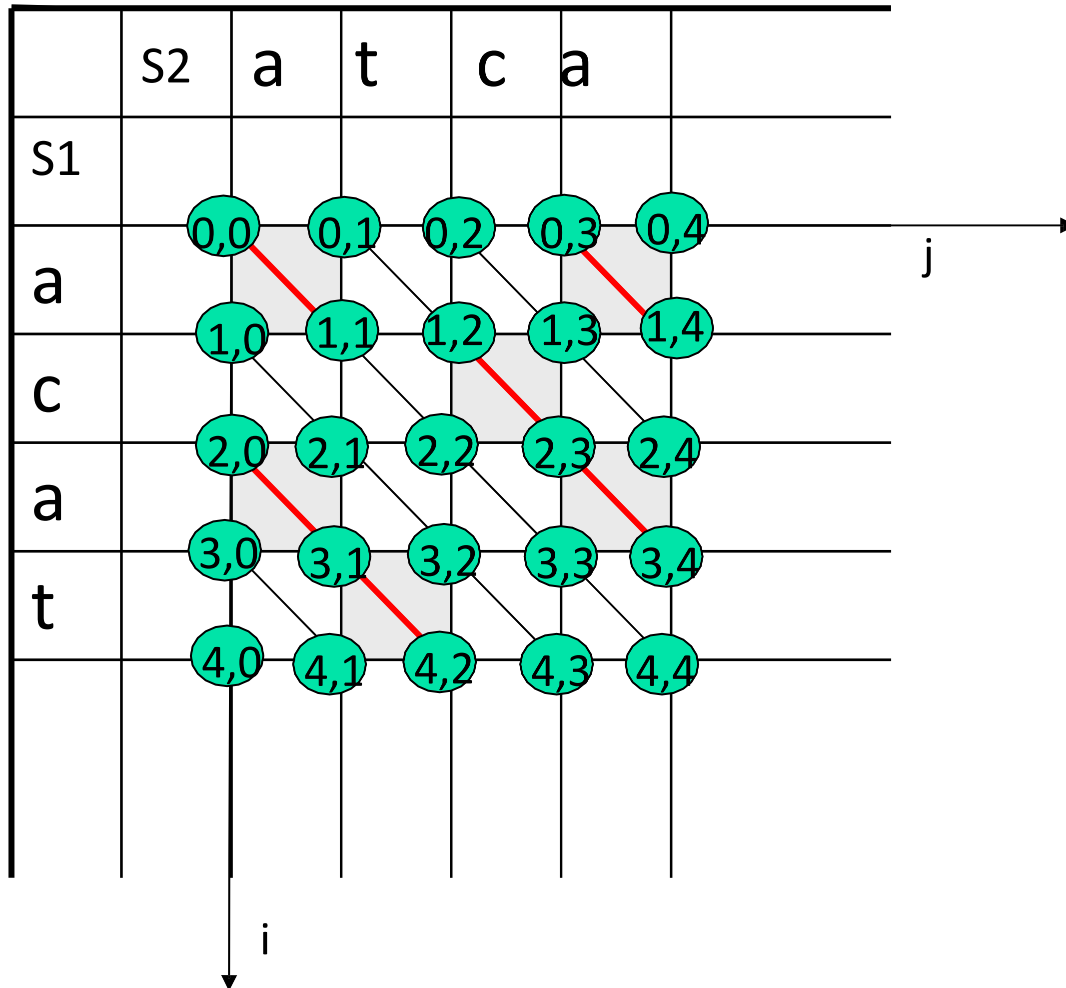
Designing Algorithms with Edit Graph Longest Common Subsequence

Lecture 07.04

by Marina Barsky

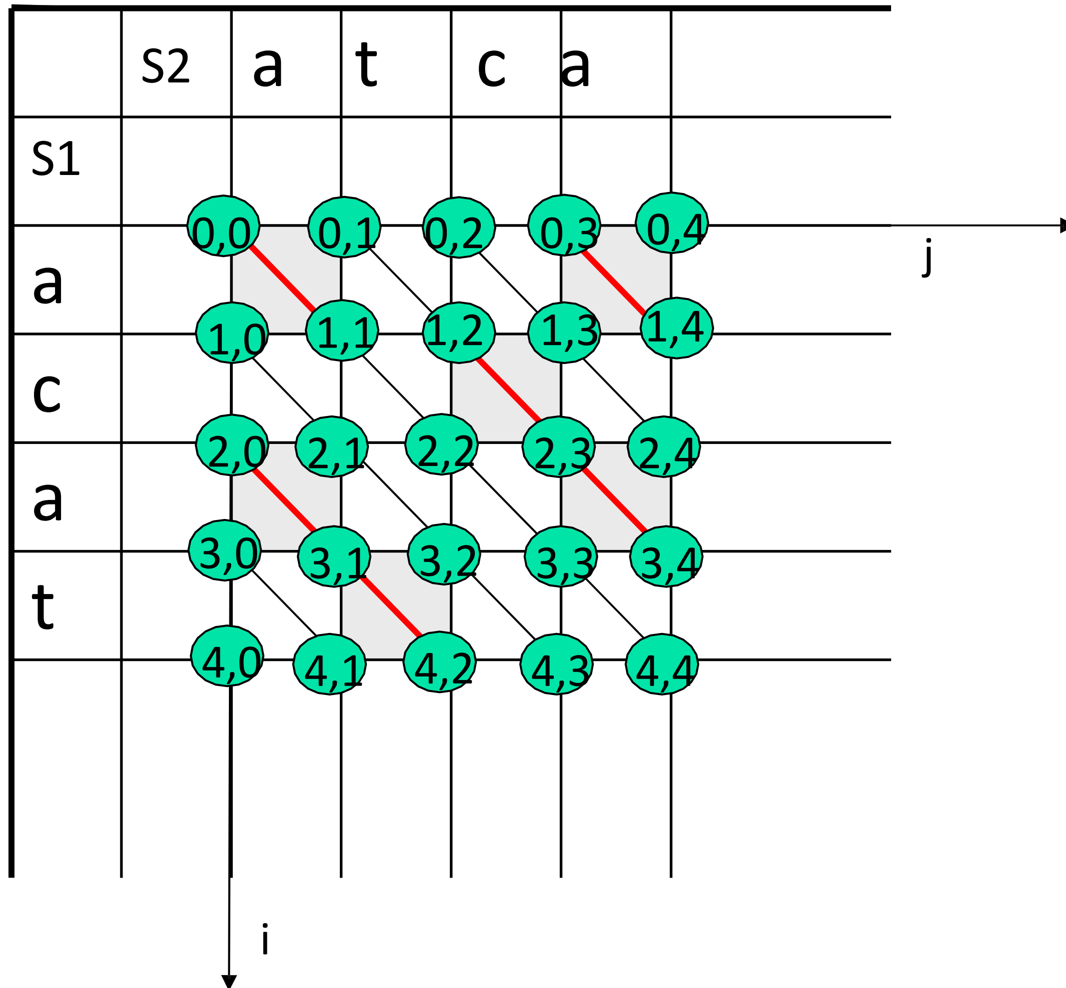
Recap: Useful abstraction: *edit graph*



An ***edit graph*** for a pair of strings S_1 and S_2 has (N+1)*(M+1) vertices, each labeled with a corresponding pair (i,j), $0 \leq i \leq N$, $0 \leq j \leq M$

The edges are **directed** and their weight depends on the specific string problem: for the edit distance problem – red edges have cost 0, black edges have cost 1

The cheapest path in the edit graph



The cost of a **cheapest path** from vertex (0,0) to vertex (*N,M*) in this edit graph corresponds to the **edit distance** between S1 and S2, and the path itself represents a series of edit operations and an optimal alignment of S1 with S2

Motivation: sequence similarity

- ❑ Life is based on a repertoire of successful structural and interrelated building blocks which are passed around
- ❑ Biological universality occurs at many levels of details, so we can compare not only the sequence data, but 3D shapes, chemical pathways, morphological features etc.

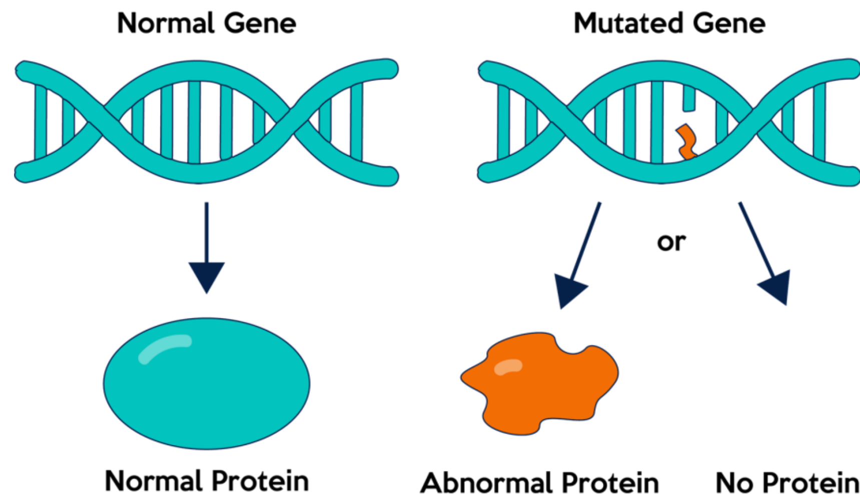
“Everything in life is so similar that the same genes that work in flies are the ones that work in humans” (Wieschaus, 1995)

Why compare biosequences

- The biological sequences (DNA, RNA or protein) encode and reflect the higher-level molecular structures and mechanisms
- **High sequence similarity <u>usually</u> implies significant structural and functional similarity**
- A tractable, though partly heuristic way to infer structure and function of an unknown protein is to search for the similar known proteins at the sequence level: **similar but not identical!**

Note of caution in interpreting sequence similarity

- There is not a one-to-one correspondence between similar sequences and similar structures or between sequences and functions:
  - Quite similar structures can be obtained from completely unrelated sequences
  - Very similar sequences can produce very different structures depending on the location of a change



Edit distance as a measure of similarity

S1	*a*	*-*	*c*	*a*	*t*
S2	*a*	*t*	*c*	*a*	*-*

If the number of basic evolutionary events is small, we infer that the divergence between S1 and S2 happened not so long time ago, and that the two strings are still *similar*

The **smaller** is the **edit distance** between 2 strings, the **more similar** they are

Optimal alignment

S1	a	-	c	a	t
S2	a	t	c	a	-

Evolutionary explanation:

S_2 evolved from S_1 by a series of the following mutations:

Insertion of nucleotide *t* at position 2

Deletion of nucleotide *t* at position 5

An optimal alignment is not unique

S1	-	a	t	t	a	a	g
S2	t	a	-	t	c	a	g

S1	-	a	t	t	a	a	g
S2	t	a	t	c	a	-	g

2 different alignments with the optimal minimal cost 3

The exact sequence of changes (mutations) cannot be determined

The edit-distance based similarity metric

S	a	c	c	g	c
S1	a	c		g	c

Edit distance: 1

S	a	c	c	g	c
S2		c	c	g	t

Edit distance: 2

The smaller is the edit distance, the larger is the similarity.

S is more similar to S1 than to S2

The edit-distance based similarity metric: not enough

S	**a**	**c**	c	g	**c**
S1	**a**	**c**		t	**c**

S	**a**		**c**	**c**		**g**	**c**
S2	**a**	c	**c**	**c**	t	**g**	**c**

The edit distance alone is not always a sufficient metric to characterize similarity between strings

In these 2 examples, the edit distance between S and S1 is the same as an edit distance between S and S2, but it is intuitively clear that S is more similar to S2 than to S1, since they share more identical characters

We want to evaluate what was preserved rather than what changed to infer similarity

The longest common **substring**

- The longest substring, common to both strings: the longest sequence of consecutive characters which occur in both strings

The longest sequence of ***consecutive matches***

S	**a**	**c**	c	g	**c**
S1	**a**	**c**		t	**c**

S	**a**		**c**	**c**		**g**	**c**
S2	**a**	c	**c**	**c**	t	**g**	**c**

The longest common **subsequence**

- A ***subsequence*** of a string *S* is a subset of characters of *S* in their original relative order. A subsequence does not need to consist of the consecutive characters of S

- Given 2 strings S1 and S2, ***a common subsequence*** for 2 strings is a subsequence which appears both in S1 and S2

- ***The longest common subsequence*** is a longest between all possible subsequences of S1 and S2

Substring vs subsequence

w	i	n	t	e	r	s

w	**i**	n	**t**	e	r	**s**

its – a subsequence of *winters*

w	**i**	**n**	**t**	**e**	**r**	s

inter – both substring and subsequence of *winters*

Longest Common **Subsequence (LCS)**

m	a	d	b	u	n	n	y
b	a	d	m	o	n	e	y

Common subsequence of length 3

m	a	d	b	u	n	n	y
b	a	d	m	o	n	e	y

Common subsequence of length 4

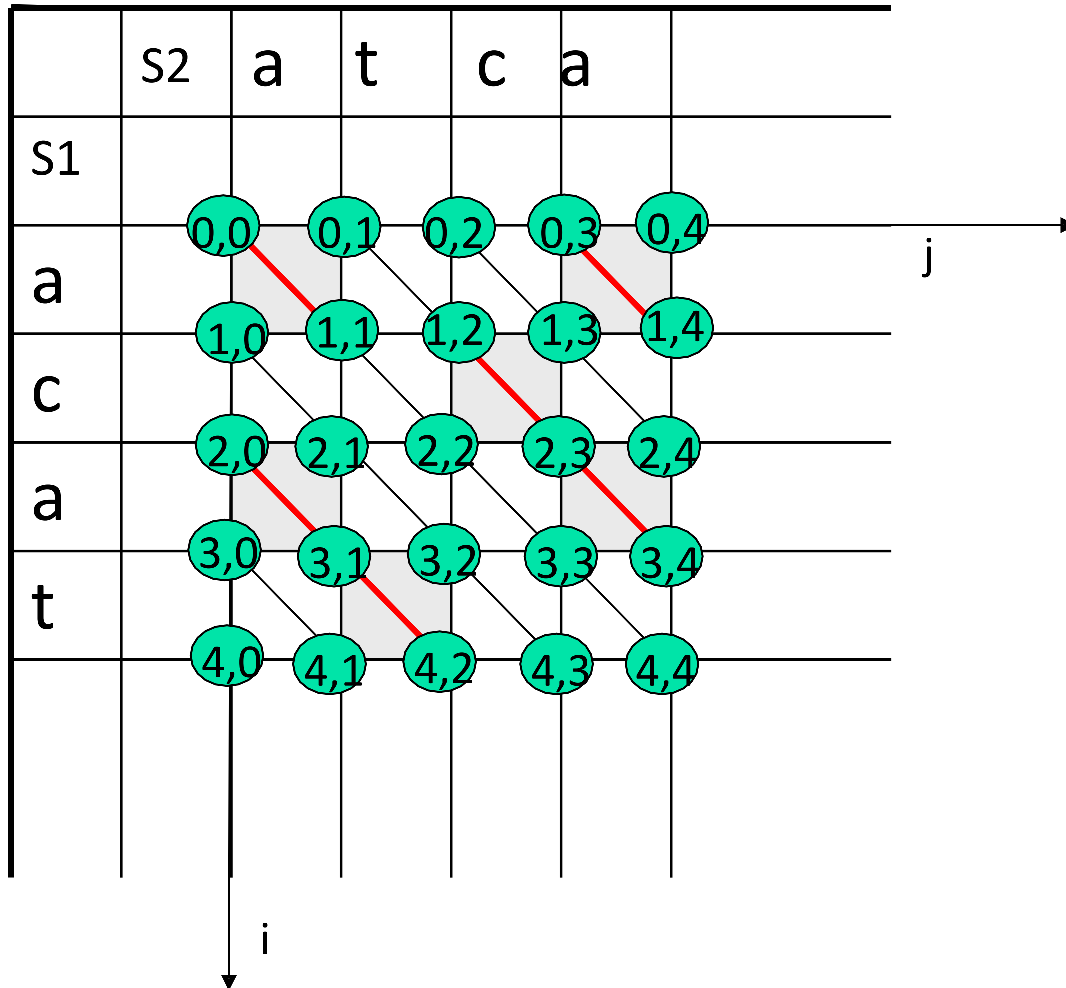
How can we be sure that ***adny*** is the **longest** common subsequence?

The LCS problem

Input: 2 strings S_1 and S_2

Output: the length of *the longest subsequence common to* both strings along with the sub-sequence itself

Edit Graph for LCS problem

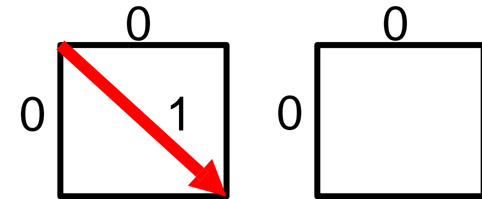


An ***edit graph*** for a pair of strings S_1 and S_2 can be used to solve the LCS problem

We need to change edge weights: in the LCS problem we are only interested in a sequence of matches – red edges have cost 1, black edges have cost 0

Dynamic Programming solution for LCS. Edit graph

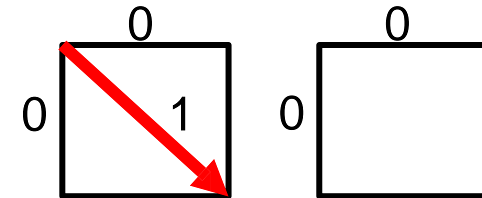
		b	*a*	*d*	*m*	*o*	*n*	*e*	*y*
	0								
m									
a									
d									
b									
u									
n									
n									
y									



Since we are interested in a longest sequence of matches, we give to the red edges cost 1 and to all the other edges cost 0

Since aligning 2 different characters does not contribute to the total score we do not consider the diagonal edges in case of mismatch

Dynamic Programming solution for LCS. Greedy path



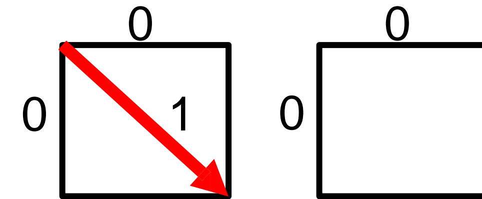
		b	*a*	*d*	*m*	*o*	*n*	*e*	*y*
	0								
m									
a									
d									
b									
u									
n									
n									
y									

The LCS problem can be reduced to finding the greediest (the longest) path through matches -

the path with the largest cost

Base condition

		b	*a*	*d*	*m*	*o*	*n*	*e*	*y*
	0	0	0	0	0	0	0	0	0
m	0								
a	0								
d	0								
b	0								
u	0								
n	0								
n	0								
y	0								



All the black edges are of cost 0, so moving strictly right or down gives paths of a total cost 0

LCS. Recurrence relation

$$\text{COST}(i,j)=\mathbf{max}\begin{cases}\text{COST}(i-1,j)\\ \text{COST}(i,j-1)\\ \text{COST}(i-1,j-1)+1 \text{ if } S1[i]=S2[j]\end{cases}$$

We only consider the diagonal edge if the characters match

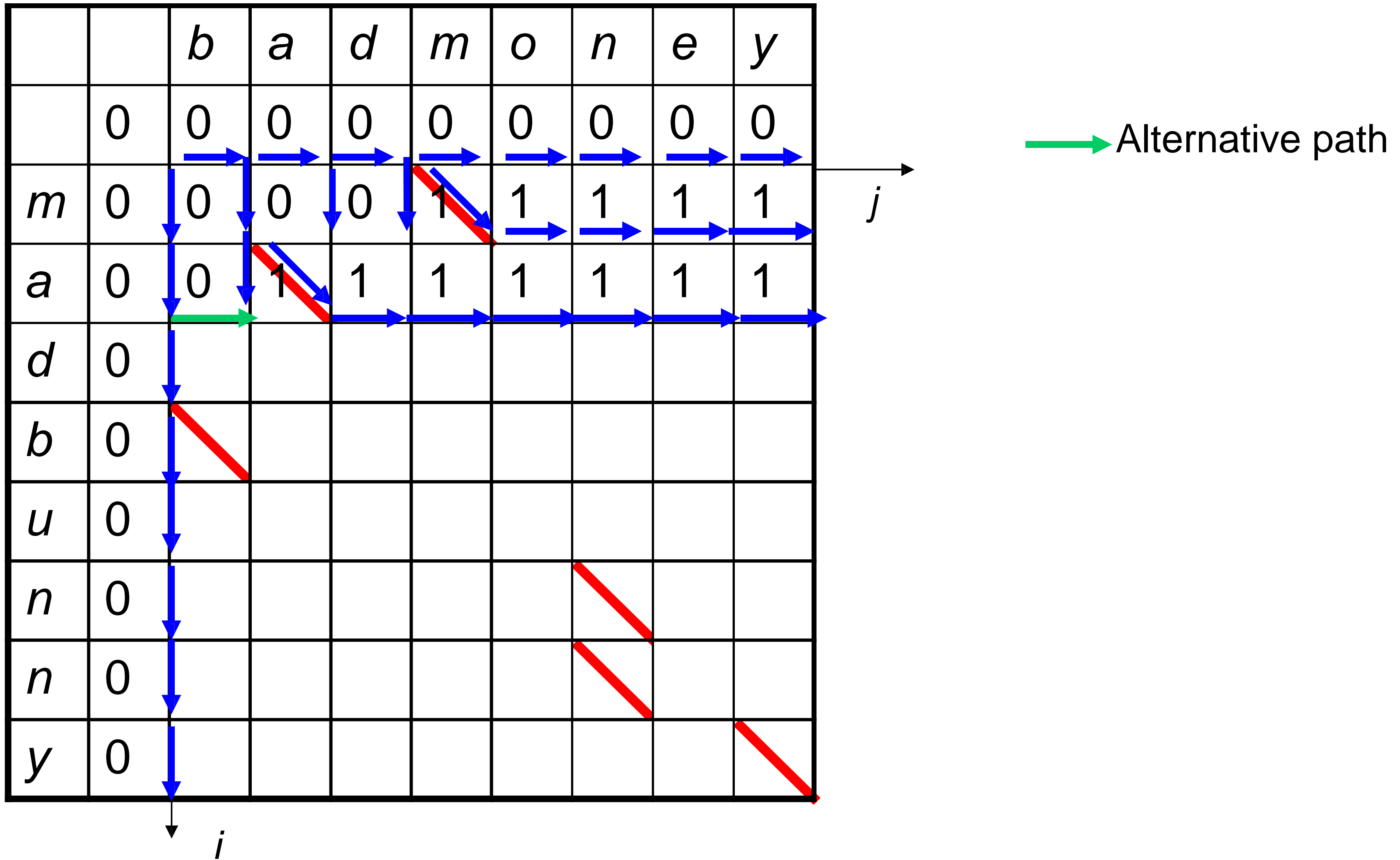
Tabular computation. Row 1

		b	*a*	*d*	*m*	*o*	*n*	*e*	*y*
	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	1	1	1	1
a	0								
d	0								
b	0								
u	0								
n	0								
n	0								
y	0								

j

i

Tabular computation. Row 2



Tabular computation. Row 3

		b	*a*	*d*	*m*	*o*	*n*	*e*	*y*
	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	1	1	1	1
a	0	0	1	1	1	1	1	1	1
d	0	0	1	2	2	2	2	2	2
b	0								
u	0								
n	0								
n	0								
y	0								

j

i

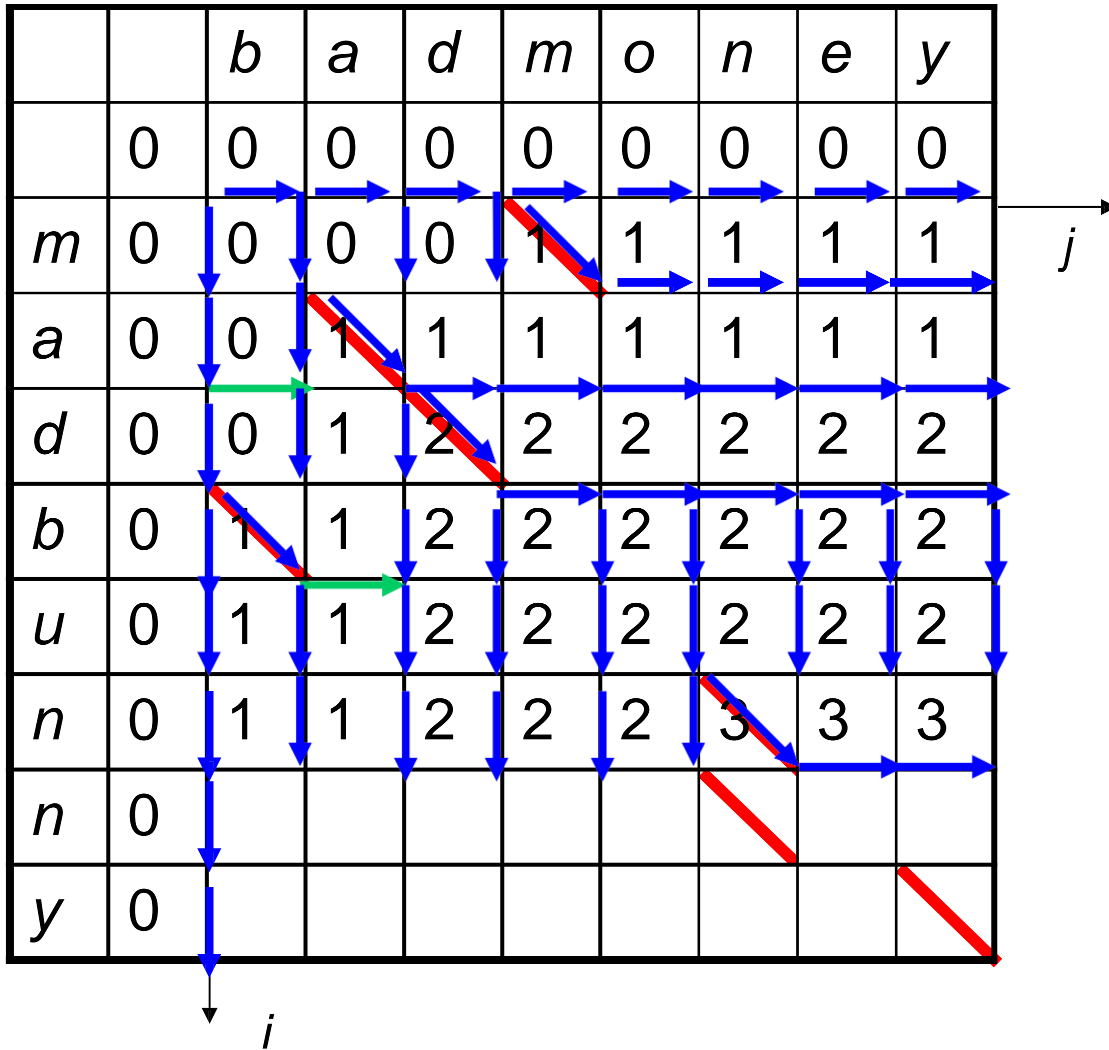
Tabular computation. Row 4

		b	*a*	*d*	*m*	*o*	*n*	*e*	*y*
	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	1	1	1	1
a	0	0	1	1	1	1	1	1	1
d	0	0	1	2	2	2	2	2	2
b	0	1	1	2	2	2	2	2	2
u	0								
n	0								
n	0								
y	0								

j

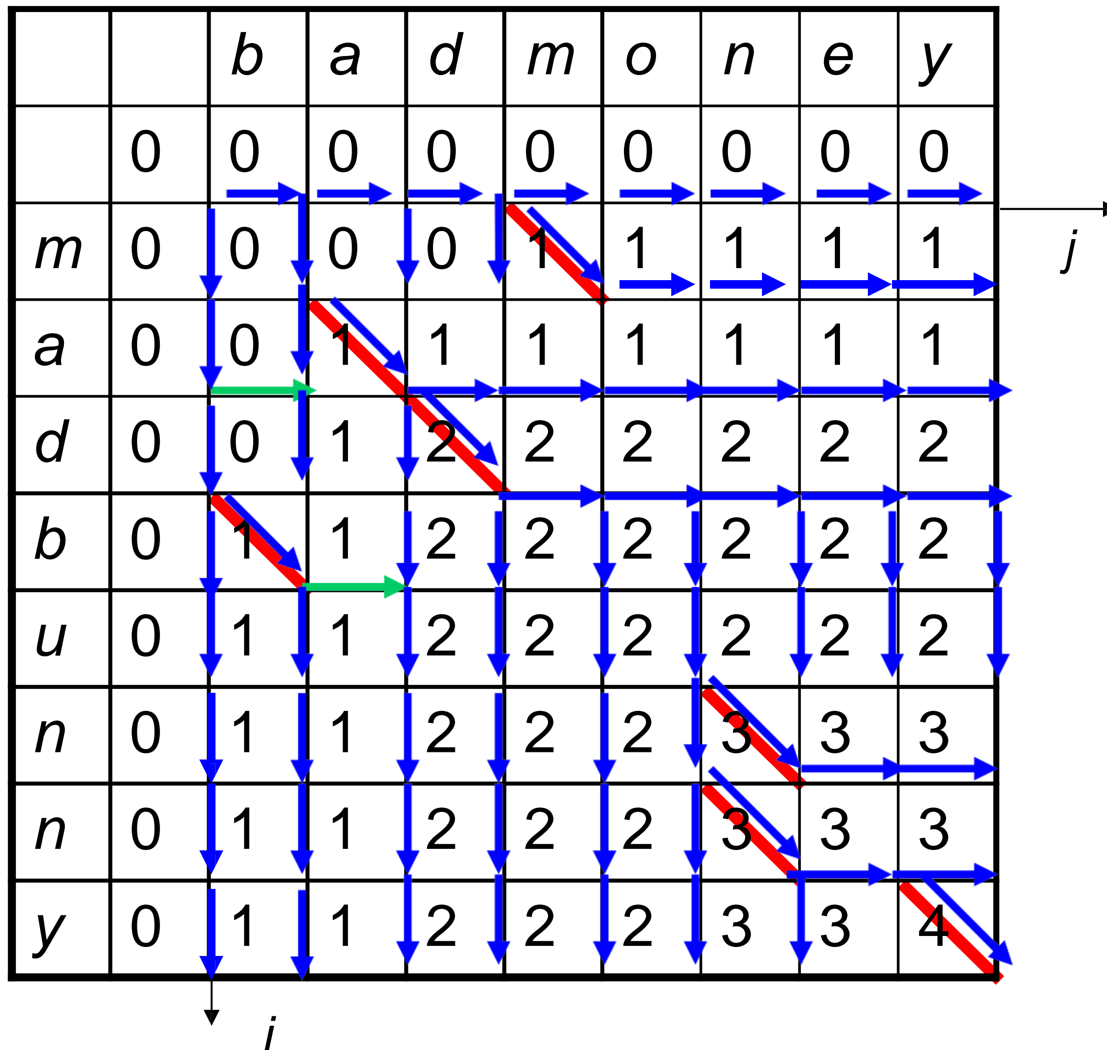
i

Tabular computation. Rows 5,6



		b	*a*	*d*	*m*	*o*	*n*	*e*	*y*
	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	1	1	1	1
a	0	0	1	1	1	1	1	1	1
d	0	0	1	2	2	2	2	2	2
b	0	1	1	2	2	2	2	2	2
u	0	1	1	2	2	2	2	2	2
n	0	1	1	2	2	2	3	3	3
n	0								
y	0								

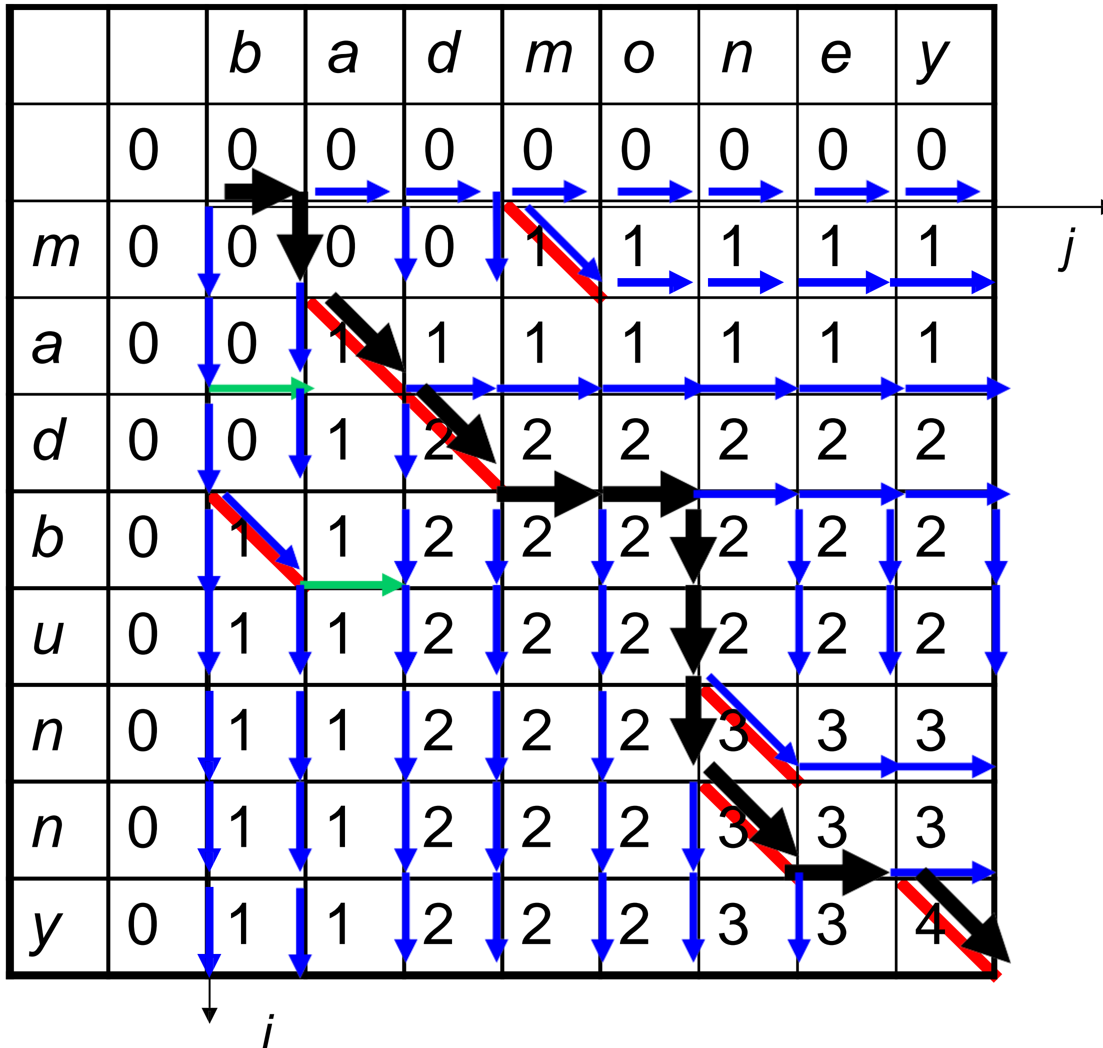
Tabular computation. End



		b	a	d	m	o	n	e	y
	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	1	1	1	1
a	0	0	1	1	1	1	1	1	1
d	0	0	1	2	2	2	2	2	2
b	0	1	1	2	2	2	2	2	2
u	0	1	1	2	2	2	2	2	2
n	0	1	1	2	2	2	3	3	3
n	0	1	1	2	2	2	3	3	3
y	0	1	1	2	2	2	3	3	4

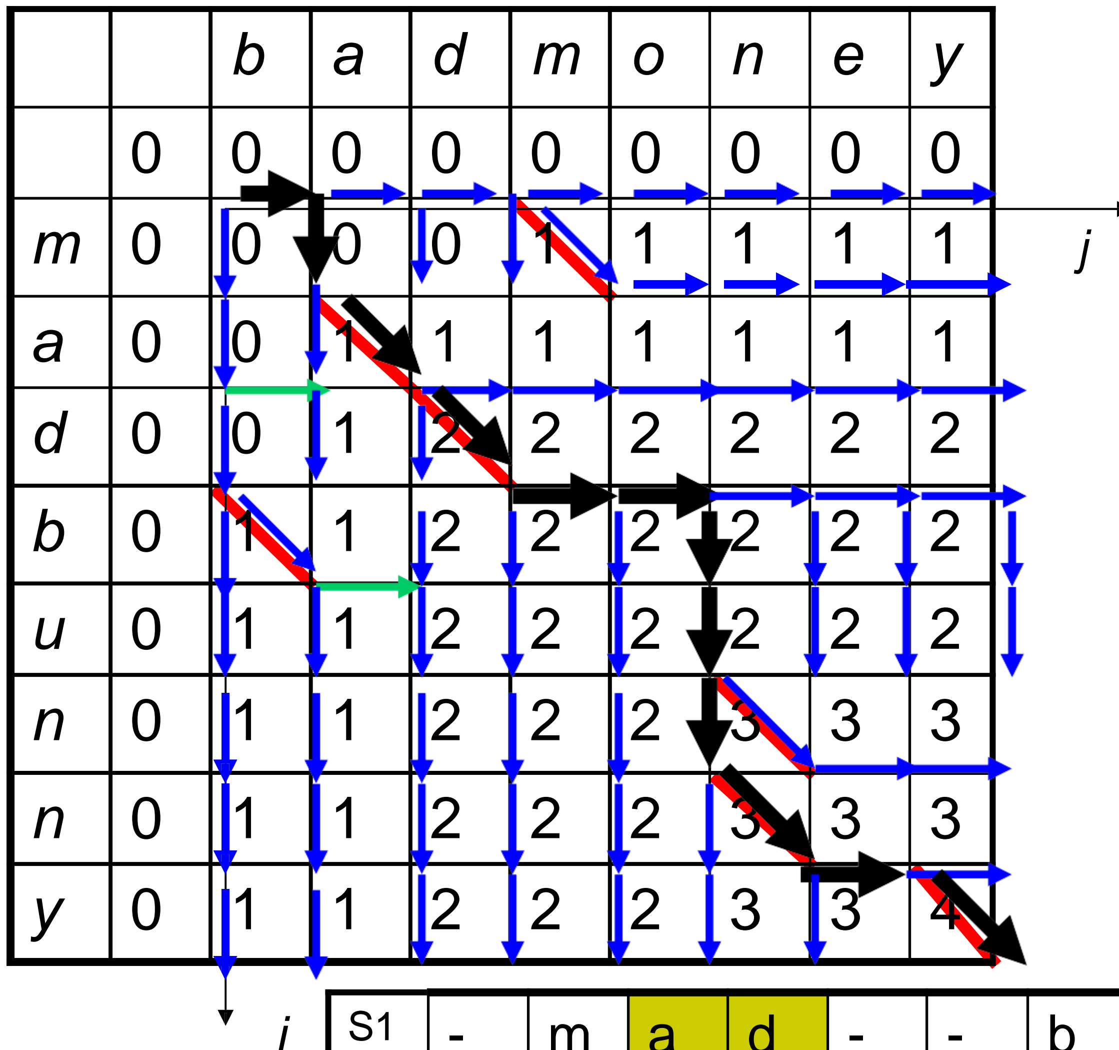
Read the length of the longest common subsequence in cell [N][M]

LCS. Traceback



Find the subsequence itself tracing the sequence of matches backwards

LCS. Alignment



		b	a	d	m	o	n	e	y
	0	0	0	0	0	0	0	0	0
m	0	0	0	0	1	1	1	1	1
a	0	0	1	1	1	1	1	1	1
d	0	0	1	2	2	2	2	2	2
b	0	1	1	2	2	2	2	2	2
u	0	1	1	2	2	2	2	2	2
n	0	1	1	2	2	2	3	3	3
n	0	1	1	2	2	2	3	3	3
y	0	1	1	2	2	2	3	3	4

S1	-	m	a	d	-	-	b	u	n	n	-	y
S2	b	-	a	d	m	o	-	-	-	n	e	y

Note, that **only the matches are aligned**, since the problem we are solving – find the longest sequence of matches

We don't count the number of edit operations, since their cost in this model is 0

The edit-distance based similarity metric: not enough

S	**a**	**c**	c	g	**c**
S1	**a**	**c**		t	**c**

S	**a**		**c**	**c**		**g**	**c**
S2	**a**	c	**c**	**c**	t	**g**	**c**

In these 2 examples, the edit distance between S and S1 is the same as an edit distance between S and S2, but it is intuitively clear that S is more similar to S2 than to S1, since they share more identical characters

The LCS-based similarity metric: not enough

S	a	c	c	c
S1	a	c	-	c

S	a	-	c	c	-	-	c
S2	a	t	c	-	t	g	c

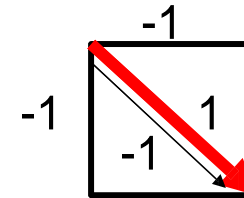
The longer is the LCS, the more similar are two strings

The LCS alone is not a sufficient similarity metric

In these 2 examples, the LCS of S and S1 is the same as the LCS of S and S2, but it is intuitively clear that S is more similar to S1 than to S2, since they have less different characters

We want to score both the matches and the differences

Basic optimal alignment scores



	S2	t	g	c	a	t	a
S1							
a							
t							
c							
t							
g							
a							
t							

Let us set the simplest weights of the edges:

For a match: award of 1

For a mismatch: penalty of -1

For a gap (insertion/deletion):

penalty of -1

Then the maximum cost of the path in the edit graph will give a numerical score of the similarity between S1 and S2: large positive values – two strings are similar, negative or low positive values – the strings are different

Everything else is exactly the same

Exercise. Longest Increasing Subsequence (LIS)

- Given a sequence of n numbers $A1 \ldots An$, determine a subsequence (*n*ot necessarily contiguous) of maximum length in which the values in the subsequence form a strictly increasing sequence.

Exercise. LIS - solution

- Given a sequence of *n* numbers $A1 \dots An$, determine a subsequence (*n*ot necessarily contiguous) of maximum length in which the values in the subsequence form a strictly increasing sequence.

- This problem can be reduced to the LCS between $A1 \dots An$, and *sorted*$(A1 \dots An)$
- If sequence A contains a permutation of all numbers from 1 to n – then the solution is just LCS between sequence values and indices.