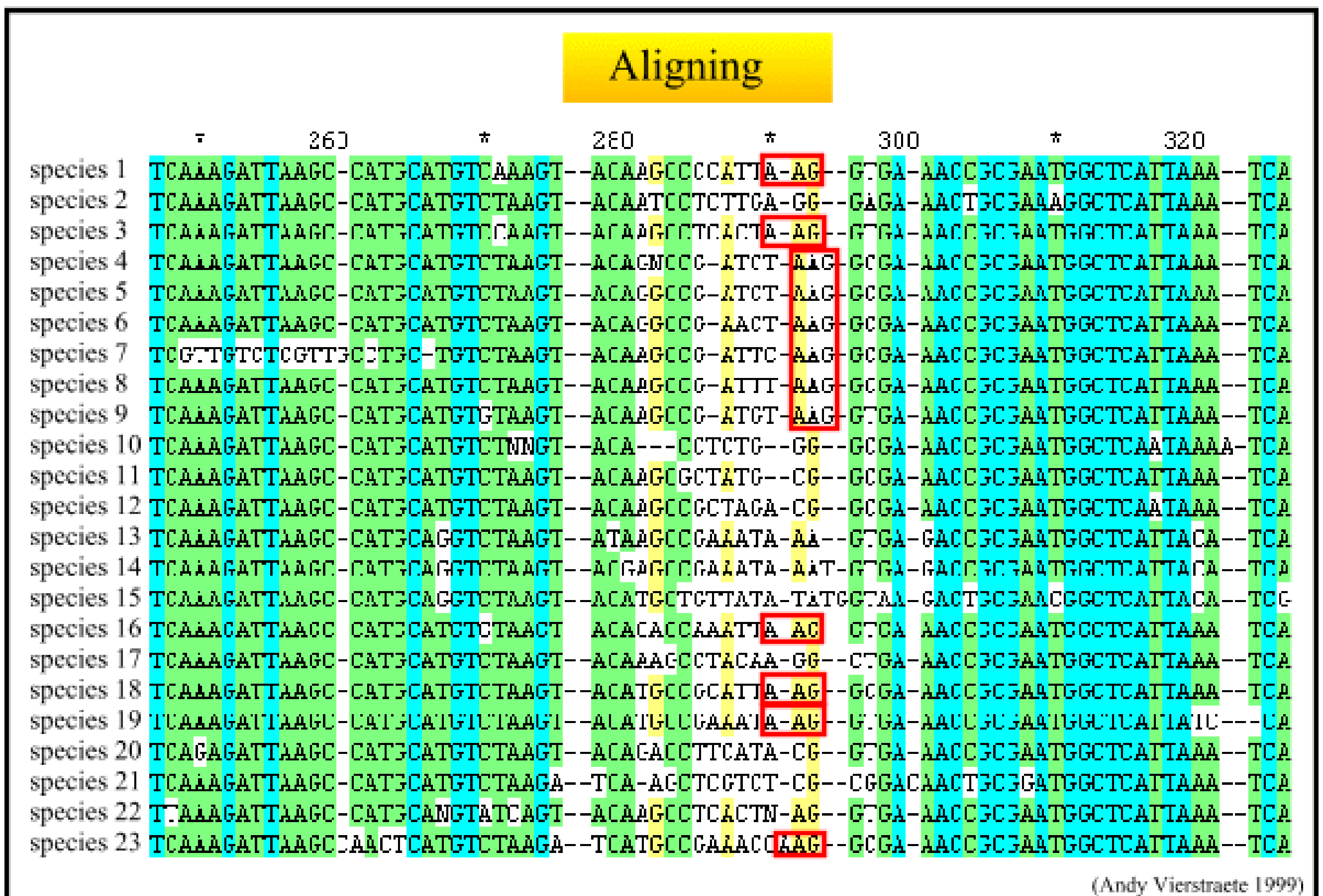
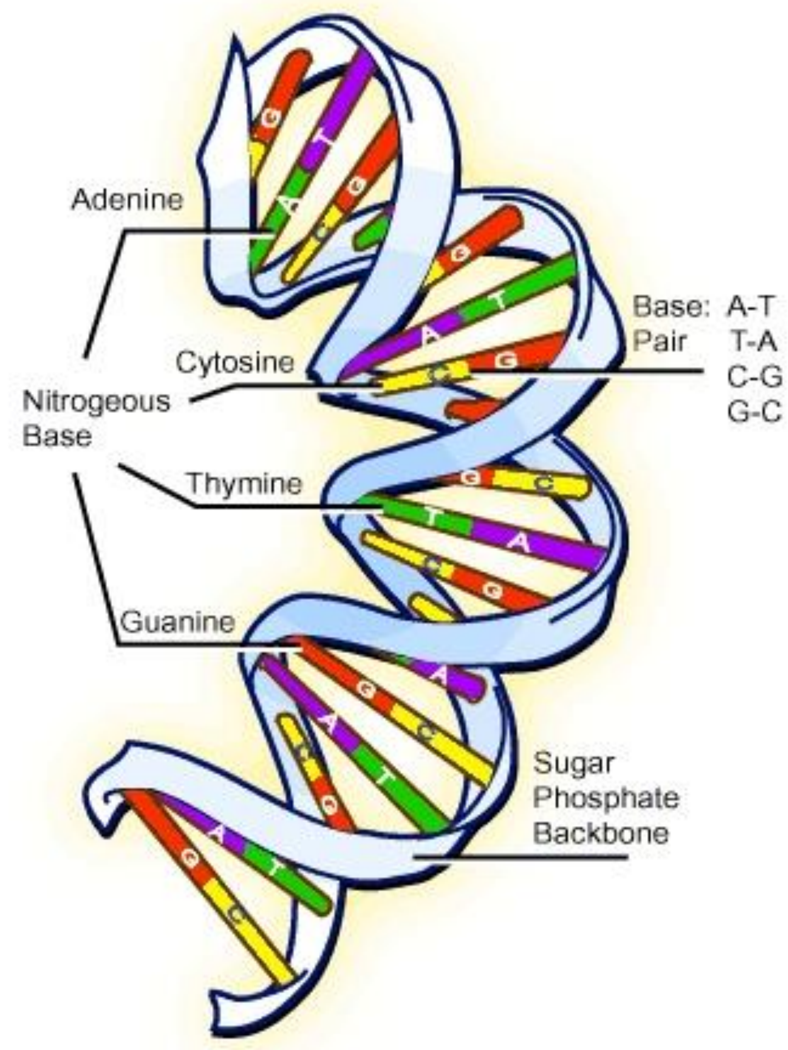
CSC 226

Algorithms and Data Structures: II
Longest Common Subsequence

Tianming Wei
twei@uvic.ca
ECS 466

Finding a Longest Common Subsequence





Applications

- Finding similarities between DNA sequences.
  - Test for genetic relationships between organisms.
  - Ex:
    - X = ACCGGTCGAGTGCGCGGAAGCCGGCCGAA
    - Y = GTCGTTCGGAATGCCGTTGCTCTGTAA
    - LCS = 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* $c = c_1 c_2 \cdots c_n$ is a sequence of characters (or symbols from an alphabet)

- A *substring* $s = s_1 s_2 \cdots s_m$ of a string $c = c_1 c_2 \cdots c_n$ is a string with

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

- A *subsequence* $x = x_1 x_2 \cdots x_r$ of a string $c = c_1 c_2 \cdots c_n$ is a string with

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

where

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

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 $s = s_1 s_2 \cdots s_n$ and $t = t_1 t_2 \cdots t_m$, 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 $x = x_1 x_2 \cdots x_n$ and $y = y_1 y_2 \cdots y_m$
- 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 $x = x_1 x_2 \cdots x_n$ and $y = y_1 y_2 \cdots y_m$ be strings, and let $z = z_1 z_2 \cdots z_k$ be an *lcs* of x and y. Then,

1. If $x_n = y_m$, then $z_k = x_n = y_m$ and $z_1 z_2 \cdots z_{k-1}$ is an lcs of $x_1 x_2 \cdots x_{n-1}$ and $y_1 y_2 \cdots y_{m-1}$.

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

3. If $x_n \neq y_m$ and $z_k \neq y_m$, then z is an lcs of x and $y_1 y_2 \cdots y_{m-1}$.

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

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

But then z could be 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) $z_1 z_2 \ldots z_{k-1}$ is an lcs of $x_1 x_2 \ldots x_{n-1}$ and $y_1 y_2 \ldots y_{m-1}$

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

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

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 $x_1\ x_2\ \ldots\ x_{n-1}$ and y

- Assume $x_n \neq y_m$ and $z_k \neq x_n$. Then z is common subsequence of $x_1\ x_2\ \ldots\ x_{n-1}$ 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 $x_1\ x_2\ \ldots\ x_{n-1}$ 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 $x_1\ x_2\ \ldots\ x_{n-1}$ and y and therefore (2)

Proof of (3)

- Works analogous to the proof of (2)

What can we conclude from the Theorem?

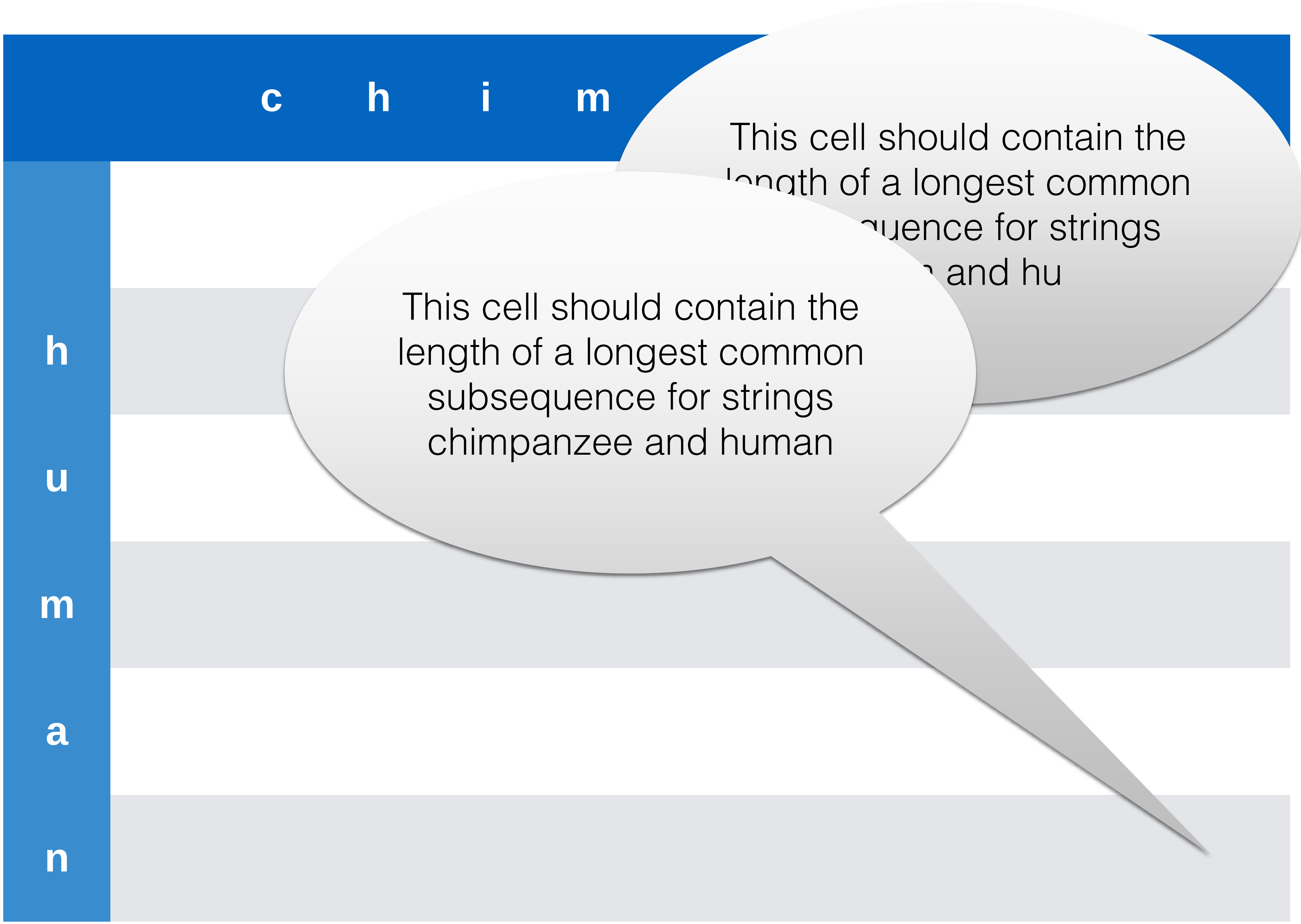
- Let $x = x_1\, x_2 \ldots x_n$ and $y = y_1\, y_2 \ldots y_m$ be strings, and let $z = z_1\, z_2 \ldots z_k$ be an lcs of x and y. Let llcs denote the length of an lcs. Then

  - If $x_n = y_m$ then

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

  - If $x_n \neq y_m$ then

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


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

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0										
u	0										
m	0										
a	0										
n	0										

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0	0									
u	0										
m	0										
a	0										
n	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								
u	0										
m	0										
a	0										
n	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							
u	0										
m	0										
a	0										
n	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										
m	0										
a	0										
n	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									
m	0										
a	0										
n	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								
m	0										
a	0										
n	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										
a	0										
n	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									
a	0										
n	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								
a	0										
n	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							
a	0										
n	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						
a	0										
n	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					
a	0										
n	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										
n	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									
n	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								
n	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							
n	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						
n	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					
n	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				
n	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			
n	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										

		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									

		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	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				

		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	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		

		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 llcs, but not an lcs

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

		c	h	i	m	p	a	n	z	e	e
		0	0	0	0	0	0	0	0	0	0
h	0										
u	0										
m	0										
a	0										
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									
u	0										
m	0										
a	0										
n	0										

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								
u	0										
m	0										
a	0										
n	0										

If x[i] ≠ 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							
u	0										
m	0										
a	0										
n	0										

If x[i] ≠ 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										
m	0										
a	0										
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									
m	0										
a	0										
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								
m	0										
a	0										
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										
a	0										
n	0										

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

		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									
a	0										
n	0										

If x[i] ≠ 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								
a	0										
n	0										

If x[i] ≠ 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							
a	0										
n	0										

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						
a	0										
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					
a	0										
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										
n	0										

If x[i] ≠ 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									
n	0										

If x[i] ≠ 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								
n	0										

If x[i] ≠ 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							
n	0										

If x[i] ≠ 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						
n	0										

If x[i] ≠ 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					
n	0										

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				
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			
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										

If x[i] ≠ 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									

If x[i] ≠ 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	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	<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	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	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**

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] $\leftarrow$ "↖"
    else if llcs[i-1, j] > llcs[i, j-1] **then**
      llcs[i, j] $\leftarrow$ llcs[i-1, j]; path[i, j] $\leftarrow$ "↑"
    else llcs[i, j] $\leftarrow$ llcs[i, j-1]; path[i, j] $\leftarrow$ "←"
return llcs & path

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

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

if path[i, j] = "↖" **then**

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

  print x[i]

else if path[i, j] = "↑" **then**

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

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

Running times

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