

# CSC 226

Algorithms and Data Structures: II

Rich Little

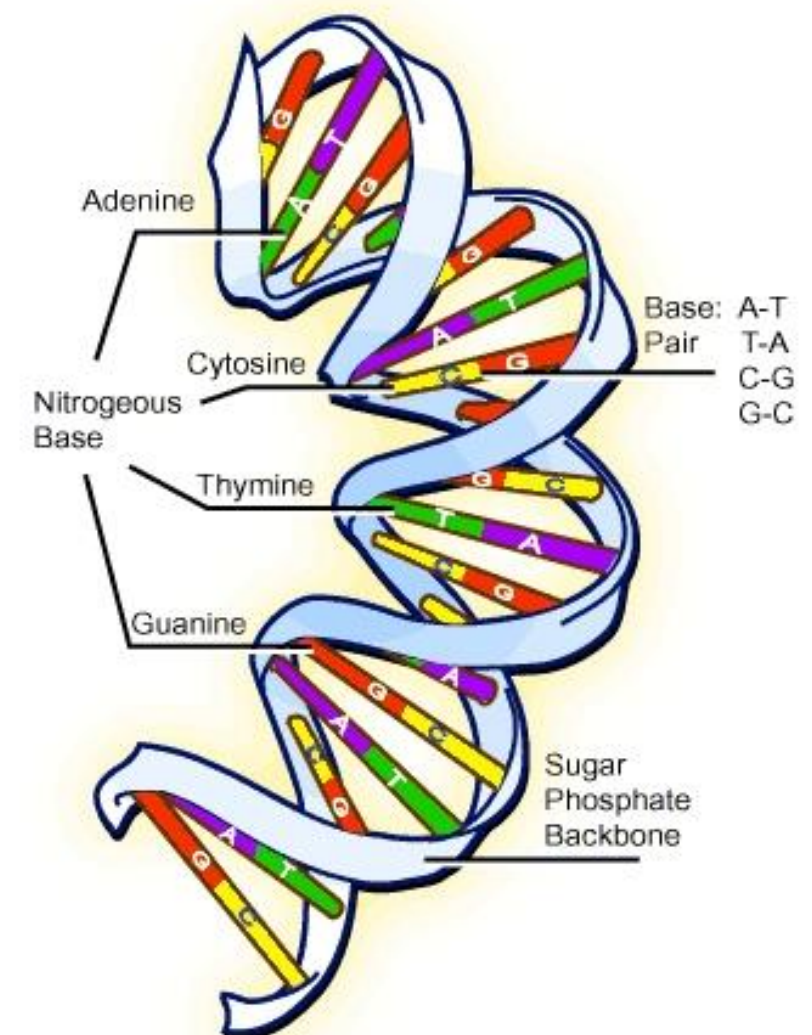
[rlittle@uvic.ca](mailto:rlittle@uvic.ca)

ECS 516

# Finding a Longest Common Subsequence

	Aligning									
	7	260	*	280	*	300	*	320		
species 1	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCCCATTAAAG	GTGA	AACC3C3AATGGCTCAATAAA	TCA				
species 2	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAATCCTCTTGA	GG	GAGA	AAC3C3AAGGCTCAATAAA	TCA			
species 3	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCTCTCTAAAG	GTGA	AACC3C3AATGGCTCAATAAA	TCA				
species 4	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAGNCCG	ATCT	AAG	GCGA	AACC3C3AATGGCTCAATAAA	TCA		
species 5	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAGGCCG	ATCT	AAG	GCGA	AACC3C3AATGGCTCAATAAA	TCA		
species 6	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAGGCCG	AAC	AAG	GCGA	AACC3C3AATGGCTCAATAAA	TCA		
species 7	TCGTTGTCTCGTTT	CTTGC	TGTCTAAAGT	ACAAGCCG	ATTC	AAG	GCGA	AACC3C3AATGGCTCAATAAA	TCA	
species 8	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCG	ATTT	AAG	GCGA	AACC3C3AATGGCTCAATAAA	TCA		
species 9	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCG	ATGT	AAG	GTGA	AACC3C3AATGGCTCAATAAA	TCA		
species 10	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACA---	CCTCTG	GG	GCGA	AACC3C3AATGGCTCAATAAA	TCA		
species 11	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCGCTATG	CG	GCGA	AACC3C3AATGGCTCAATAAA	TCA			
species 12	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCGCTAGA	CG	GCGA	AACC3C3AATGGCTCAATAAA	TCA			
species 13	TCAAAGATTAAAGC	CATGCAGGTCTAAAGT	ATAAGCCGAAATA	AA	GTGA	GACC3C3AATGGCTCAATACA	TCA			
species 14	TCAAAGATTAAAGC	CATGCAGGTCTAAAGT	ATGAGCCGAAATA	AAT	GTGA	GACC3C3AATGGCTCAATACA	TCA			
species 15	TCAAAGATTAAAGC	CATGCAGGTCTAAAGT	ACATGCTCTTATA	TATG	GTA	GACT3C3AATGGCTCAATACA	TCA			
species 16	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACACACCAAAATTA	AG	GTGA	AACC3C3AATGGCTCAATAAA	TCA			
species 17	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCCTACAA	GG	CTGA	AACC3C3AATGGCTCAATAAA	TCA			
species 18	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCGCTATTA	AG	GCGA	AACC3C3AATGGCTCAATAAA	TCA			
species 19	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCGCTATTA	AG	GTGA	AACC3C3AATGGCTCAATAAA	TCA			
species 20	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCCTTACATA	CG	GTGA	AACC3C3AATGGCTCAATAAA	TCA			
species 21	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	TCA	AGCTCGTCT	CG	CGGACA	AAC3C3GATGGCTCAATAAA	TCA		
species 22	TCAAAGATTAAAGC	CATGCATGTCTAAAGT	ACAAGCCCTTACTN	AG	GTGA	AACC3C3AATGGCTCAATAAA	TCA			
species 23	TCAAAGATTAAAGC	CAACTCATGTCTAAAGT	TCATGCCGAAACCAAG	GCGA	AACC3C3AATGGCTCAATAAA	TCA				

(Andy Vierstraete 1999)



# Applications

- Finding similarities between DNA sequences.
  - Test for genetic relationships between organisms.
  - Ex:
    - $X = \text{ACCGGTCGAGTGCGCGGAAGCCGGCCGAA}$
    - $Y = \text{GTCGTTCGGAATGCCGTTGCTCTGTAA}$
    - $\text{LCS} = \text{GTCGTCGGAAGCCGGCCGAA}$
- Determining change from one version of source code to another.
- Unix “diff” command for comparing files.
- Distinguish between similar web pages in web crawlers.

# Terminology

- A **string**  $c = c_1c_2 \cdots c_n$  is a sequence of characters (or symbols from an alphabet)
- A **substring**  $s = s_1s_2 \cdots s_m$  of a string  $c = c_1c_2 \cdots c_n$  is a string with

$$s_1 = c_i, s_2 = c_{i+1}, \dots, s_m = c_{i+m-1}$$

- A **subsequence**  $x = x_1x_2 \cdots x_r$  of a string  $c = c_1c_2 \cdots c_n$  is a string with

$$x_1 = c_{i_1}, x_2 = c_{i_2}, \dots, x_r = c_{i_r}$$

where

$$1 \leq i_1 < i_2 < \cdots < i_r \leq n$$

# Notes

- **Note 1:** The characters of a subsequence of a string are not necessarily consecutive in the string.
- **Note 2:** The characters of a subsequence of a string do preserve the order they have in the string.
- **Note 3:** A substring is also a subsequence, but the reverse is not necessarily true!
- **Note 4:** The *length* of a string is the number of characters in the string.

# Examples

- The string lamp is neither a substring nor a subsequence of the string examples.
- The string ample is a substring of the string examples
- amps is a subsequence but not a substring of the string examples

# More terminology

- Given two strings  $s = s_1s_2 \cdots s_n$  and  $t = t_1t_2 \cdots t_m$ , a ***common subsequence*** of  $s$  and  $t$  is a string that is both a subsequence of  $s$  and a subsequence of  $t$ .
- A ***longest common subsequence (lcs)*** of two strings  $s$  and  $t$  is a common subsequence of maximum length.

# Longest Common Subsequence Problem (LCS)

- Input: Strings  $x = x_1x_2 \cdots x_n$  and  $y = y_1y_2 \cdots y_m$
- Output: string  $lcs$ , where  $lcs$  is a longest common subsequence of  $x$  and  $y$



# Computing an lcs

- Idea:
  - For the shorter of the two given strings create all possible subsequences
  - From the longest to shortest: check if it is also a subsequence of the longer string; output the first one found
- Running time very high

# Dynamic programming

- Elements of dynamic programming
  - optimal substructure
  - overlapping subproblems

# The three steps of Dynamic Programming

- Characterize the structure of an optimal solution
- Recursively define the value of an optimal solution
- Compute the value of an optimal solution in a bottom-up fashion (e.g., via table)

c h i m p a n z e e

h

u

m

a

n

# Theorem (Optimal Substructure of lcs)

- Let  $x = x_1x_2 \cdots x_n$  and  $y = y_1y_2 \cdots y_m$  be strings, and let  $z = z_1z_2 \cdots z_k$  be an lcs of  $x$  and  $y$ . Then,
  1. If  $x_n = y_m$ , then  $z_k = x_n = y_m$  and  $z_1z_2 \cdots z_{k-1}$  is an lcs of  $x_1x_2 \cdots x_{n-1}$  and  $y_1y_2 \cdots y_{m-1}$ .
  2. If  $x_n \neq y_m$  and  $z_k \neq x_n$ , then  $z$  is an lcs of  $x_1x_2 \cdots x_{n-1}$  and  $y$ .
  3. If  $x_n \neq y_m$  and  $z_k \neq y_m$ , then  $z$  is an lcs of  $x$  and  $y_1y_2 \cdots y_{m-1}$ .

**Proof of 1. If  $x_n = y_m$  then (a)  $z_k = x_n = y_m$  and (b)  $z_1 z_2 \dots z_{k-1}$  is an lcs of  $x_1 x_2 \dots x_{n-1}$  and  $y_1 y_2 \dots y_{m-1}$**

$x_n = y_m$ . We prove claim (a) using contradiction: Assume  $z_k \neq x_n$ .

But then  $z$  could be made longer by appending  $x_n = y_m$  to  $z$ . But according to the assumptions of the Theorem,  $z$  is an lcs of  $x$  and  $y$ . Therefore, no longer common subsequence of  $x$  and  $y$ , including  $z x_n$ , can exist, a contradiction.

This proves (a).

**Proof of 1. If  $x_n = y_m$  then (a)  $z_k = x_n = y_m$  and (b)  $z_1 z_2 \dots z_{k-1}$  is an lcs of  $x_1 x_2 \dots x_{n-1}$  and  $y_1 y_2 \dots y_{m-1}$**

$x_n = y_m$  and  $z_k = x_n = y_m$ . We prove claim (b) contradiction:  
Assume  $z_1 z_2 \dots z_{k-1}$  is not an lcs of  $x_1 x_2 \dots x_{n-1}$  and  $y_1 y_2 \dots y_{m-1}$

But then there must exist a string  $w$  that is of length greater than  $k-1$  and a common subsequence of  $x_1 x_2 \dots x_{n-1}$  and  $y_1 y_2 \dots y_{m-1}$ .

Appending  $x_n = y_m$  to  $w$  creates a common subsequence of  $x$  and  $y$ . But  $w$  is longer than  $z$ , a contradiction.

This proves (b).

Proof of 2. If  $x_n \neq y_m$  then  $z_k \neq x_n$  implies that  $z$  is an lcs of  $x_1 x_2 \dots x_{n-1}$  and  $y$

- Assume  $x_n \neq y_m$  and  $z_k \neq x_n$ . Then  $z$  is common subsequence of  $x_1 x_2 \dots x_{n-1}$  and  $y$  (since otherwise  $z$  would not be a subsequence of  $x$  and  $y$ ).
- Assume that  $z$  is not a longest common subsequence. Then there exists a longer common subsequence,  $w$ , of  $x_1 x_2 \dots x_{n-1}$  and  $y$ . But then  $w$  is of length greater than  $k$ , which does not exist according to the assumptions of the Theorem. A contradiction. This proves that  $z$  is an lcs of  $x_1 x_2 \dots x_{n-1}$  and  $y$  and therefore (2)



# Proof of (3)

- Works analogous to the proof of (2)

# What can we conclude from the Theorem?

- Let  $x = x_1 x_2 \dots x_n$  and  $y = y_1 y_2 \dots y_m$  be strings, and let  $z = z_1 z_2 \dots z_k$  be an lcs of  $x$  and  $y$ . Let  $\text{llcs}$  denote the length of an lcs. Then

- If  $x_n = y_m$  then

$$\text{llcs}(x, y) = \text{llcs}(x_1 x_2 \dots x_{n-1}, y_1 y_2 \dots y_{m-1}) + 1$$

- If  $x_n \neq y_m$  then

$$\text{llcs}(x, y) = \max\{\text{llcs}(x_1 x_2 \dots x_{n-1}, y), \text{llcs}(x, y_1 y_2 \dots y_{m-1})\}$$

c h i m				
h	<div>This cell should contain the length of a longest common subsequence for strings chimpanzee and human</div>			
u				
m				
a				
n				



















































		c	h	i	m	p	a	n	z	e	e
h u m a n		0	0	0	0	0	0	0	0	0	0
	h	0	0	1	1	1	1	1	1	1	1
	u	0	0	1	1	1	1	1	1	1	1
	m	0	0	1	1	2	2	2	2	2	2
	a	0	0	1	1	2	2	3	3	3	3
	n	0	0								

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	0	1	1	1	1	1	1	1	1	1
u	0	0	1	1	1	1	1	1	1	1	1
m	0	0	1	1	2	2	2	2	2	2	2
a	0	0	1	1	2	2	3	3	3	3	3
n	0	0	1								

	c	h	i	m	p	a	n	z	e	e
	0	0	0	0	0	0	0	0	0	0
h	0	0	1	1	1	1	1	1	1	1
u	0	0	1	1	1	1	1	1	1	1
m	0	0	1	1	2	2	2	2	2	2
a	0	0	1	1	2	2	3	3	3	3
n	0	0	1	1	2	2	3			

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	0	1	1	1	1	1	1	1	1	1
u	0	0	1	1	1	1	1	1	1	1	1
m	0	0	1	1	2	2	2	2	2	2	2
a	0	0	1	1	2	2	3	3	3	3	3
n	0	0	1	1	2	2	3	4			

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h		0	0	1	1	1	1	1	1	1	1
u		0	0	1	1	1	1	1	1	1	1
m		0	0	1	1	2	2	2	2	2	2
a		0	0	1	1	2	2	3	3	3	3
n		0	0	1	1	2	2	3	4	4	

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	0	1	1	1	1	1	1	1	1	1
u	0	0	1	1	1	1	1	1	1	1	1
m	0	0	1	1	2	2	2	2	2	2	2
a	0	0	1	1	2	2	3	3	3	3	3
n	0	0	1	1	2	2	3	4	4	4	4

# We know the length lcs, but not an lcs

- Each time when computing the content of a new cell, remember where you came from!









































If  $x[i] \neq y[j]$ :  $llcs[i,j] = \max\{llcs[i,j-1], llcs[i-1,j]\}$

[illegible]



If  $x[i] = y[j]$ :  $llcs[i,j] = llcs[i-1,j-1]+1$

[illegible]

If  $x[i] \neq y[j]$ :  $llcs[i,j] = \max\{llcs[i,j-1], llcs[i-1,j]\}$

[illegible]

If  $x[i] \neq y[j]$ :  $llcs[i,j] = \max\{llcs[i,j-1], llcs[i-1,j]\}$

		c	h	i	m	p	a	n	z	e	e
h u m a n		0	0	0	0	0	0	0	0	0	0
	h	0 ← 0	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1
	u	0 ← 0	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1	1 ← 1
	m	0 ← 0	1 ← 1	1 ← 2	2 ← 2	2 ← 2	2 ← 2	2 ← 2	2 ← 2	2 ← 2	2 ← 2
	a	0 ← 0	1 ← 1	1 ← 2	2 ← 2	2 ← 3	3 ← 3	3 ← 3	3 ← 3	3 ← 3	3 ← 3
n	0										



If  $x[i] \neq y[j]$ :  $llcs[i,j] = \max\{llcs[i,j-1], llcs[i-1,j]\}$

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h		0 ← 0	1 ← 1 ← 1 ← 1 ← 1 ← 1 ← 1 ← 1 ← 1								
u		0 ← 0	1 ← 1 ← 1 ← 1 ← 1 ← 1 ← 1 ← 1 ← 1								
m		0 ← 0	1 ← 1	2 ← 2 ← 2 ← 2 ← 2 ← 2 ← 2							
a		0 ← 0	1 ← 1	2 ← 2	3 ← 3 ← 3 ← 3 ← 3						
n		0 ← 0	1								



If  $x[i] \neq y[j]$ :  $llcs[i,j] = \max\{llcs[i,j-1], llcs[i-1,j]\}$

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3				

If  $x[i] = y[j]$ :  $llcs[i,j] = llcs[i-1,j-1] + 1$

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4			

If  $x[i] = y[j]$ :  $llcs[i,j] = llcs[i-1,j-1] + 1$

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4		

If  $x[i] = y[j]$ :  $llcs[i,j] = llcs[i-1,j-1] + 1$

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

Step 4: Extracting the path to  
obtain the lcd

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4



		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
n	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	a	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
a	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0	← 0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	<b>4</b>	← 4	← 4	← 4



		c	h	i	m	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	m	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
m	0	← 0	1	← 1	<b>2</b>	← <b>2</b>	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0	← 0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	<b>4</b>	← <b>4</b>	← <b>4</b>	← <b>4</b>

		c	h	i	<u>m</u>	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0	← 0	1	← 1	<b>2</b>	← <b>2</b>	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0	← 0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	<b>4</b>	← <b>4</b>	← <b>4</b>	← <b>4</b>

		c	h	i	<u>m</u>	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0	← 0	1	← 1	<b>2</b>	← <b>2</b>	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0	← 0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	<b>4</b>	← <b>4</b>	← <b>4</b>	← <b>4</b>

		c	h	i	<u>m</u>	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0	← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0	← 0	1	← 1	2	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0	← 0	1	← 1	2	← 2	3	← 3	← 3	← 3	← 3
<u>n</u>	0	← 0	1	← 1	2	← 2	3	4	← 4	← 4	← 4

		c	h	i	<u>m</u>	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0 ← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0 ← 0	<b>1</b>	← <b>1</b>	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0 ← 0	1	← 1	<b>2</b>	← <b>2</b>	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0 ← 0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3	← 3
<u>n</u>	0 ← 0	1	← 1	2	← 2	3	<b>4</b>	← <b>4</b>	← <b>4</b>	← <b>4</b>	← <b>4</b>

		c	h	i	<u>m</u>	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0 ← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0 ← 0	<b>1</b>	← <b>1</b>	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0 ← 0	1	← 1	<b>2</b>	← <b>2</b>	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0 ← 0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3	← 3
<u>n</u>	0 ← 0	1	← 1	2	← 2	3	<b>4</b>	← <b>4</b>	← <b>4</b>	← <b>4</b>	← <b>4</b>



		c	h	i	<u>m</u>	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0 ← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0 ← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0 ← 0	1	← 1	<b>2</b>	← 2	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0 ← 0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3	← 3
<u>n</u>	0 ← 0	1	← 1	2	← 2	3	<b>4</b>	← 4	← 4	← 4	← 4

		c	<u>h</u>	i	m	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
<u>h</u>	0 ← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0 ← 0	1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0 ← 0	1	← 1	<b>2</b> ← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0 ← 0	1	← 1	2 ← 2	<b>3</b> ← 3	← 3	← 3	← 3	← 3	← 3	← 3
<u>n</u>	0 ← 0	1	← 1	2 ← 2	3	<b>4</b> ← 4	← 4	← 4	← 4	← 4	← 4

		c	<u>h</u>	i	<u>m</u>	p	<u>a</u>	<u>n</u>	z	e	e
		0	0	0	0	0	0	0	0	0	0
<u>h</u>	0 ← 0	0	<b>1</b>	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
u	0 ← 0	0	<b>1</b>	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
<u>m</u>	0 ← 0	0	1	← 1	<b>2</b>	← 2	← 2	← 2	← 2	← 2	← 2
<u>a</u>	0 ← 0	0	1	← 1	2	← 2	<b>3</b>	← 3	← 3	← 3	← 3
<u>n</u>	0 ← 0	0	1	← 1	2	← 2	3	<b>4</b>	← 4	← 4	← 4

# Algorithm LCS(string $x$ , string $y$ )

```
 $n \leftarrow x.length()$ 
 $m \leftarrow y.length()$ 
for  $i$  from 1 to  $n$  do  $llcs[i,0] \leftarrow 0$ 
for  $j$  from 1 to  $m$  do  $llcs[0,j] \leftarrow 0$ 
for  $i$  from 1 to  $n$  do
  for  $j$  from 1 to  $m$  do
    if  $x[i] = y[j]$  then
       $llcs[i,j] \leftarrow llcs[i-1,j-1] + 1$ ;  $path[i,j] \leftarrow \text{“}\nwarrow\text{”}$ 
    else if  $llcs[i-1,j] > llcs[i,j-1]$  then
       $llcs[i,j] \leftarrow llcs[i-1,j]$ ;  $path[i,j] \leftarrow \text{“}\uparrow\text{”}$ 
    else  $llcs[i,j] \leftarrow llcs[i,j-1]$ ;  $path[i,j] \leftarrow \text{“}\leftarrow\text{”}$ 
return  $llcs$  &  $path$ 
```

# Algorithm Print-LCS(matrix path, string $x$ , positive integers $i, j$ )

**if**  $i = 0$  or  $j = 0$  **then return**

**if**  $\text{path}[i, j] = \nwarrow$  **then**

    print-LCS(path,  $x$ ,  $i-1, j-1$ )

    print  $x[i]$

**else if**  $\text{path}[i, j] = \uparrow$  **then**

    print-LCS(path,  $x$ ,  $i-1, j$ )

**else** print-LCS(path,  $x$ ,  $i, j-1$ )

# Running times

- LCS:  $O(nm)$
- print-LCS:  $O(n+m)$