String Algorithms and Data Structures

Approximate Pattern Matching

CS 199-225

Brad Solomon

April 11, 2022

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

A_fmi reflection

All students responded 1-2 hours for time

The comments in the assignment / the breakdown of functions seemed reasonable

A_pigeon due today!

Last assignment today!

Remember only best 10 assignments are counted towards final grade

Next few lectures will be bonus material only

Approximate Pattern Matching

Input: A text *T*, a pattern *P*, and a distance *d*

Output: All positions in *T* where *P* has at most *d* mismatches or edits

P: word

T: There would have been a time for such a word

Alignment 1:
```
would
word
```

~~Not a match!~~

Distance 2 match!

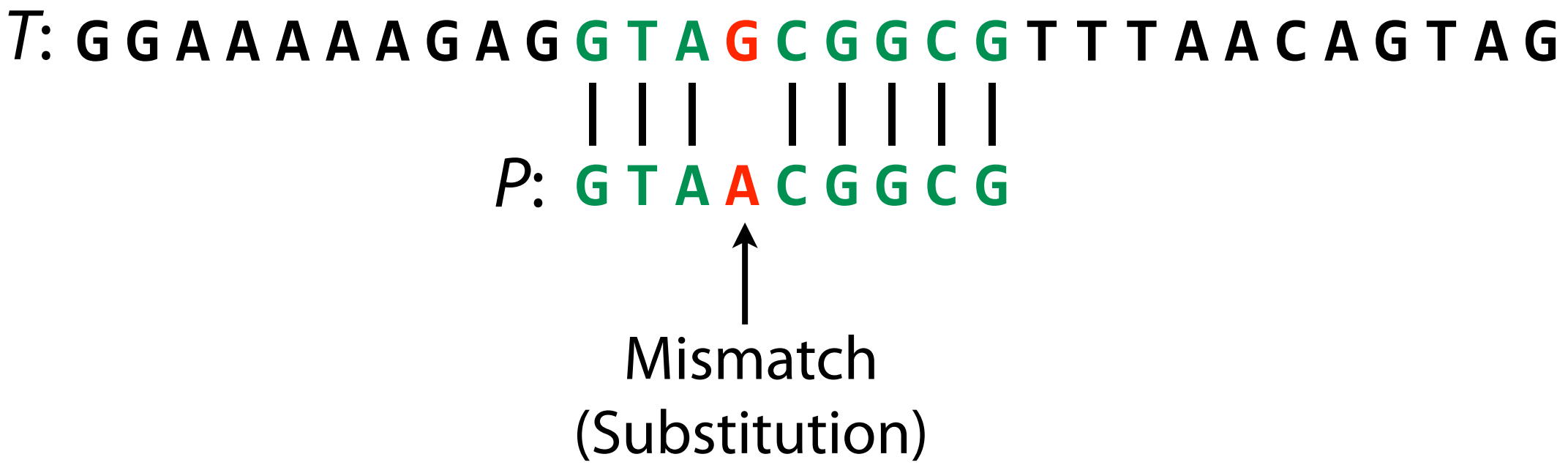
Alignment 2:
```
word
word
```

Match!

Distance 0 match!

Hamming Distance

The number of **substitutions** required to turn one string into another



Edit Distance

```
Score =  248 bits (129), Expect = 1e-63
Identities = 213/263 (80%), Gaps = 34/263 (12%)
Strand = Plus / Plus
```

Substitution

```
Query: 161 atatcaccacgtcaaaggtgactccaactcca---ccactccattttgttcagataatgc 217
           ||||||||||||||||||||||||||||  |     | |   || ||||||||||||||
Sbjct: 481 atatcaccacgtcaaaggtgactccaact-tattgatagtgtttttatgttcagataatgc 539

Query: 218 ccgatgatcatgtcatgcagctccaccgattgtgagaacgacagcgacttccgtcccagc 277
           |||||||   ||||||||||||||||||| || |           ||||||||||||
Sbjct: 540 ccgatgactttgtcatgcagctccaccgattttg-g----------ttccgtcccagc 586
```

Deletion

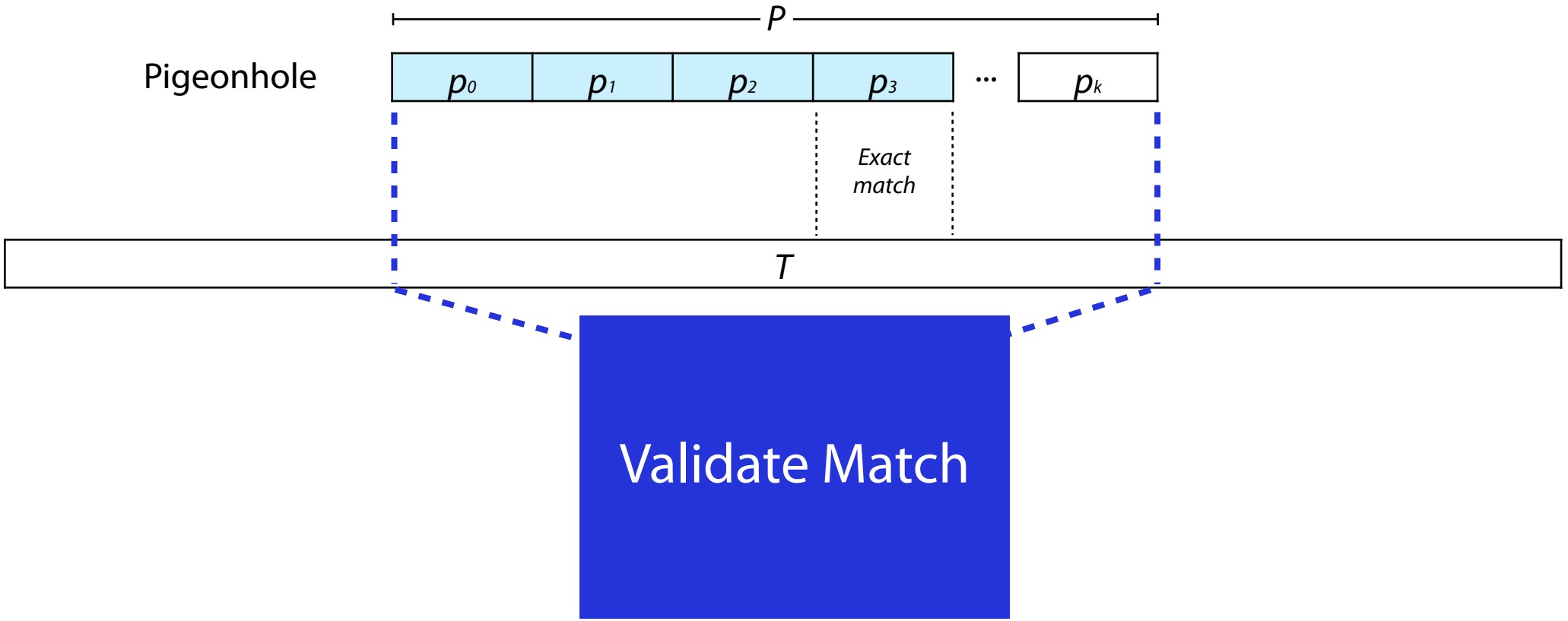
```
Query: 278 c-gtgcc--aggtgctgcctcagattcaggttatgccgctcaattcgctgcgtatatcgc 334
           |  ||  |  | ||||||||||||||||||||||||||||||||||||| |||||||||
Sbjct: 587 caatgacgta-gtgctgcctcagattcaggttatgccgctcaattcgctgggtatatcgc 645

Query: 335 ttgctgattacgtgcagctttcccttcaggcggga----------ccagccatccgtc 382
           ||||||||||||||||||||||||||||||||           ||||||||||||
Sbjct: 646 ttgctgattacgtgcagctttcccttcaggcgggattcatacagcggccagccatccgtc 705
```

Insertion

```
Query: 383 ctccatatc-accacgtcaaagg 404
           ||||||||  ||||||||||||
Sbjct: 706 atccatatcaaccacgtcaaagg 728
```

Approximate Pattern Matching

Find an exact match partition and validate the overall alignment



Learning Objectives

Review approximate pattern matching

Formalize edit distance storage as an ‘edit string’

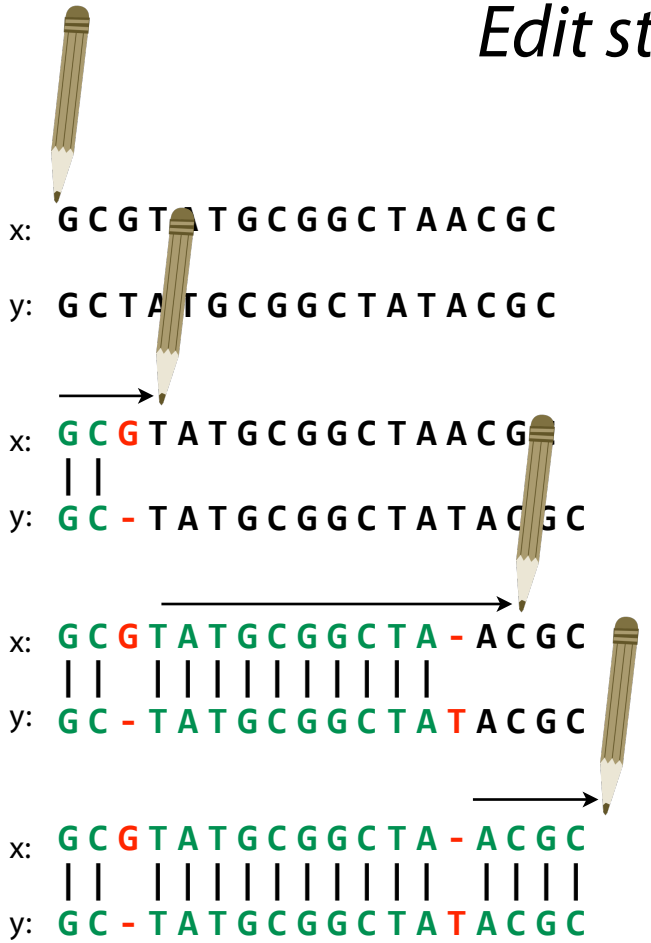
Discuss strategies for efficient APM with edits

Introduce dynamic programming

Edit Distance

Imagine edits are introduced by an *optimal editor* working left-to-right:

Edit string summarizes how editor turns *x* into *y:*



Operations:

M = match, **R** = replace (substitute),

I = insert into *x*, **D** = delete from *x*

MMD

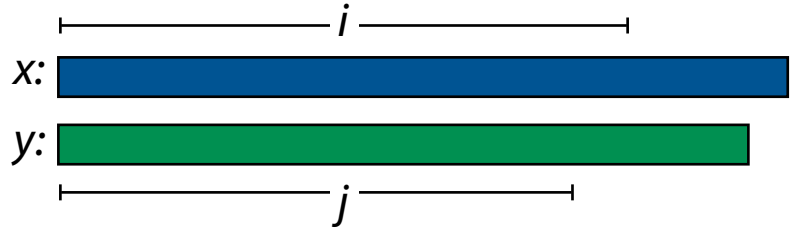
Reminder: this is **D**, not **I**, because we have to delete a character from *x* to make it more like *y*

MMDMMMMMMMMMI

MMDMMMMMMMMMIMMMM

Edit Distance

D[*i*, *j*]: edit distance between length-*i* prefix of *x* and length-*j* prefix of *y*



Optimal edit string for D[*i*, *j*] is built by ***extending a shorter optimal string by 1 operation***. 3 options:

Append **D** to transcript for D[*i*-1, *j*]

Append **I** to transcript for D[*i*, *j*-1]

Append **M** or **R** to transcript for D[*i*-1, *j*-1]

We choose based on whichever option has the fewest edits

Edit Distance

X: GTTTAA Y: GGTTTA

D[*5*, *6*]

```
GTTTA
GGTTTA
```

D[*5*, *5*]

```
GTTTA
GGTTT
```

D[*6*, *5*]

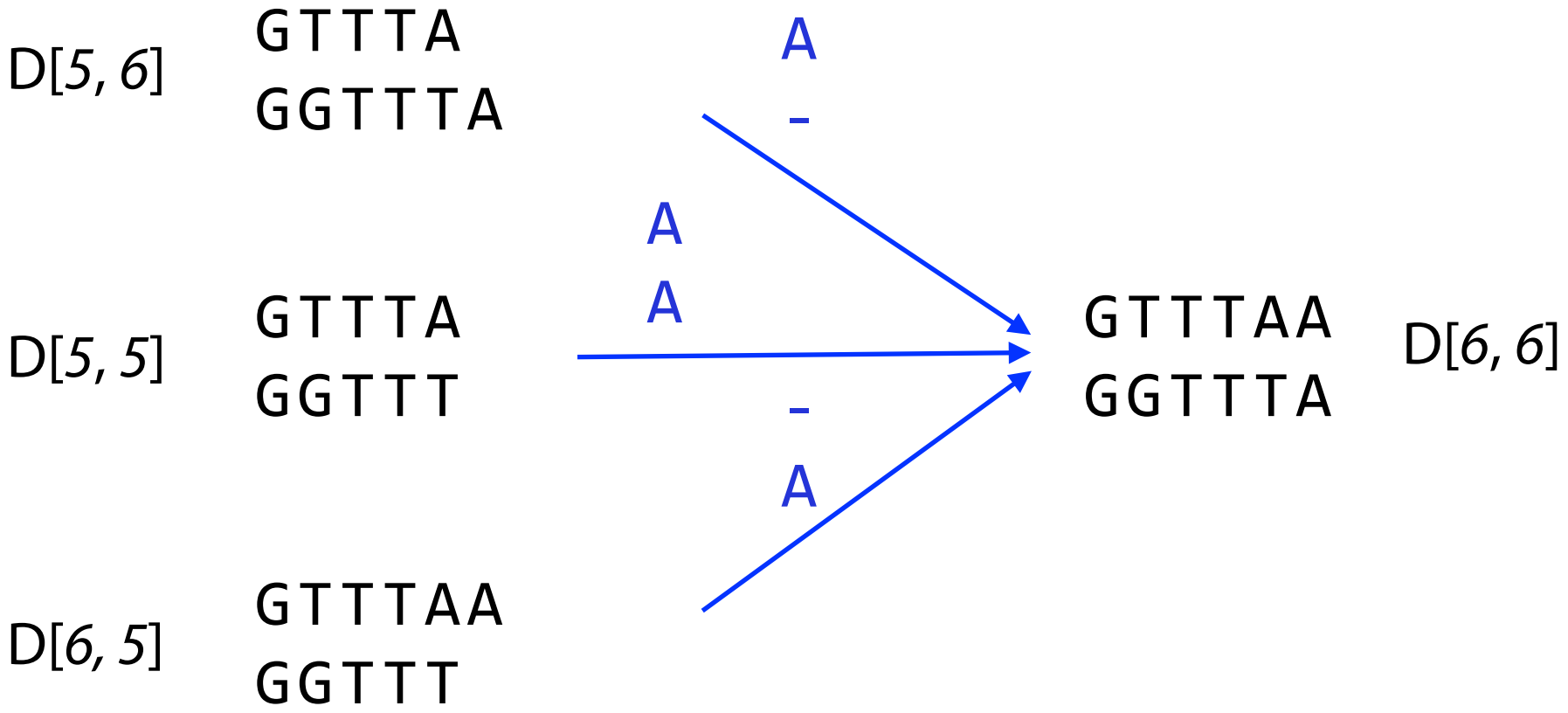
```
GTTTAA
GGTTT
```

```
GTTTAA
GGTTTA
```

D[*6*, *6*]

Edit Distance

X: GTTTAA Y: GGTTTA



Edit Distance

X: GTTTAA Y: GGTTTA

MIMMMM

D[*5*, *6*]

G	-	T	T	T	A
G	G	T	T	T	A

MRMMR

D[*5*, *5*]

G	T	T	T	A
G	G	T	T	T

MRMMRD

D[*6*, *5*]

G	T	T	T	A	A
G	G	T	T	T	-

Edit Distance

X: GTTTAA Y: GGTTTA

MIMMMM

D[*5*, *6*]

G	-	T	T	T	A
G	G	T	T	T	A

G	-	T	T	T	A	
G	G	T	T	T	A	

D[*6*, *6*][1]

MRMMR

D[*5*, *5*]

G	T	T	T	A
G	G	T	T	T

G	T	T	T	A	
G	G	T	T	T	

D[*6*, *6*][2]

MRMMRD

D[*6*, *5*]

G	T	T	T	A	A
G	G	T	T	T	-

G	T	T	T	A	A	
G	G	T	T	T	-	

D[*6*, *6*][3]

Edit Distance

X: GTTTAA Y: GGTTTA

D[*5*, *6*]

MIMMMM

G	-	T	T	T	A
G	G	T	T	T	A

MIMMMMD

G	-	T	T	T	A	A
G	G	T	T	T	A	-

D[*6*, *6*][1]

D[*5*, *5*]

MRMMR

G	T	T	T	A
G	G	T	T	T

MRMMRM

G	T	T	T	A	A
G	G	T	T	T	A

D[*6*, *6*][2]

D[*6*, *5*]

MRMMRD

G	T	T	T	A	A
G	G	T	T	T	-

MRMMRDI

G	T	T	T	A	A	-
G	G	T	T	T	-	A

D[*6*, *6*][3]

Edit Distance

X: ACCT Y: AGCC

D[*3*, *4*]

A	-	C	C	
A	G	C	C	

MIMM

D[*3*, *3*]

A	C	C	
A	G	C	

MRM

D[*4*, *3*]

A	C	C	T	
A	G	C	-	

MRMD

D[4,4] extends one of these options

D[*4*, *4*]

Edit Distance

X: ACCT Y: AGCC

D[*3*, *4*]

A	-	C	C	T
A	G	C	C	-

MIMMD

D[*3*, *3*]

A	C	C	T
A	G	C	C

MRMR

D[*4*, *3*]

A	C	C	T	-
A	G	C	-	C

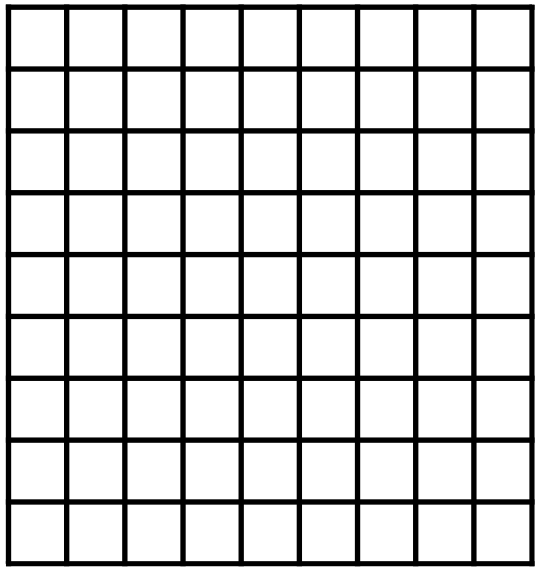
MRMDI

D[4,4] extends one of these options

D[*4*, *4*]

Edit Distance

We can store D as a 2D matrix:



Is at beginning

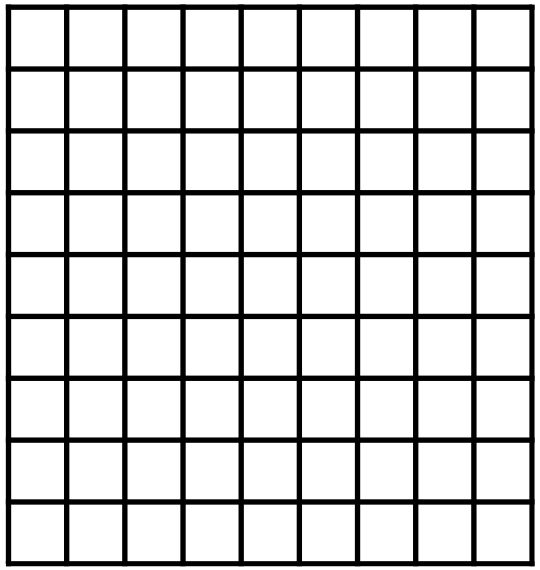
Ds at beginning

Let $D[0, j] = j$, and let $D[i, 0] = i$

```
----BBBBBBB
    |||||||
AAAABBBBBBB
```

```
the longest----
    |||||||
----longest day
```

Edit Distance



We can store D as a 2D matrix:

Is at beginning

Ds at beginning

Let $D[0, j] = j$, and let $D[i, 0] = i$

Otherwise, let $D[i, j] = \min$

$$\begin{cases} D[i-1, j] + 1 \\ D[i, j-1] + 1 \\ D[i-1, j-1] + \delta(x[i-1], y[j-1]) \end{cases}$$

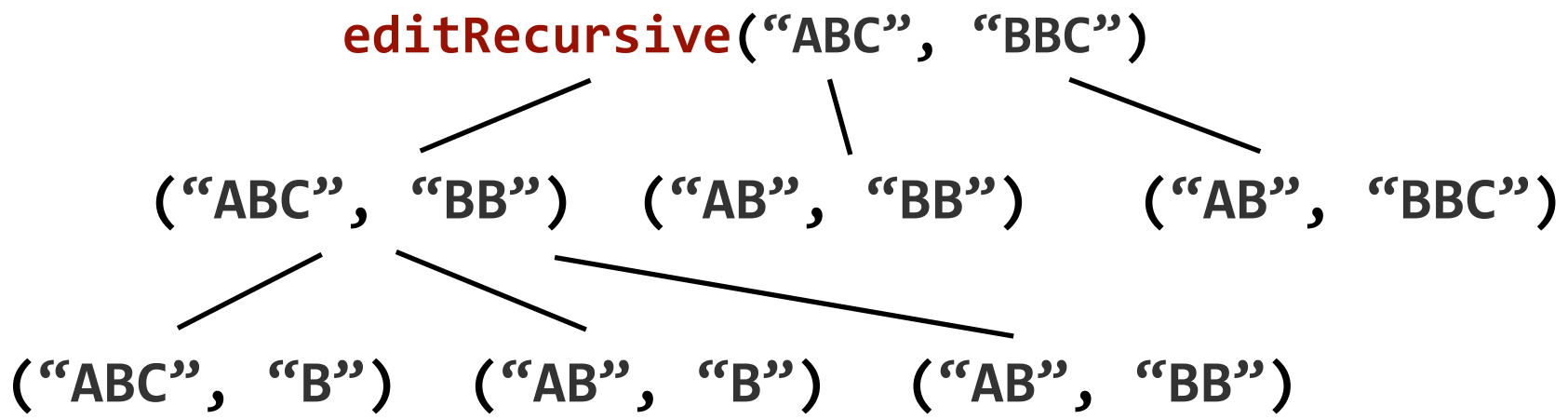
vertical (D)

horizontal (I)

diagonal (M or R)

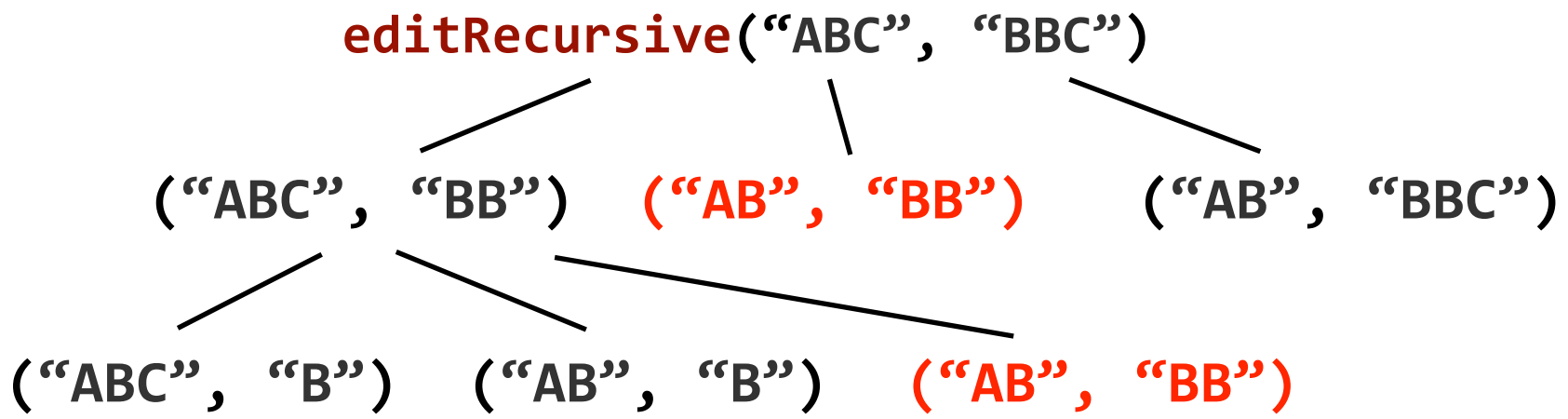
$\delta(a, b)$ is 0 if $a = b$, 1 otherwise

Edit Distance



(only part of recursion tree shown)

Edit Distance



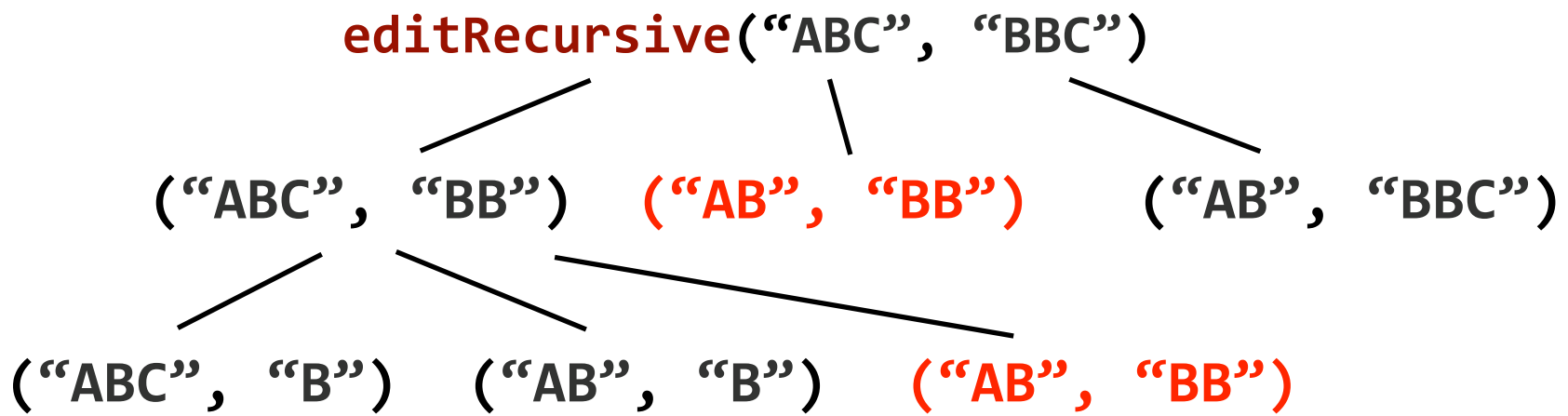
(only part of recursion tree shown)

`editRecursive("Shakespeare", "shake spear")`

`Calculate ("Shake", "shake") 8989 times`

How can we address this problem?

Edit Distance

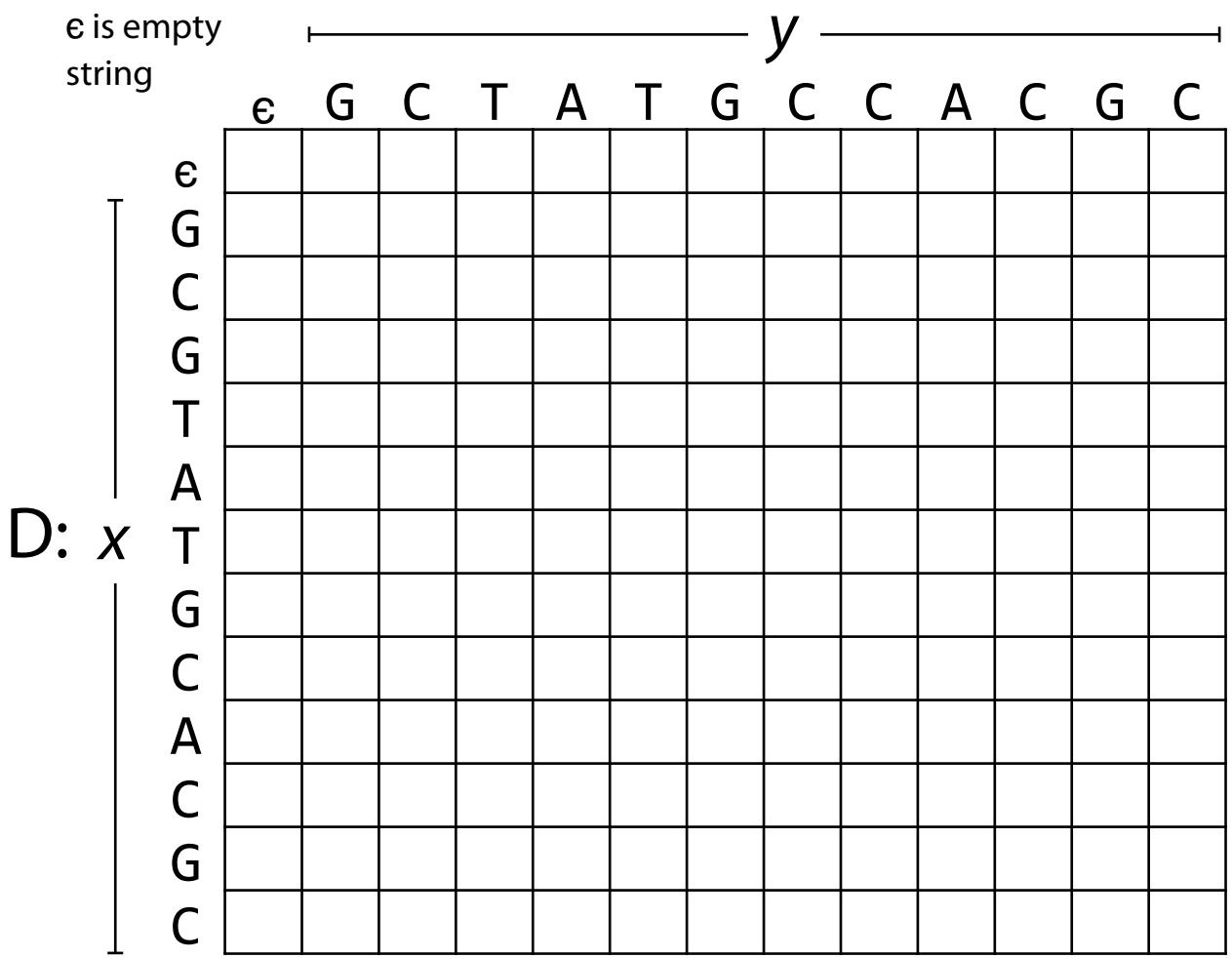


Memoization: Top-down

Dynamic Programming: Bottom-up

Both: Solve individual sub-problems once

Edit Distance: dynamic programming



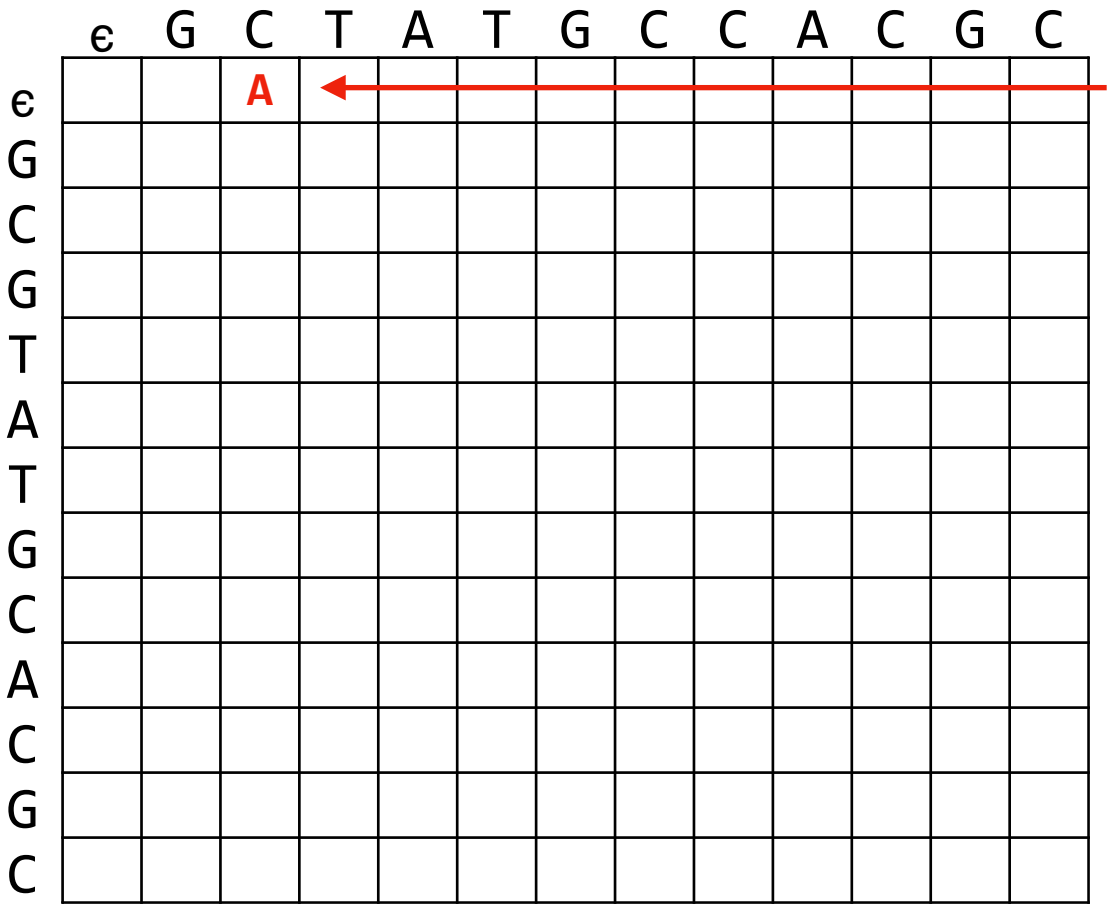
Let $n = |x|$, $m = |y|$

D: $(n+1)$ x $(m+1)$ matrix

D[i, j] = edit distance b/t length-i prefix of x and length-j prefix of y

Edit Distance: dynamic programming

Let $D[0, j] = j$, and let $D[i, 0] = i$

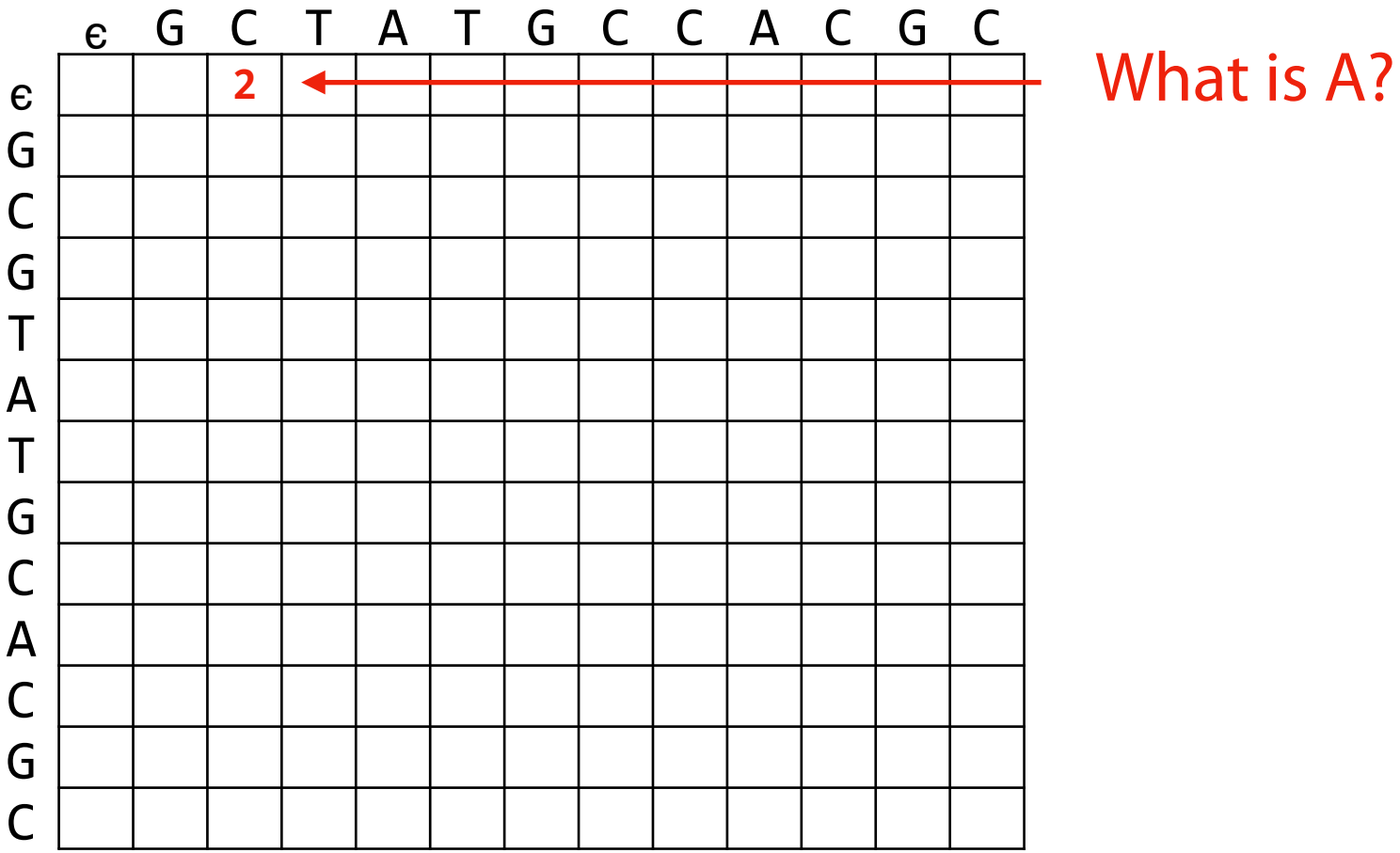


What is A?

D[*i*, *j*] = edit distance b/t length-*i* prefix of *x* and length-*j* prefix of *y*

Edit Distance: dynamic programming

Let $D[0, j] = j$, and let $D[i, 0] = i$



Edit Distance: dynamic programming

Let $D[0, j] = j$, and let $D[i, 0] = i$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1												
C	2												
G	3												
T	4												
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

Edit Distance: dynamic programming

Let $D[0, j] = j$, and let $D[i, 0] = i$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1												
C	2												
G	3												
T	4												
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

- - -

G C T

Edit Distance: dynamic programming

Let $D[0, j] = j$, and let $D[i, 0] = i$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1												
C	2												
G	3												
T	4												
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

G C G T A T

- - - - - -

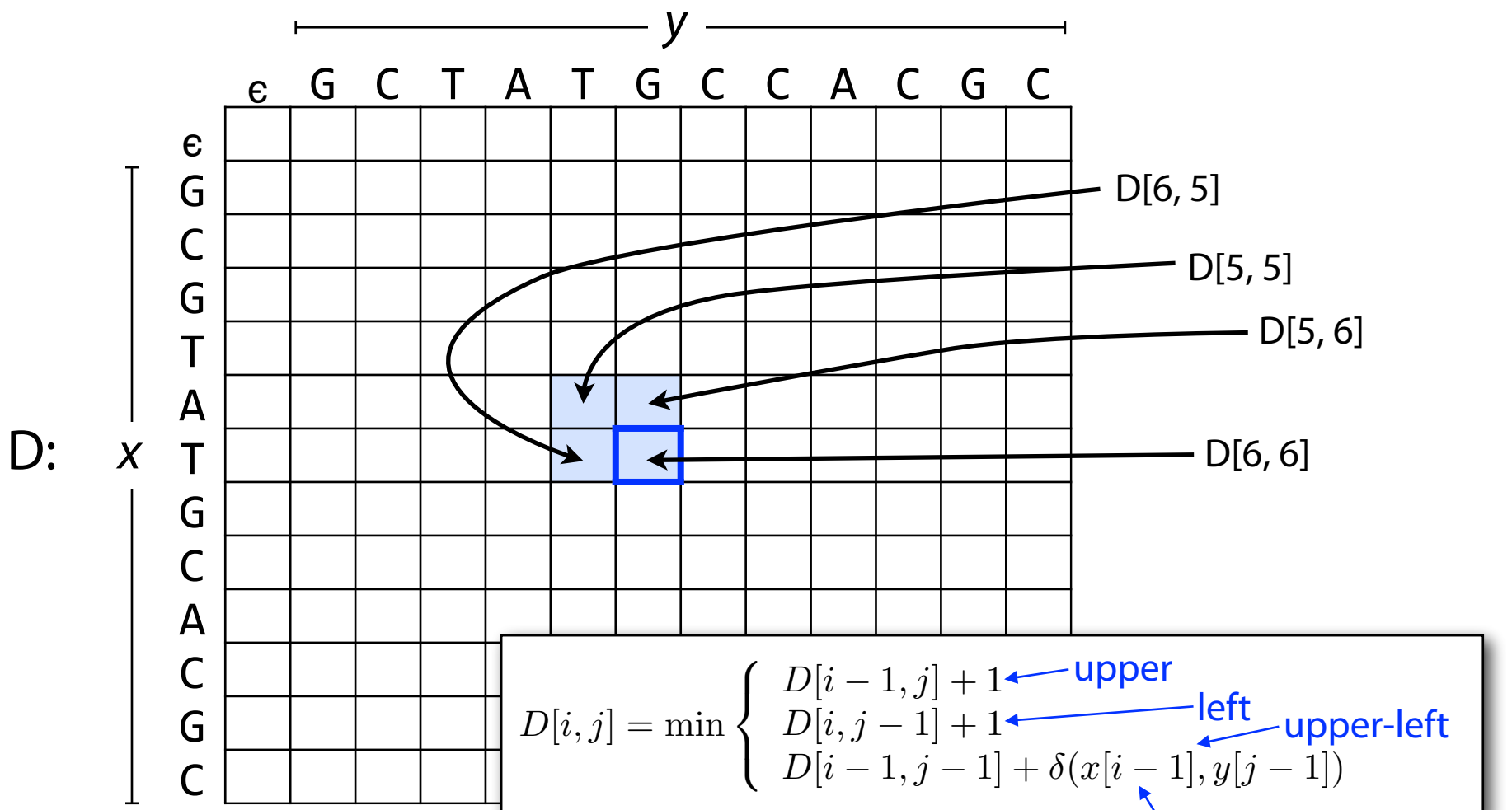
Edit Distance: dynamic programming

Let $D[0, j] = j$, and let $D[i, 0] = i$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
	0	1	2	3	4	5	6	7	8	9	10	11	12
ϵ	0												
G	1												
C	2												
G	3												
T	4												
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

D is one row / column larger than x / y!

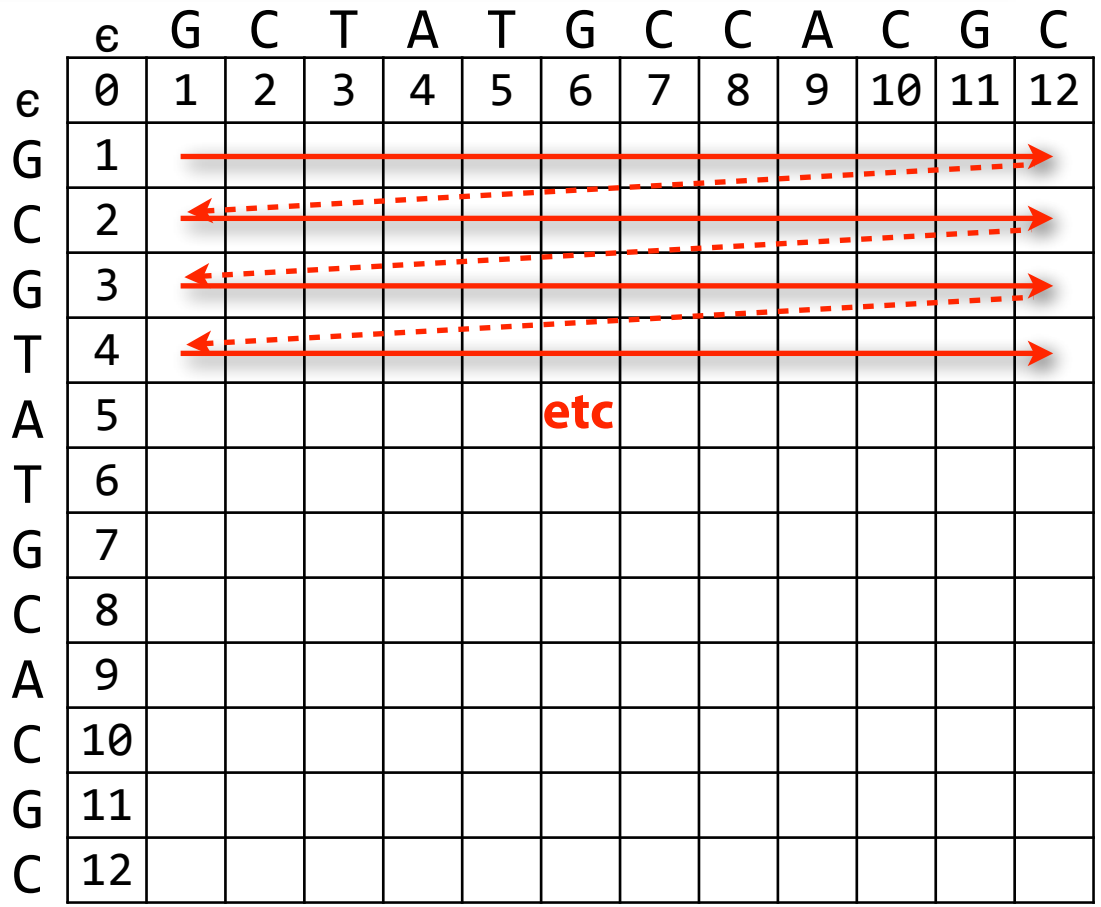
Edit Distance: dynamic programming



Cell depends upon its upper, left, and upper-left neighbors

Edit Distance: dynamic programming

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\delta(x[i-1],y[j-1]) \end{cases}$$



Fill remaining cells from top row to bottom and from left to right

Edit Distance: dynamic programming

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\delta(x[i-1],y[j-1]) \end{cases}$$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	?											
C	2												
G	3												
T	4												
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

What goes here in **i=1,j=1**?

x[i-1] = '**G**',
y[j-1] = '**G**',
so **delt = 0**

- G	G -	G |
G -	- G	G
ID	**DI**	**M**

```
D[i, j] = min(D[i-1, j]+1,
              D[i, j-1]+1,
              D[i-1, j-1]+delt)
        = min(1 + 1, 1 + 1, 0 + 0)
        = 0
```

Edit Distance: dynamic programming

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\delta(x[i-1],y[j-1]) \end{cases}$$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
	0	1	2	3	4	5	6	7	8	9	10	11	12
ϵ	0												
G	1	0	?										
C	2												
G	3												
T	4												
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

What goes here in **i=1, j=2**?

x[i-1] = 'G',
y[j-1] = 'C',
so **delt = 1**

```
D[i, j] = min(D[i-1, j]+1,
              D[i, j-1]+1,
              D[i-1, j-1]+delt)
        = min(2 + 1, 0 + 1, 1 + 1)
        = 1
```

- - G	G -	- G
G C -	G C	G C
IID	MI	IR

Edit Distance: dynamic programming

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\delta(x[i-1],y[j-1]) \end{cases}$$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6					
C	2	1	0	1	2	3	4	5					
G	3	2	1	1	2	3	3	4					
T	4	3	2	1	2	2	3	?					
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

What goes here in **i=4,j=7**?

Edit Distance: dynamic programming

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\delta(x[i-1],y[j-1]) \end{cases}$$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6					
C	2	1	0	1	2	3	4	5					
G	3	2	1	1	2	3	3	4					
T	4	3	2	1	2	2	3	4					
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

What goes here in **i=4,j=7**?

x[i-1] = '**T**',
y[j-1] = '**C**',
so **delt = 1**

```
G C - - - G T        G C - G T - -
G C T A T G C        G C T A T G C
M M I I I M R        M M I R M I I
```

```
D[i, j] = min(D[i-1, j]+1,
              D[i, j-1]+1,
              D[i-1, j-1]+delt)
        = min(4 + 1, 3 + 1, 3 + 1)
        = 4
```

Edit Distance: dynamic programming

$$D[i,j] = \min \begin{cases} D[i-1,j]+1 \\ D[i,j-1]+1 \\ D[i-1,j-1]+\delta(x[i-1],y[j-1]) \end{cases}$$

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

Fill remaining cells from top row to bottom and from left to right

← Edit distance for x, y

Edit Distance

		Y				
		ϵ	C	A	A	T
X	ϵ	0	1	2	3	4
	C	1	0	**A**	2	3
	A	2	1	0	**B**	
	T	3	2	**C**		

Edit Distance

		ϵ	C	A	A	T
		Y				
	ϵ	0	1	2	3	4
	C	1	0	**1**	2	3
X	A	2	1	0	**B**	
	T	3	2	**C**		

Edit Distance

		Y				
		ϵ	C	A	A	T
X	ϵ	0	1	2	3	4
	C	1	0	**1**	2	3
	A	2	1	0	**1**	
	T	3	2	**C**		

Edit Distance

		Y				
		ϵ	C	A	A	T
X	ϵ	0	1	2	3	4
	C	1	0	**1**	2	3
	A	2	1	0	**1**	
	T	3	2	**1**		

Edit Distance

		Y				
		ϵ	C	A	A	T
X	ϵ	0	1	2	3	4
	C	1	0	1	2	3
	A	2	1	0	1	2
	T	3	2	1	1	1

eMatrix buildEditMatrix(X, Y)

Input:

`string X:` Input string X (edits with respect to X)

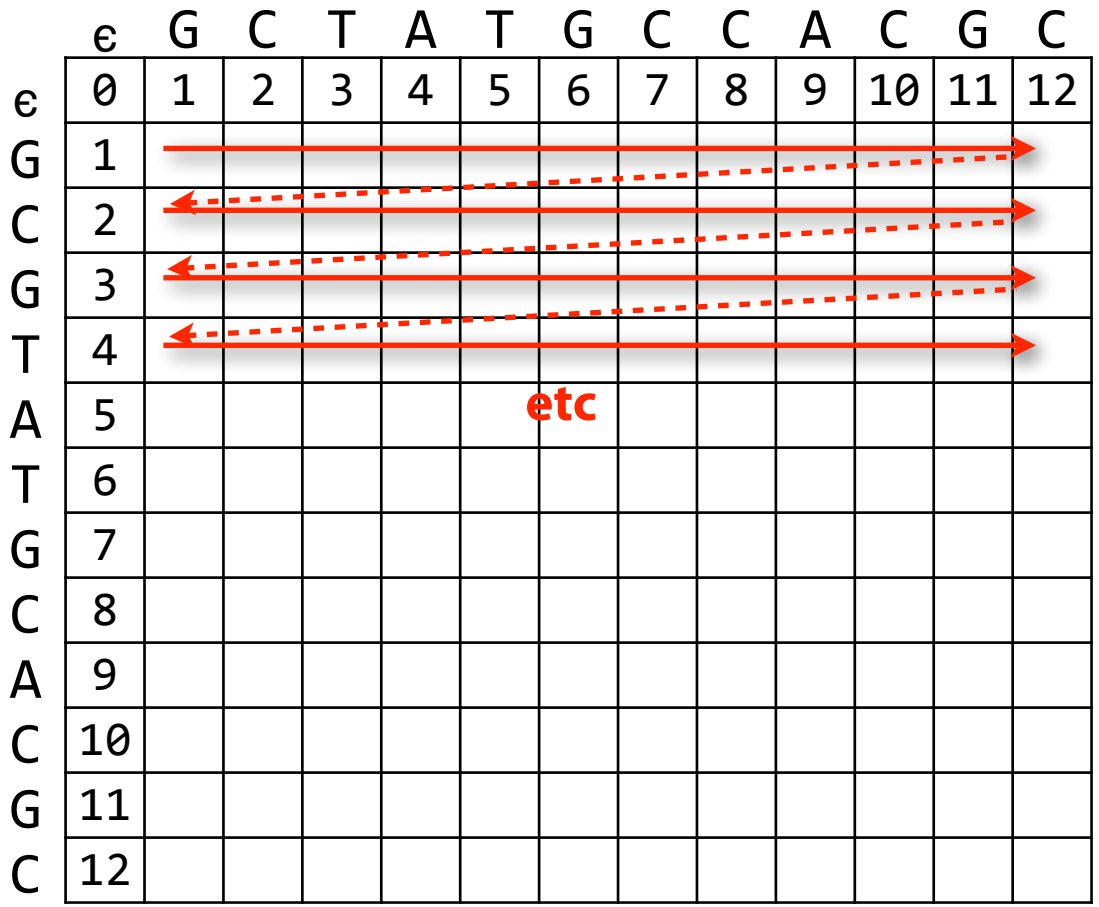
`string Y:` Input string Y (edits turn X into Y)

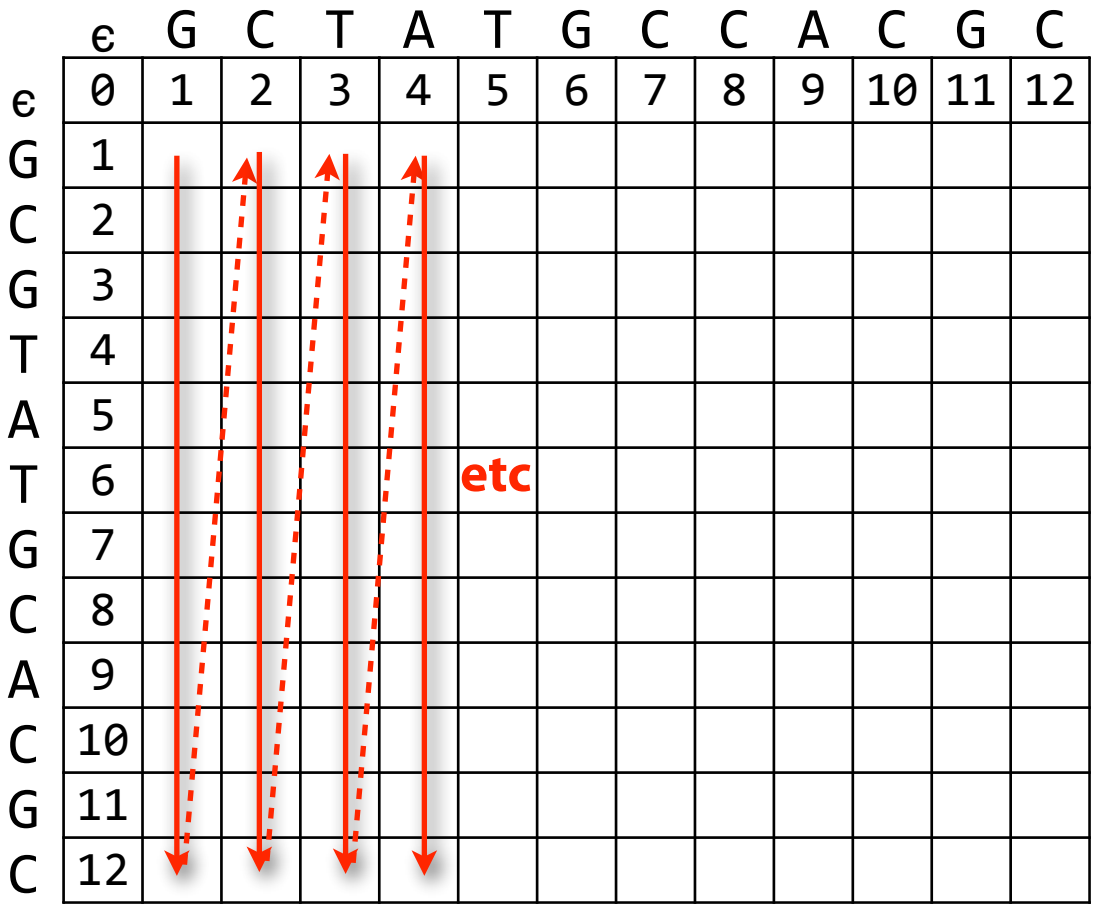
Output:

`eMatrix` : vector<vector<int>> storing all optimal edit distances

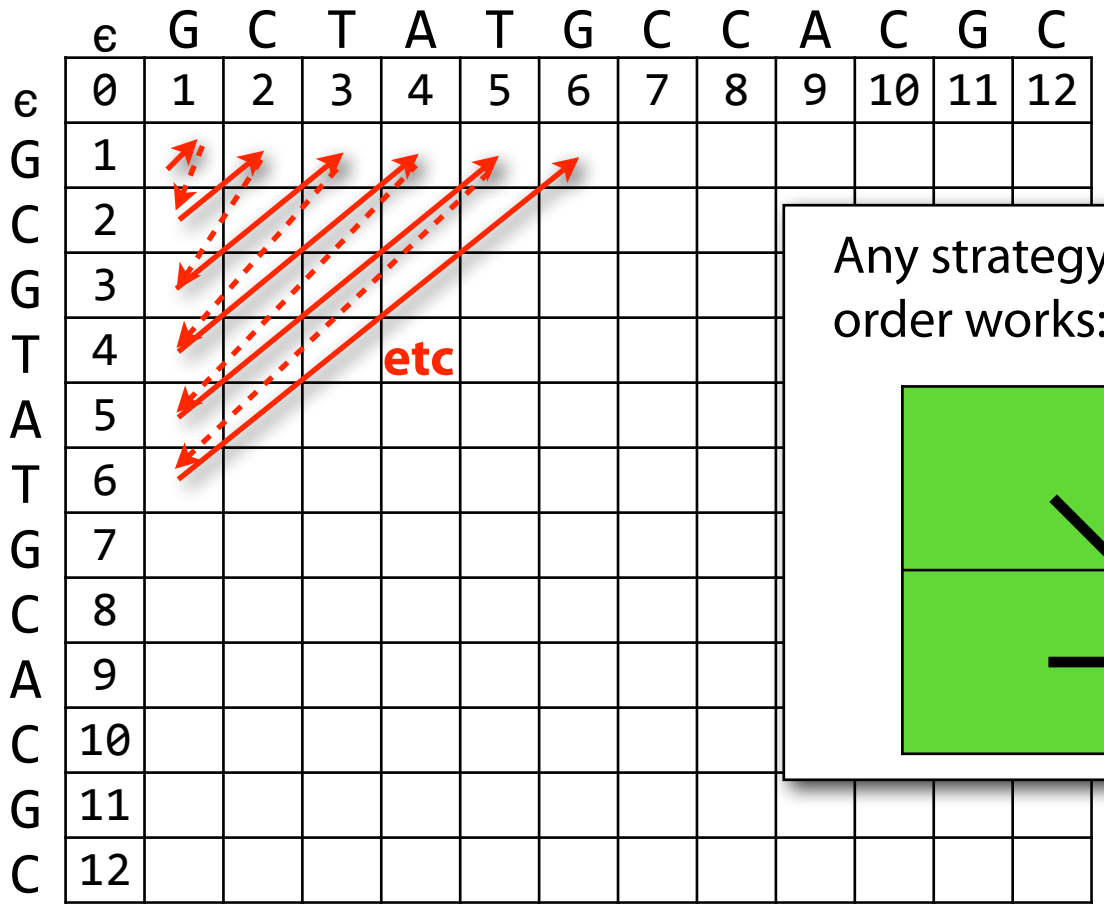
	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	1	2
T	3	2	1	1	1

Edit Distance: dynamic programming

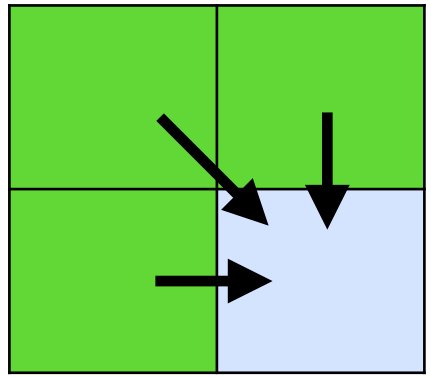


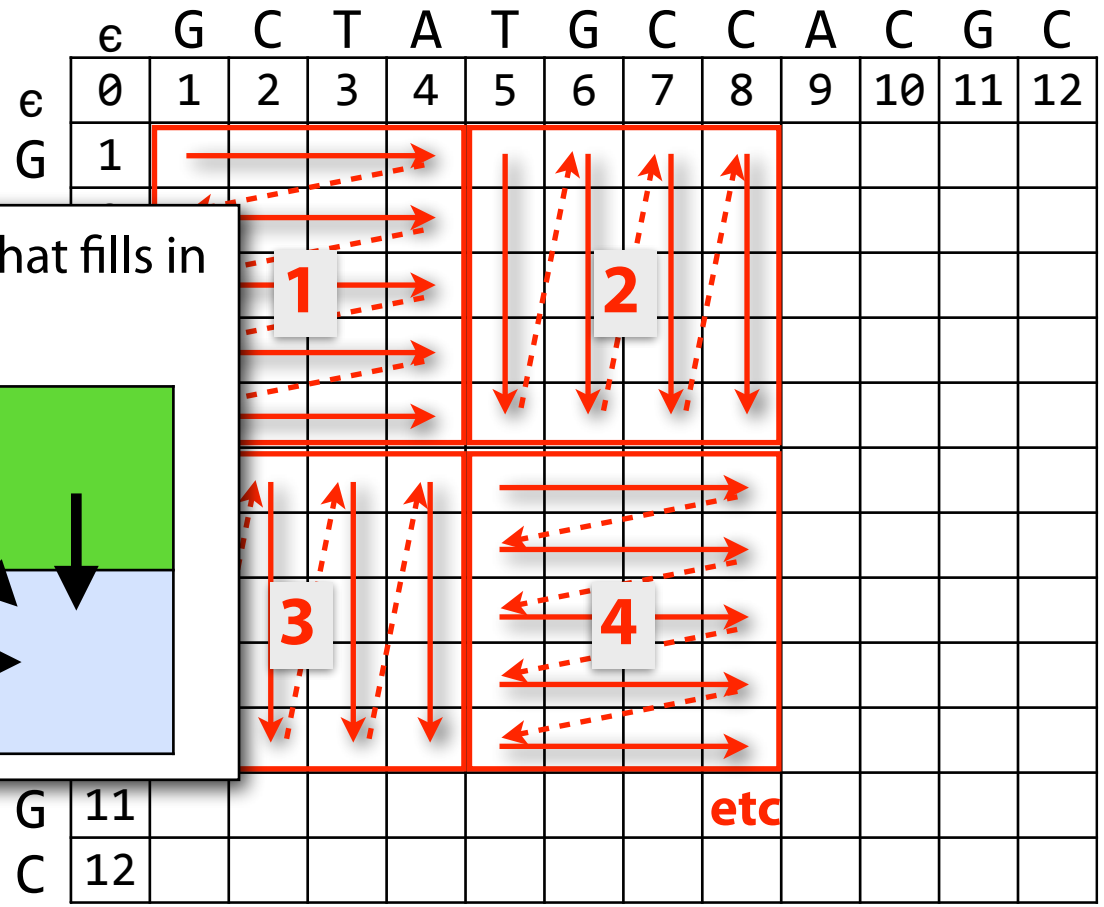


Edit Distance: dynamic programming



Any strategy that fills in order works:





Assignment 11: a_edist

Learning Objective:

Use dynamic programming to build an edit distance matrix

Construct an optimal edit string from the edit matrix

Consider: Does substitution, insertion, and deletion need to have the same 'weight' as a penalty? How could you modify the code to take a user-specified input for each?

Edit Distance

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	1	2
T	3	2	1	1	1

← Edit distance for x, y

But where and what is the edit?

Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	1	2
T	3	2	1	1	**1**

← Q: How did I get here?

Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	1	**2**
T	3	2	1	1	1

$D[2,\ 4] =$ _____

← Q: How did I get here?

Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	1	2
T	3	2	1	1	1

$D[3,\ 3] =$ _____

← Q: How did I get here?

Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	**1**	2
T	3	2	1	1	1

$D[2,\ \ 3] =$ _____

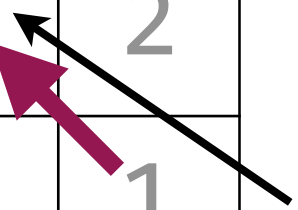
Q: How did I get here?

Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	1	2
T	3	2	1	1	1



Q: How did I get here?

$D[1,\ 3] =$ _____

$D[1,\ 2] =$ _____

$D[2,\ 2] =$ _____

Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	**1**	2	
A	2	1	**0**	**1**	2
T	3	2	1	1	1

Tie!

$$D[1,\ 3] = 2 + 1$$

$$D[1,\ 2] = 1 + 0$$

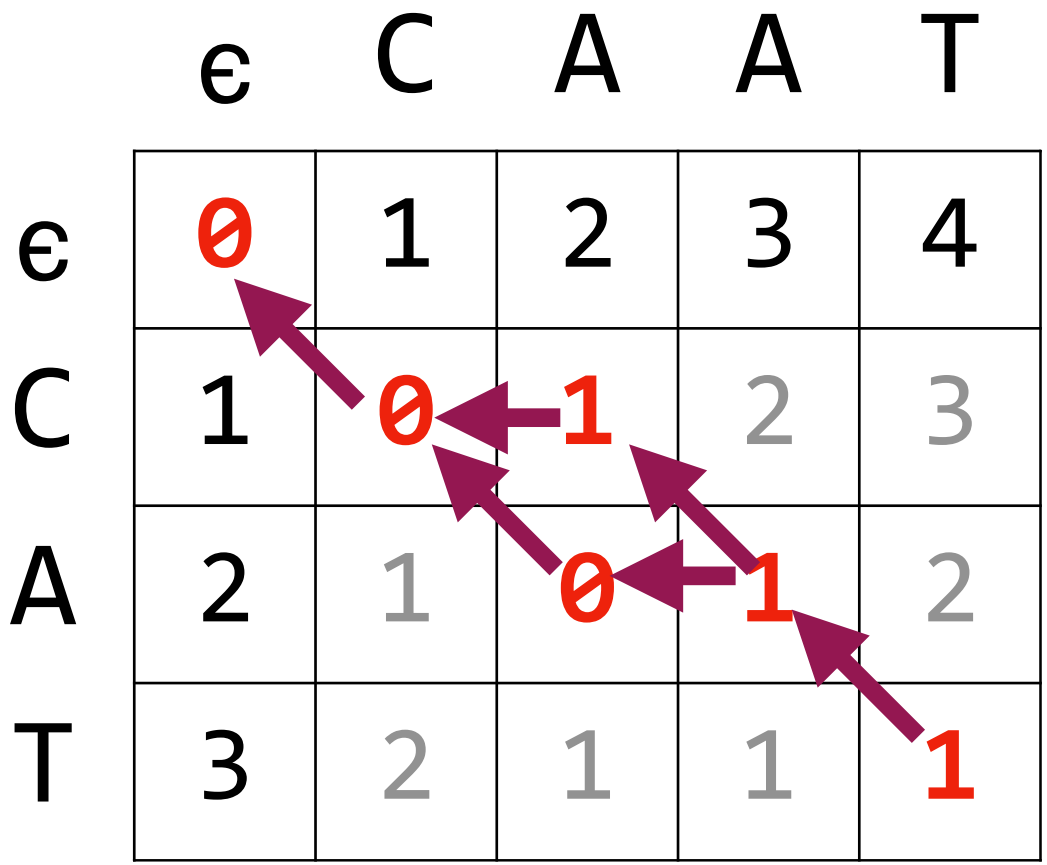
$$D[2,\ 2] = 0 + 1$$

Q: How did I get here?

Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

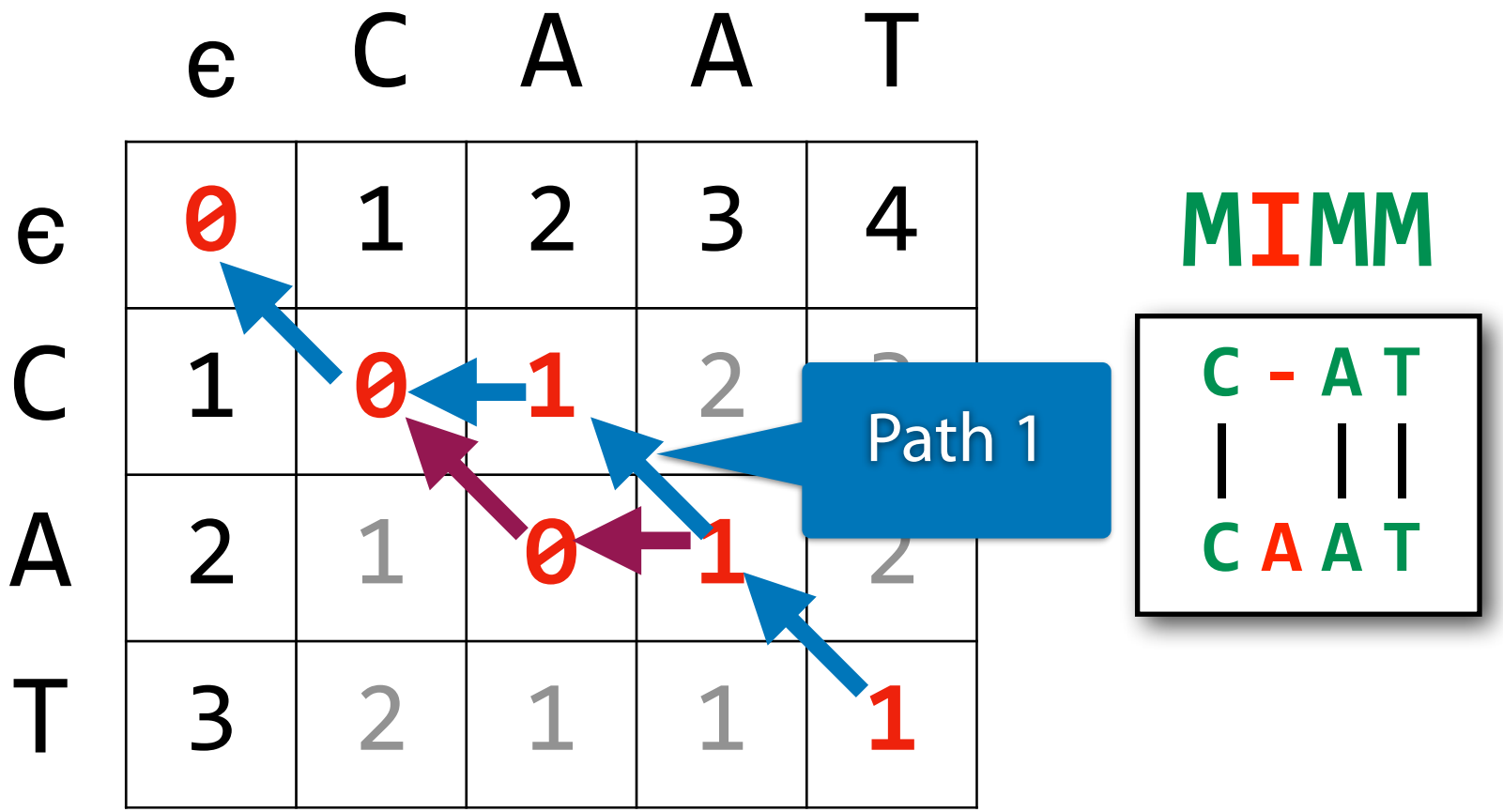
At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?



Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

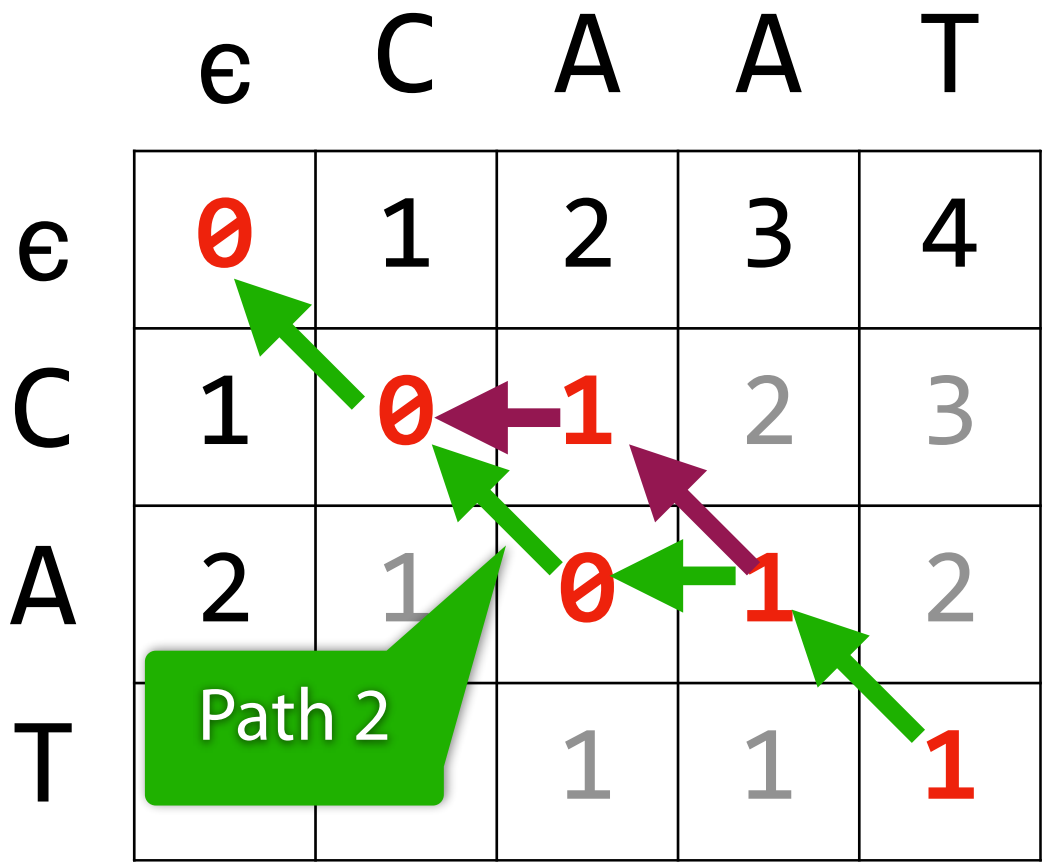
At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?



Edit Distance

Traceback corresponds to an optimal alignment / edit transcript

At each step, ask: which neighbor (⇖, ⇐ or ⇑) gave the minimum?



MIMM

```
C - A T
|   | |
C A A T
```

MMIM

```
C A - T
| |   |
C A A T
```

Edit Distance

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

Q: How did I get here?

Edit Distance

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

$$D[11,\ 12] = 3 + 1$$

$$D[11,\ 11] = 2 + 0$$

$$D[12,\ 11] = 3 + 1$$

A: From here

Q: How did I get here?

Edit Distance

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

Q: How did I get here?

Edit Distance

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

$$D[10,\ 11] = 3 + 1$$

$$D[10,\ 10] = 2 + 0$$

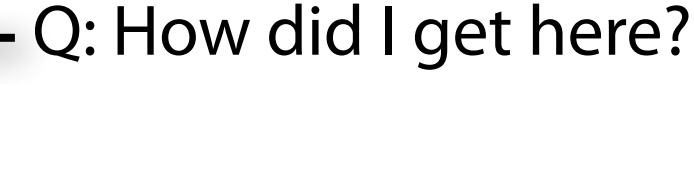
$$D[11,\ 10] = 3 + 1$$

A: From here

Q: How did I get here?

Edit Distance

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2



Edit Distance

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

$$D[9,\ 10] = 3 + 1$$

$$D[9,\ 9] = 2 + 0$$

$$D[10,\ 9] = 3 + 1$$

A: From here

Q: How did I get here?

Edit Distance

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

Alignment:

```
G C G T A T G - C A C G C
|| |  |||| |  |||||
G C - T A T G C C A C G C
```

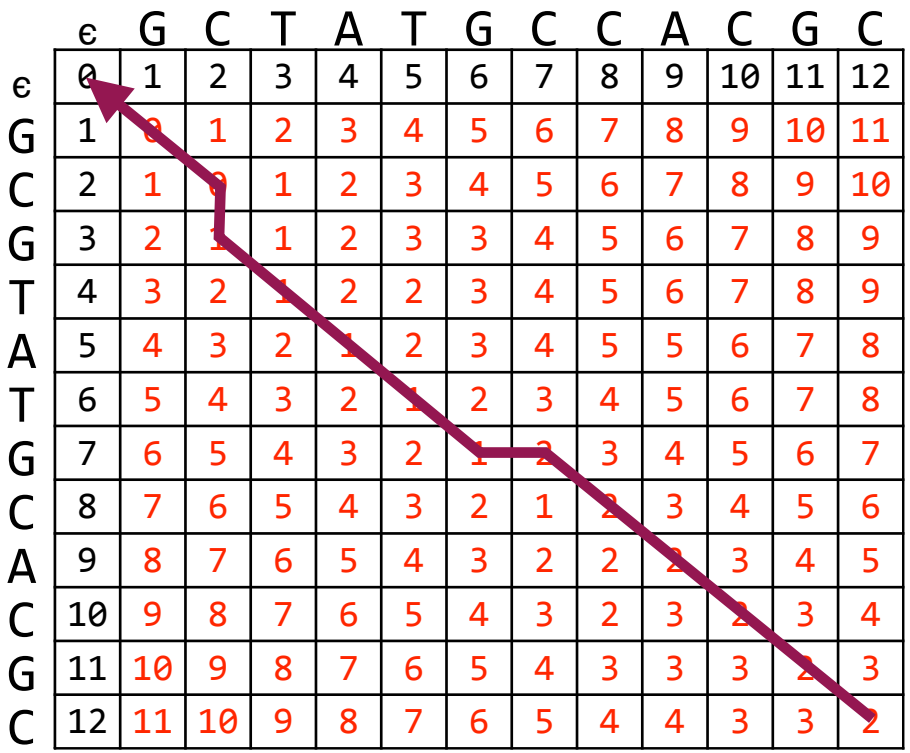
```
MMDMMMMIMMMMM
```

Edit Distance

Dynamic Programming fills our table with optimal distances

Traceback identifies the optimal *edit string* to convert *X* to *Y*

What is our efficiency? ($|X| = m, |Y| = n$)



	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

Edit Distance

Dynamic Programming fills our table with optimal distances

Traceback identifies the optimal *edit string* to convert *X* to *Y*

	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ	0	1	2	3	4	5	6	7	8	9	10	11	12
G	1	0	1	2	3	4	5	6	7	8	9	10	11
C	2	1	0	1	2	3	4	5	6	7	8	9	10
G	3	2	1	1	2	3	3	4	5	6	7	8	9
T	4	3	2	1	2	2	3	4	5	6	7	8	9
A	5	4	3	2	1	2	3	4	5	5	6	7	8
T	6	5	4	3	2	1	2	3	4	5	6	7	8
G	7	6	5	4	3	2	1	2	3	4	5	6	7
C	8	7	6	5	4	3	2	1	2	3	4	5	6
A	9	8	7	6	5	4	3	2	2	2	3	4	5
C	10	9	8	7	6	5	4	3	2	3	2	3	4
G	11	10	9	8	7	6	5	4	3	3	3	2	3
C	12	11	10	9	8	7	6	5	4	4	3	3	2

What is our efficiency? ($|X| = m$, $|Y| = n$)

Table filling: Filling (m + 1) (n + 1) cells, each requiring constant work, so $O(mn)$

Traceback: Each step goes ⇖, ⇐ or ⇑. Worst case: traceback never moves diagonally, requiring m ⇑'s and n ⇐'s, so $O(m + n)$

`string buildEditString(X, Y)`

Input: **`string X:`** Input string X (edits with respect to X)

`string Y:` Input string Y (edits turn X into Y)

Output: **`string`**: An optimal edit string produced by the matrix

	ϵ	C	A	A	T
ϵ	0	1	2	3	4
C	1	0	1	2	3
A	2	1	0	1	2
T	3	2	1	1	1

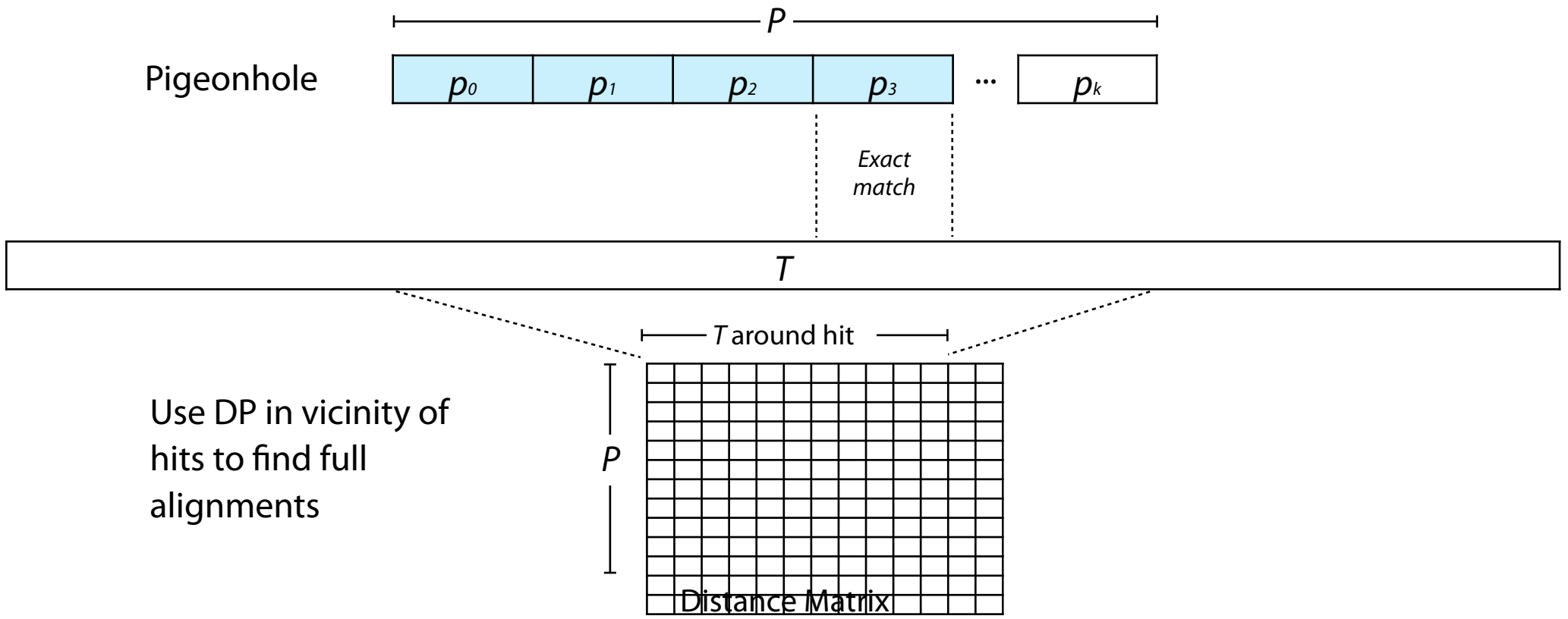
On tie: prioritize diagonal, then vertical, then horizontal

MIMM

```
C - A T
|   | |
C A A T
```

Approximate Pattern Matching

"Seed and extend" works for edit distance too!



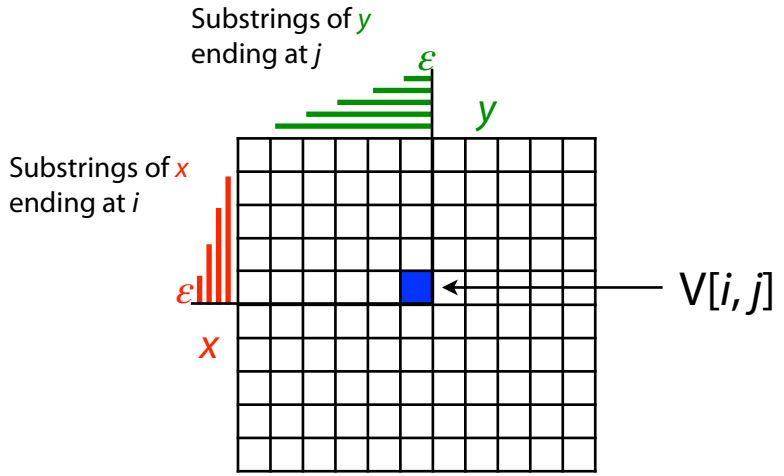
Bonus Slide

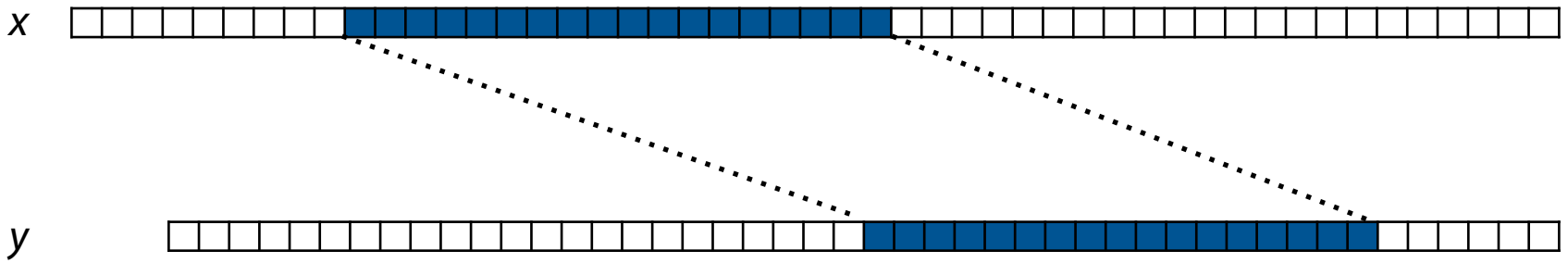
$s(a, b)$

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

$s(a, b)$

	A	C	G	T	-
A	1	-1	-1	-1	-1
C	-1	1	-1	-1	-1
G	-1	-1	1	-1	-1
T	-1	-1	-1	1	-1
-	-1	-1	-1	-1	





… and much more!