String Comparison

Slides by Carl Kingsford

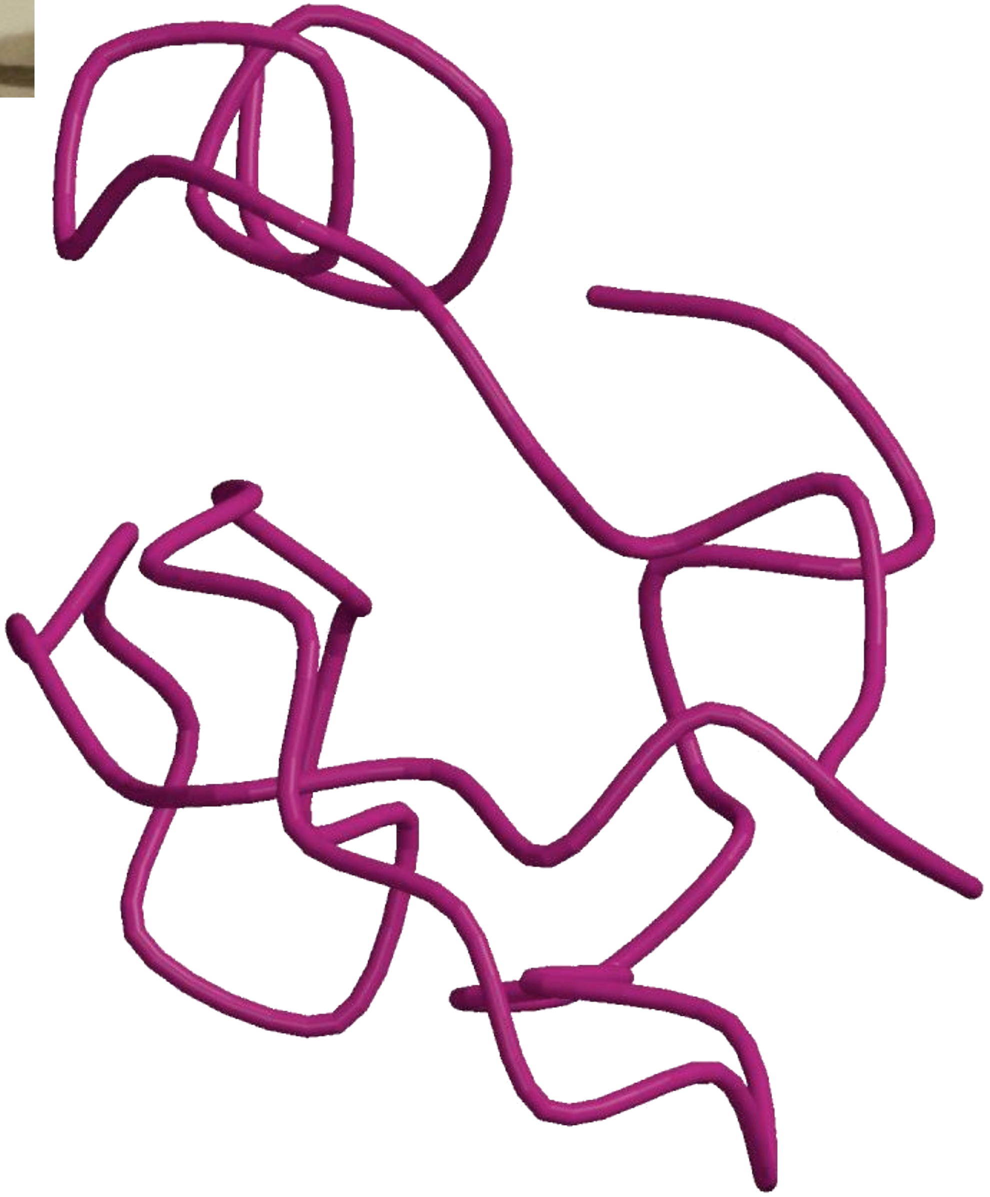
Why compare DNA or protein sequences?

Partial CTCF protein sequence in 8 organisms:

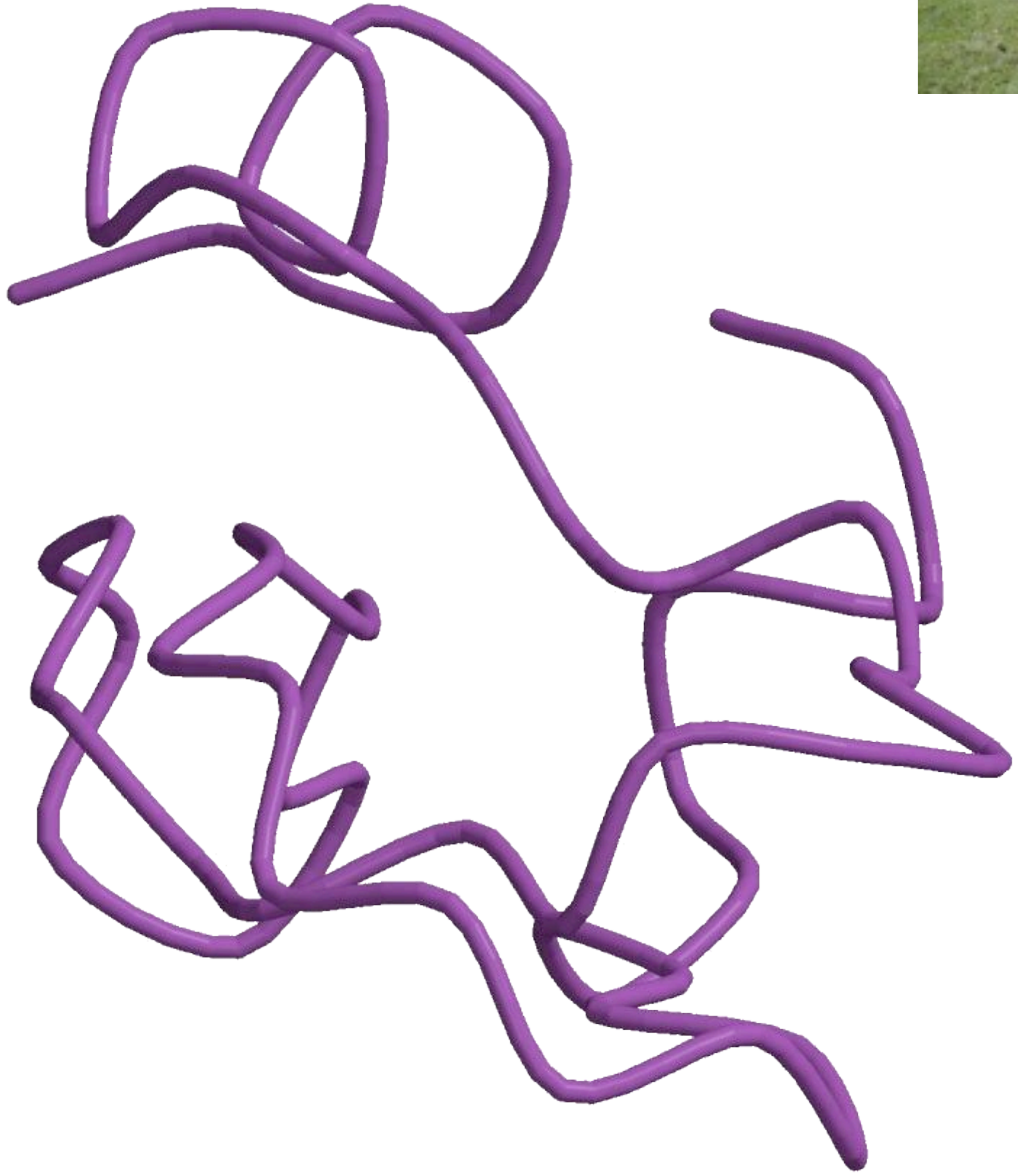
```
H. sapiens       -EDSSDS-ENAEPDLDDNEDEEEPAVEIEPEPE----------PQPVTPA
P. troglodytes   -EDSSDS-ENAEPDLDDNEDEEEPAVEIEPEPE----------PQPVTPA
C. lupus         -EDSSDS-ENAEPDLDDNEDEEEPAVEIEPEPE----------PQPVTPA
B. taurus        -EDSSDS-ENAEPDLDDNEDEEEPAVEIEPEPE----------PQPVTPA
M. musculus      -EDSSDSEENAEPDLDDNEEEEEPAVEIEPEPE--PQPQPPPPPQPVAPA
R. norvegicus    -EDSSDS-ENAEPDLDDNEEEEEPAVEIEPEPEPQPQPQPQPQPQPVAPA
G. gallus        -EDSSDSEENAEPDLDDNEDEEETAVEIEAEPE----------VSAEAPA
D. rerio         DDDDDDSDEHGEPDLDDIDEEDEDDL-LDEDQMGLLDQAPPSVPIP-APA
```

- Identify important sequences by finding conserved regions.
- Find genes similar to known genes.
- Understand evolutionary relationships and distances (D. rerio aka zebrafish is farther from humans than G. gallus aka chicken).
- Interface to databases of genetic sequences.
- As a step in genome assembly, and other sequence analysis tasks.
- Provide hints about protein structure and function (next slide).

Sequence can reveal structure



(a) 1dtk



(b) 5pti

1dtk XAKY**C**KL**P**LRI**GPCK**RK**I**PSFY**Y**KW**KA**KQ**C**LP**F**D**Y**S**GC**GGNA**N**R**FK**TI**E**E**C**R**RTC**V**G**-

5pti RPDF**C**LE**P**PYT**GPCK**AR**I**IRYF**Y**NA**KA**GL**C**QT**F**V**Y**G**GC**RAKR**N**N**FK**SA**E**D**C**M**RTC**G**G**A

The Simplest String Comparison Problem

Given: Two strings

$$a = a_1a_2a_3a_4 \ldots a_m$$
$$b = b_1b_2b_3b_4 \ldots b_n$$

where a_i, b_i are letters from some alphabet like {A,C,G,T}.

Compute how similar the two strings are.

What do we mean by "similar"?

Edit distance between strings *a* and *b* = the smallest number of the following operations that are needed to transform *a* into *b*:

- mutate (replace) a character
- delete a character
- insert a character

$$\text{riddle} \xrightarrow{\text{delete}} \text{ridle} \xrightarrow{\text{mutate}} \text{riple} \xrightarrow{\text{insert}} \text{triple}$$

Representing edits as alignments

```
prin-ciple
|||| |||xx
prinncipal
(1 gap, 2 mm)
```

```
prin-cip-le
|||| ||| |
prinncipal-
(3 gaps, 0 mm)
```

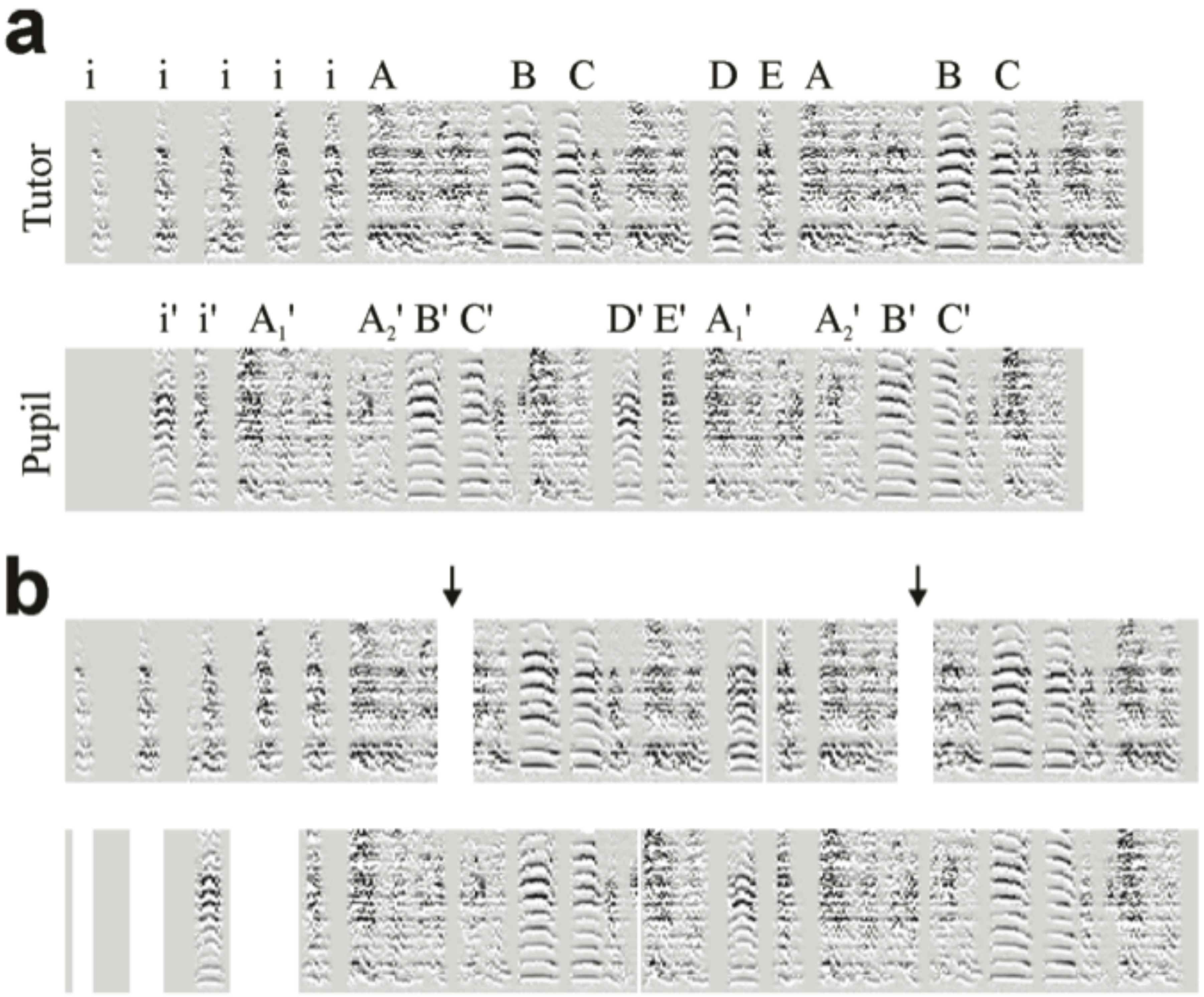
```
misspell
||| ||||
mis-pell
(1 gap)
```

```
prehistoric
   ||||||||
---historic
(3 gaps)
```

```
aa-bb-ccaabb
|x || | | |
ababbbc-a-b-
(5 gaps, 1 mm)
```

```
al-go-rithm-
|| xx ||x |
alKhwariz-mi
(4 gaps, 3 mm)
```

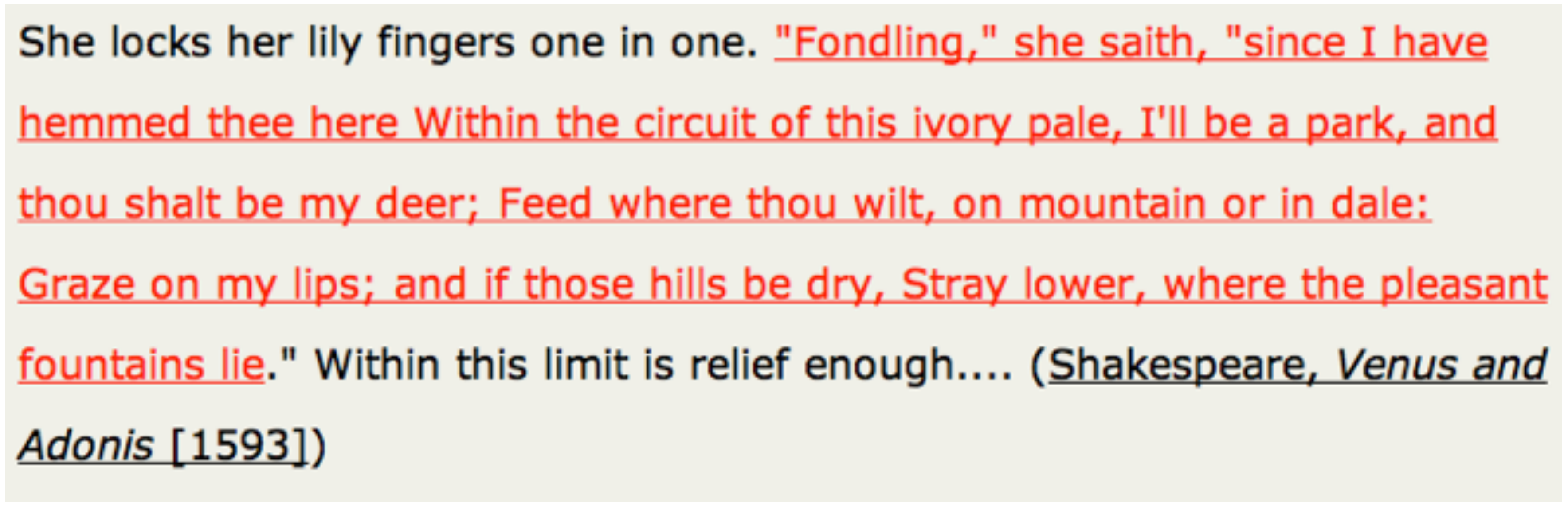
Comparing Bird Songs



Florian et al. Hidden Markov Models in the Neurosciences

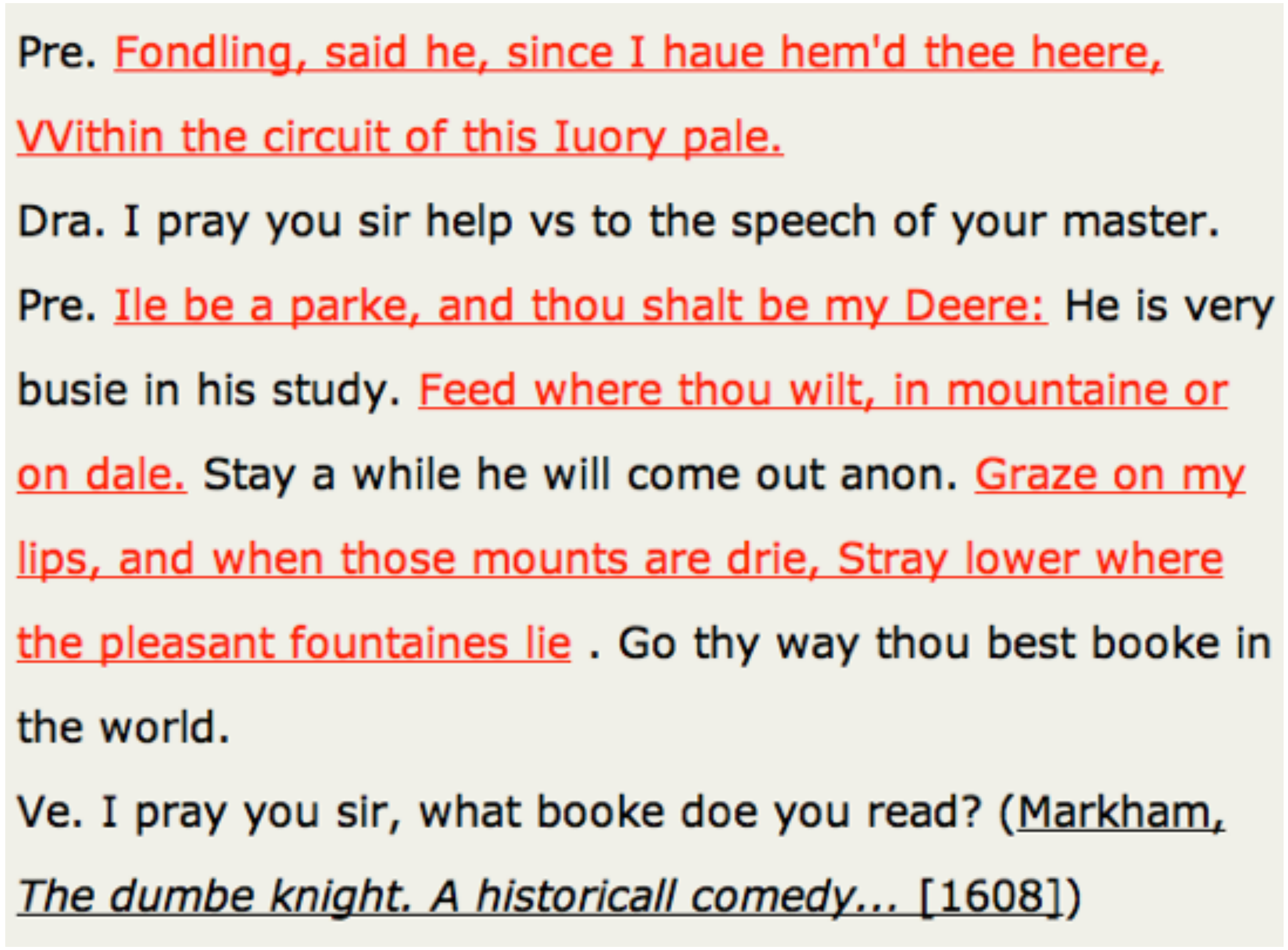
Tracing Textual Influences

Example from Horton, Olsen, Roe, Digital Studies / Le champ numérique, Vol 2, No 1 (2010)



She locks her lily fingers one in one. "Fondling," she saith, "since I have hemmed thee here Within the circuit of this ivory pale, I'll be a park, and thou shalt be my deer; Feed where thou wilt, on mountain or in dale: Graze on my lips; and if those hills be dry, Stray lower, where the pleasant fountains lie." Within this limit is relief enough.... (Shakespeare, *Venus and Adonis* [1593])

This later play by Markham references Shakespeare's poem.



Pre. Fondling, said he, since I haue hem'd thee heere, VVithin the circuit of this Iuory pale.

Dra. I pray you sir help vs to the speech of your master.

Pre. Ile be a parke, and thou shalt be my Deere: He is very busie in his study. Feed where thou wilt, in mountaine or on dale. Stay a while he will come out anon. Graze on my lips, and when those mounts are drie, Stray lower where the pleasant fountaines lie . Go thy way thou best booke in the world.

Ve. I pray you sir, what booke doe you read? (Markham, *The dumbe knight. A historicall comedy...* [1608])

Common passages identified by sequence alignment algorithms.

The String Alignment Problem

Parameters:

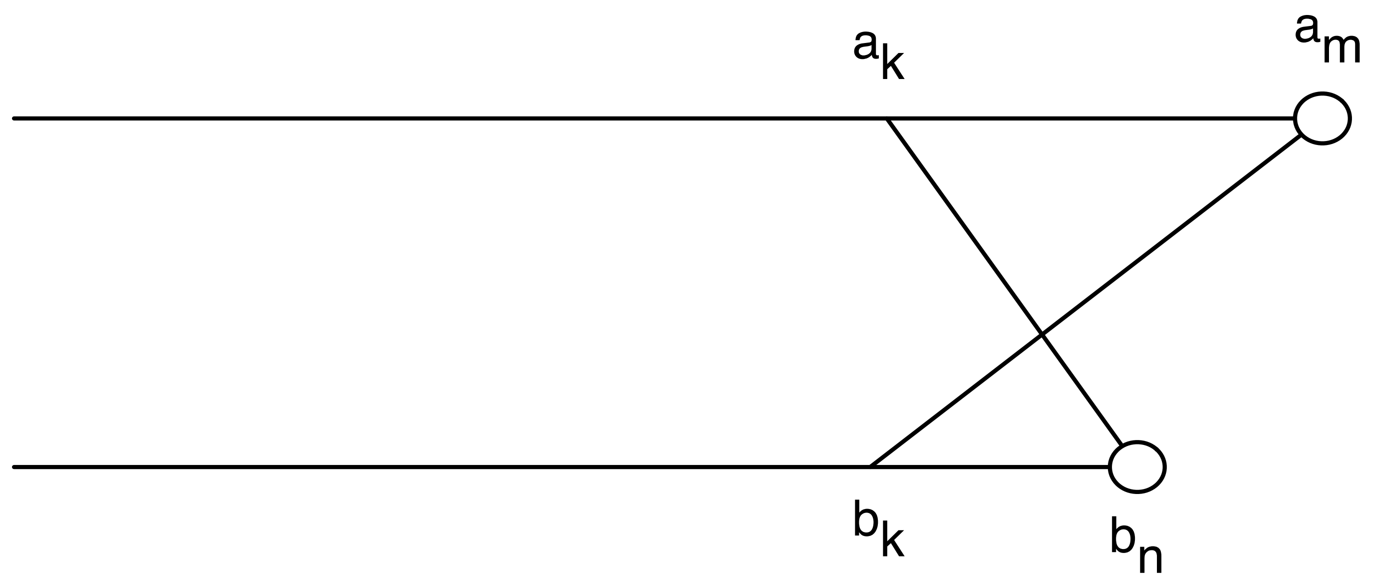
- "*gap*" is the cost of inserting a "-" character, representing an insertion or deletion
- *cost*(x,y) is the cost of aligning character x with character y.
  In the simplest case, *cost*(x,x) = 0 and *cost*(x,y) = mismatch penalty.

Goal:

- Can compute the edit distance by finding the **lowest cost alignment**.
- Cost of an alignment is: sum of the *cost*(x,y) for the pairs of characters that are aligned + *gap* × number of - characters inserted.

No Crossing Rule Forbids #4

4. a_m is matched to some b_j ($j \neq n$) and b_n is matched to some a_k ($k \neq m$).



So, the only possibilities for what happens to the last characters are:

1. (a_m, b_n) are matched to each other
2. a_m is not matched at all
3. b_n is not matched at all

Recursive Solution

Turn the 3 possibilities into 3 cases of a recurrence:

$$\underset{\substack{\uparrow\\ \text{Cost of the optimal}\\ \text{alignment between}\\ a_1 \ldots a_i \text{ and } b_1 \ldots b_j}}{OPT}(i,j) = \min \begin{cases} \text{cost}(a_i, b_j) + OPT(i-1, j-1) & \text{match } a_i, b_j \\ \text{gap} + OPT(i-1, j) & a_i \text{ is not matched} \\ \text{gap} + \underset{\substack{\uparrow\\ \text{Written in terms of}\\ \text{the costs of smaller}\\ \text{problems}}}{OPT}(i, j-1) & b_j \text{ is not matched} \end{cases}$$

Key: we don't know which of the 3 possibilities is the right one, so we try them all.

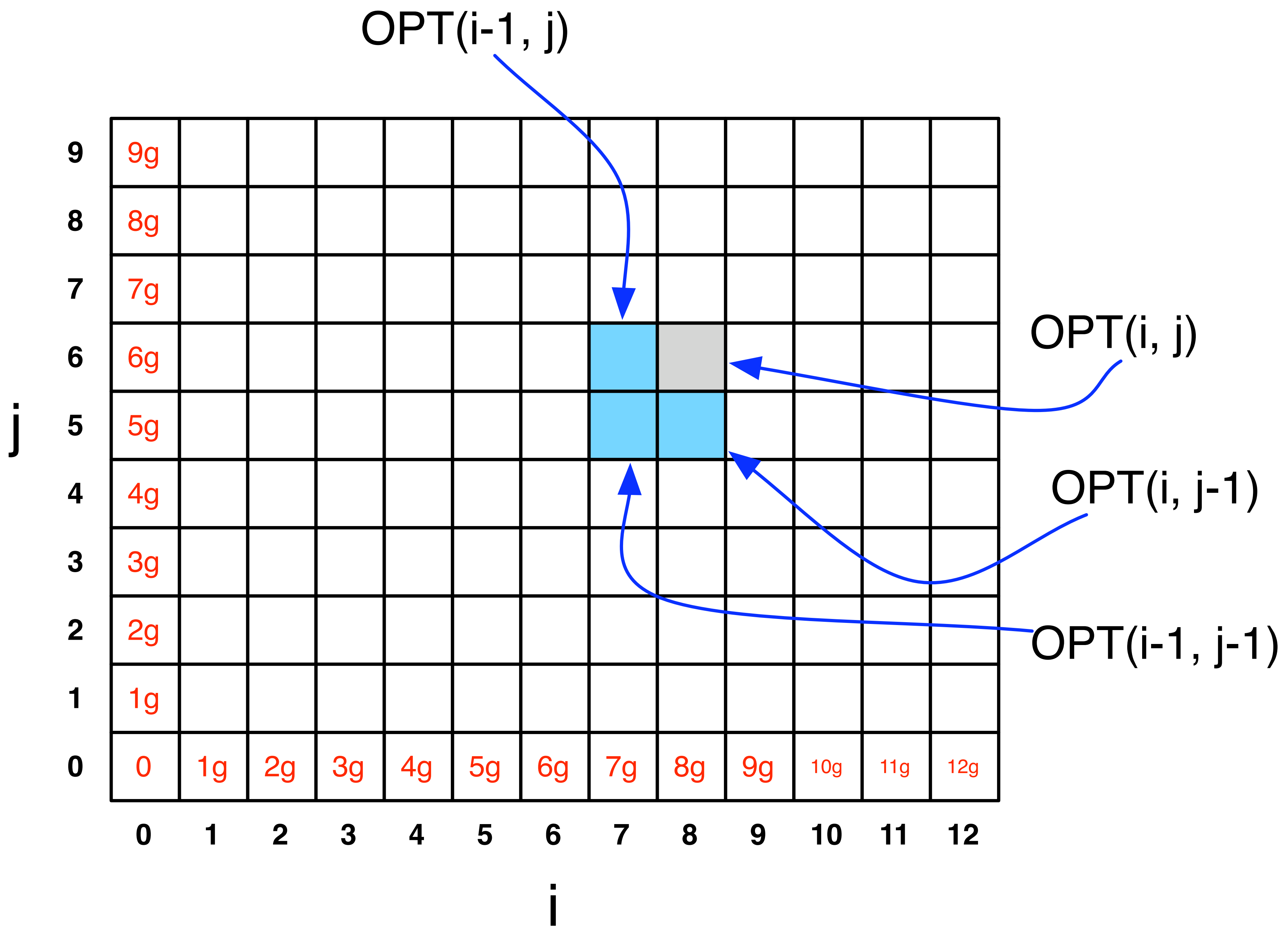
<u>Base case:</u> $OPT(i,0) = i \times \text{gap}$ and $OPT(0,j) = j \times \text{gap}$

(Aligning *i* characters to 0 characters must use *i* gaps.)

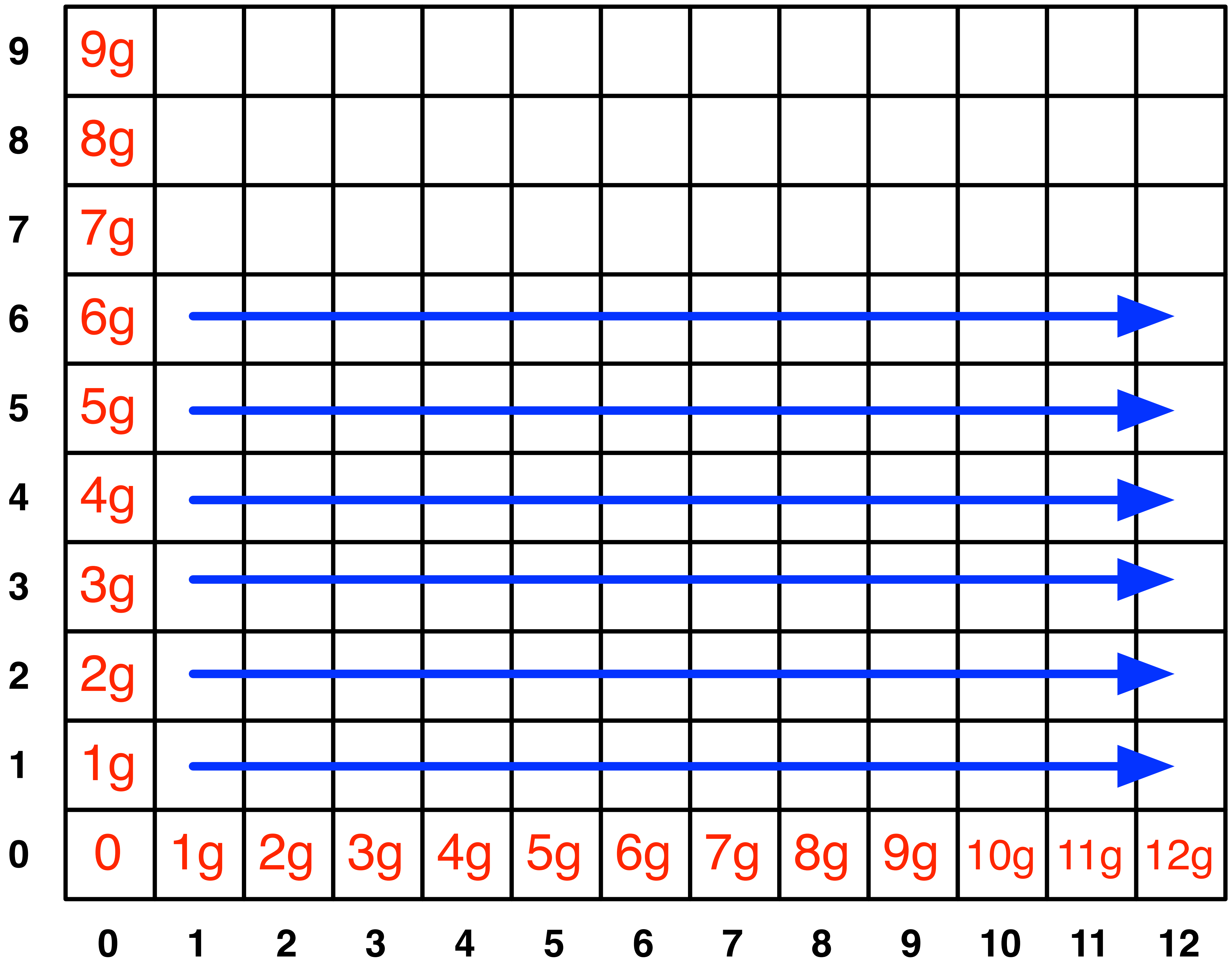
Computing OPT(i,j) Efficiently

We're ultimately interested in $OPT(n,m)$, but we will compute all other $OPT(i,j)$ $(i \le n, j \le m)$ on the way to computing $OPT(n,m)$.

Store those values in a 2D array:



Filling in the 2D Array



Edit Distance Computation

```
EditDistance(X,Y):
   For i = 1,...,m: A[i,0] = i*gap
   For j = 1,...,n: A[0,j] = j*gap

   For i = 1,...,m:
      For j = 1,...,n:
         A[i,j] = min(
            cost(a[i],b[j]) + A[i-1,j-1],
            gap + A[i-1,j],
            gap + A[i,j-1]
         )
      EndFor
   EndFor
   Return A[m,n]
```

Where's the answer?

OPT(n,m) contains the edit distance between the two strings.

Why? By induction: EVERY cell contains the optimal edit distance between some prefix of string 1 with some prefix of string 2.

Running Time

Number of entries in array $\approx m \times n$, where m and n are the lengths of the 2 strings.

Filling in each entry takes constant time.

Total running time is $\approx mn$.

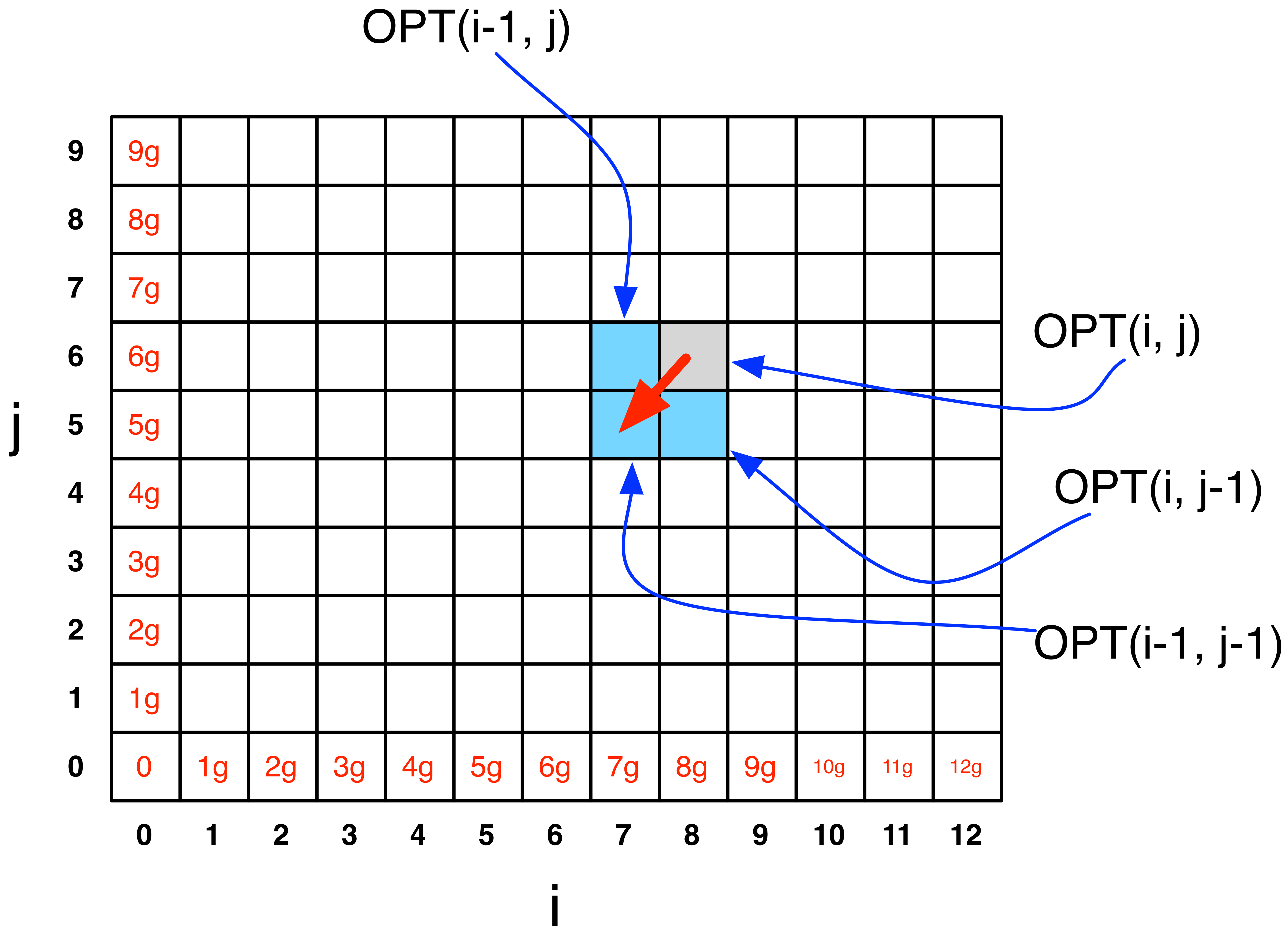
Alignment Python Code

```python
def local_align(x, y, score):
    “""Compute the score of an alignment between x and y”""

    # create a zero-filled matrix
    A = make_matrix(len(x) + 1, len(y) + 1)
    for i in xrange(0, len(x)):
        A[i][0] = i*score.gap
    for j in xrange(0, len(y)):
        A[0][j] = j*score.gap

    # fill in A in the right order
    for i in xrange(1, len(x)):
        for j in xrange(1, len(y)):

            if x[i] == y[j]:
                mm = score.mismatch
            else:
                mm = score.match

            # the local alignment recurrence rule:
            A[i][j] = max(
               A[i][j-1] + score.gap,
               A[i-1][j] + score.gap,
               A[i-1][j-1] + mm,
            )

    # return the opt score
    return A[len(x)-1][len(y)-1]
```

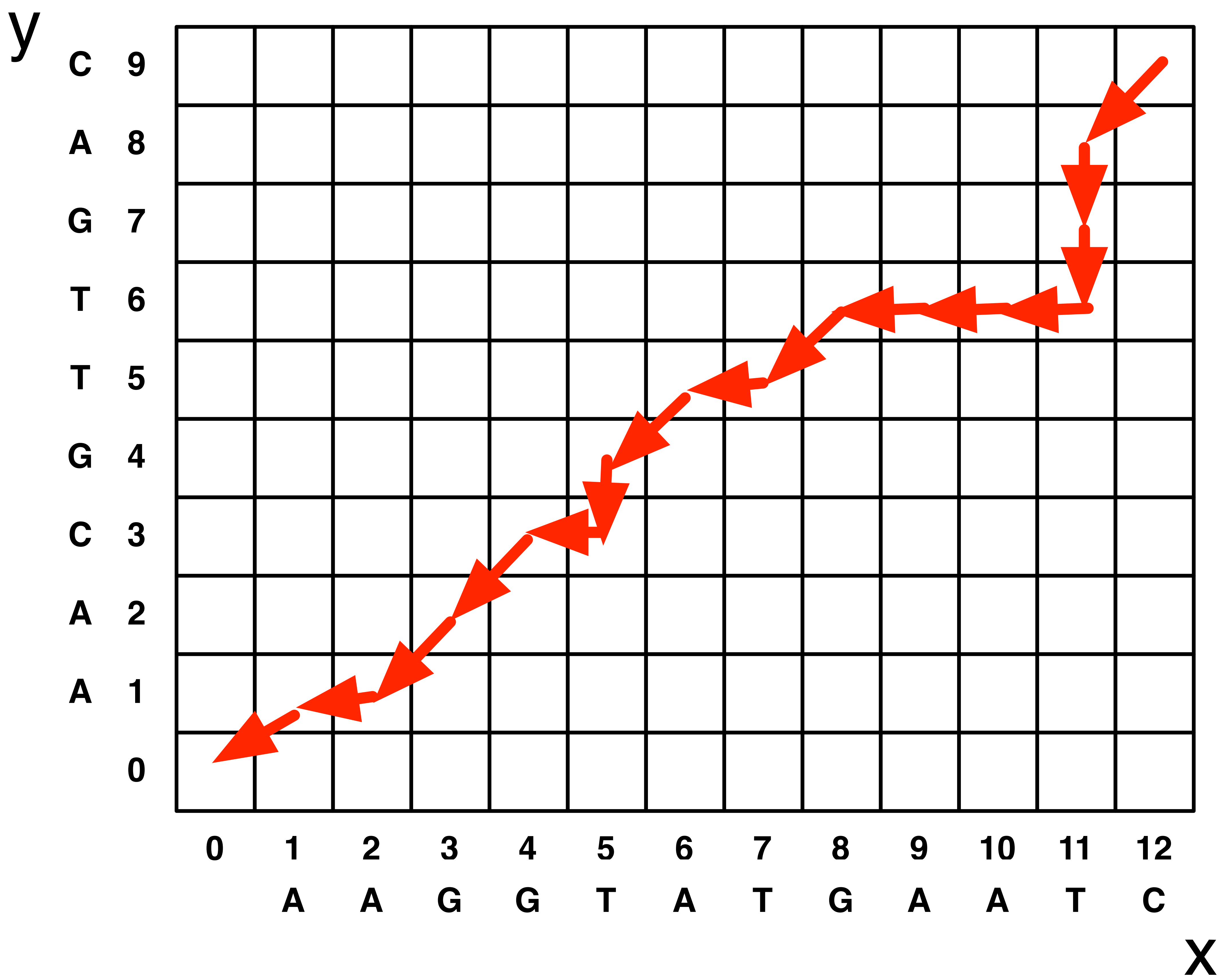
Local Alignment Python Code

```
def make_matrix(sizex, sizey):
    """Creates a sizex by sizey matrix filled with zeros."""
    return [[0]*sizey for i in xrange(sizex)]


class ScoreParam:
    """The parameters for an alignment scoring function"""
    def __init__(self, gap, match, mismatch):
        self.gap = gap
        self.match = match
        self.mismatch = mismatch
```

Finding the actual alignment



Trace the arrows all the way back



Outputting the Alignment

Build the alignment from right to left.

```
ACGT
A-GA
```

Follow the backtrack pointers starting from entry (n,m).

- If you follow a diagonal pointer, add both characters to the alignment,
- If you follow a left pointer, add a gap to the y-axis string and add the x-axis character
- If you follow a down pointer, add the y-axis character and add a gap to the x-axis string.