS length for X and Y

LCS EXAMPLE (6)

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

ABCB
BDCAB

```
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_i
    for j = 1 to n              // for all Y_j
        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] )
return c[m,n]   // return LCS length for X and Y
```

LCS EXAMPLE (7)

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**	
3	**C**	**0**					
4	**B**	**0**					

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

ABCB
BDCAB

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_i
 6. for j = 1 to n // for all Y_j
 7. if ($X_i == Y_j$)
 8. c[i,j] = c[i-1,j-1] + 1
 9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

LCS EXAMPLE

	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					
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

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_i
   6. for j = 1 to n // for all Y_j
   7. if ($X_i == Y_j$)
   8. c[i,j] = c[i-1,j-1] + 1
   9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

LCS EXAMPLE

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			
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

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_i
    6. for j = 1 to n // for all Y_j
    7. if (X_i == Y_j)
    8. c[i,j] = c[i-1,j-1] + 1
    9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

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**		
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

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_i
    6. for j = 1 to n // for all Y_j
    7. if ($X_i == Y_j$)
    8. c[i,j] = c[i-1,j-1] + 1
    9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

LCS EXAMPLE

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**					

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

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_i
    6. for j = 1 to n   // for all Y_j
    7. if (X_i == Y_j)
    8. c[i,j] = c[i-1,j-1] + 1
    9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n]   // return LCS length for X and Y

LCS EXAMPLE (13)

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**				

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)
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_i
    6. for j = 1 to n   // for all Y_j
    7. if ($X_i == Y_j$)
    8.   c[i,j] = c[i-1,j-1] + 1
    9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

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 (14)

ABCB
BDCAB

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_i
 6. for j = 1 to n // for all Y_j
 7. if (X_i == Y_j)
 8. c[i,j] = c[i-1,j-1] + 1
 9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

LCS EXAMPLE

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

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])

ABCB
BDCAB

LCS EXAMPLE (15)

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_i
    6. for j = 1 to n   // for all Y_j
    7. if (X_i == Y_j)
    8. c[i,j] = c[i-1,j-1] + 1
    9. else c[i,j] = max(c[i-1,j], c[i,j-1])
10. return c[m,n] // return LCS length for X and Y

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

- 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:

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
- Look first to see if 2nd case above was true
- If not, then *c[i,j] = c[i-1, j-1]+1*, so 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

- Here's a recursive algorithm to do this:

```
LCS_print(x, m, n, c) {
   if (c[m][n] == c[m-1][n]) // go up?
      LCS_print(x, m-1, n, c);
   else if (c[m][n] == c[m][n-1] // go left?
      LCS_print(x, m, n-1, c);
   else { // it was a match!
      LCS_print(x, m-1, n-1, c);
      print(x[m]); // print after recursive call
   }
}
```

FINDING LCS

	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**

FINDING LCS

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**

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

(this string turned out to be a palindrome ☺)

Longest Common *Subsequence* $\Rightarrow$ Longest Common <u>Substring</u>

LONGEST COMMON SUBSTRING

- Given two strings, the task is to find the **Longest Common Substring** present in the given strings in the same order.
- The substring is a **contiguous** sequence of characters within a string.
- For example, “bit” is a substring of the string “Interviewbit”.

Example:
Input s1: “dadef” **s2:** “adwce”
Output: 2
Explanation: Substring “ad” of length 2 is the longest.

Input s1: “abcdxyz”
s2: “xyzabcd”
Output: 4
Explanation: Substring “abcd” of length 4 is the longest.

- ***The recurrence equations***

$$lenLCS(i,j) = \begin{cases} 0 & if\ i = 0, or\ j = 0 \\ lenLCS(i-1, j-1) + 1 & if\ i,j > 0\ and\ x_i = y_j \\ \cancel{\max\{lenLCS(i-1,j), lenLCS(i,j-1)\}}\ 0 & otherwise \end{cases}$$

LCS LENGTH ALGORITHM | BOTTOM UP

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_i
    6. for j = 1 to n // for all Y_j
    7. if (X_i == Y_j)
    8. c[i,j] = c[i-1,j-1] + 1
    9. else ~~c[i,j] = max(c[i-1,j], c[i,j-1])~~ 0
10. return c[m,n] // return LCS length for X and Y

LCS LENGTH ALGORITHM | BOTTOM UP

Consider the below example –
str1 = “ABCXYZAY”
str2 =” “XYZABCB”

The longest common substring is **“XYZA”,** which is of length 4.

		A	B	C	X	Y	Z	A	Y
	0	0	0	0	0	0	0	0	0
X	0	0	0	0	1	0	0	0	0
Y	0	0	0	0	0	2	0	0	1
Z	0	0	0	0	0	0	3	0	0
A	0	1	0	0	0	0	0	4	0
B	0	0	2	0	0	0	0	0	0
C	0	0	0	3	0	0	0	0	0
B	0	0	1	0	0	0	0	0	0

InterviewBit

IMPLEMENTATION

1. Longest Common Subsequence
2. Longest Common Substring

QUESTIONS/ANSWERS

LCS | OPTIMAL SUBSTRUCTURE PROOF OF LCS

- Often when solving a problem, we start with what is known and then figure out how to construct a solution.
- The optimal substructure analysis takes the reverse strategy: assume you have found an optional solution (Z below) and figure out what you must have done to get it!

Notation:

- $X_i = prefix \langle x_1, \dots, xi \rangle$
- $Y_i = prefix \langle y_1, \dots, yi \rangle$

Theorem:

Let $Z = \langle z_1, \dots, z_k \rangle$ be any LCS of $X = \langle x_1, \dots, x_m \rangle$ and $Y = \langle y_1, \dots, y_n \rangle$. Then

1. If $x_m = y_n$, then $z_k = x_m = y_n$, and Z_{k_1} is an LCS of X_{m_1} and Y_{n_1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m \Rightarrow Z$ is an LCS of X_{m_1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n \Rightarrow Z$ is an LCS of X and Y_{n_1}.

- ***Sketch of proofs:***

- (1) can be proven by contradiction:
  If $z_k \neq x_m$, then we could append $x_m = y_n$ to Z to obtain a common subsequence of X and Y of length $k + 1$, contradicting the supposition that Z is a longest common subsequence of X and Y .
  Thus, we must have $z_k = x_m = y_n$.

  Now, the prefix Z_{k-1} is a length-$(k-1)$ common subsequence of X_{m-1} and Y_{n-1}.
  We wish to show that it is an LCS.
  Suppose for the purpose of contradiction that there is a common subsequence W of X_{m-1} and Y_{n-1} with length greater than $k - 1$. Then, appending $x_m = y_n$ to W produces a common subsequence of X and Y whose length is greater than k, which is a contradiction.

- ***Sketch of proofs (cont.):***

- (2) and (3) have symmetric proofs:
  Suppose there exists a subsequence W of X_{m-1} and Y $(or\ of\ X\ and\ Yn_{-1})$ with length $> k$.
  Then W is a common subsequence of X and Y, contradicting Z being an LCS.

- Therefore, **an LCS of two sequences contains as prefix an LCS of prefixes of the sequences.** We can now use this fact construct a recursive formula for the value of an LCS.