Dynamic programming and edit distance

Ben Langmead

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Department of Computer Science

Beyond approximate matching: sequence similarity

In many settings, Hamming and edit distance are too simple. Biologically-relevant distances require algorithms. We will expand our tool set accordingly.

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

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

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

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

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

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

Example BLAST alignment

Approximate string matching

A *mismatch* is a single-character substitution:

```
X:  G T A G C G G C G
    | | |   | | | | |
Y:  G T A A C G G C G
```

An *edit* is a single-character substitution or *gap* (*insertion* or *deletion*):

```
X:  G T A G C G G C G
    | | |   | | | | |
Y:  G T A A C G G C G
```

Gap in X

```
X:  G T A G C - G C G
    | | | | |   | | |
Y:  G T A G C G G C G
```

AKA *insertion* in *Y* or *deletion* in *X*

```
X:  G T A G C G G C G
    | |   | | | | | |
Y:  G T - G C G G C G
```

AKA *insertion* in *X* or *deletion* in *Y*

Gap in Y

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Alignment

```
X: G C G T A T G A G G C T A - A C G C
   | |   | | | |   | | | | |   | | | |
Y: G C - T A T G C G G C T A T A C G C
```

Above is an *alignment*: a way of lining up the characters of *x* and *y*

Could include mismatches, gaps or both

Vertical lines are drawn where opposite characters match

Hamming and edit distance

Finding Hamming distance between 2 strings is easy:

```
def hammingDistance(x, y):
    assert len(x) == len(y)
    nmm = 0
    for i in xrange(0, len(x)):
        if x[i] != y[i]:
            nmm += 1
    return nmm
```

```
GAGGTAGCGGCGTTTAAC
|  |||||  |||  |||||||
GTGGTAACGGGGTTTAAC
```

Edit distance is harder:

```
def editDistance(x, y):
    ???
```

```
GCGTATGCGGCTA-ACGC
||  |||||||||||  |||
GC-TATGCGGCTATACGC
```

Edit distance

```
def editDistance(x, y):
    return ???
```

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

If strings *x* and *y* are same length, what can we say about **editDistance**(*x*, *y*) relative to **hammingDistance**(*x*, *y*)?

$$\texttt{editDistance}(x, y) \leq \texttt{hammingDistance}(x, y)$$

If strings *x* and *y* are different lengths, what can we say about **editDistance**(*x*, *y*)?

$$\texttt{editDistance}(x, y) \geq ||x| - |y||$$

Python example: http://bit.ly/CG_DP_EditDist

Edit distance

Can think of edits as being introduced by an *optimal editor* working left-to-right.
Edit transcript describes how editor turns *x* into *y*.

```
x: G C G T A T G C G G C T A A C G C

y: G C T A T G C G G C T A T A C G C

x: G C G T A T G C G G C T A A C G C
   | |
y: G C - T A T G C G G C T A T A C G C

x: G C G T A T G C G G C T A - A C G C
   | |   | | | | | | | | | |
y: G C - T A T G C G G C T A T A C G C

x: G C G T A T G C G G C T A - A C G C
   | |   | | | | | | | | | |   | | | |
y: G C - T A T G C G G C T A T A C G C
```

Operations:
M = match, **R** = replace,
I = insert into *x*, **D** = delete from *x*

MMD

MMDMMMMMMMMMMI

MMDMMMMMMMMMMIMMMM

Edit distance

Alignments:

Edit transcripts with respect to *x*:

```
x: G C G T A T G C G G C T A - A C G C
   | |   | | | | | | | | | |   | | | |
y: G C - T A T G C G G C T A T A C G C
```

MMDMMMMMMMMMMMIMMMM

Distance = 2

```
x: G C G T A T G A G G C T A - A C G C
   | |   | | | |   | | | | |   | | | |
y: G C - T A T G C G G C T A T A C G C
```

MMDMMMMRMMMMMMIMMMM

Distance = 3

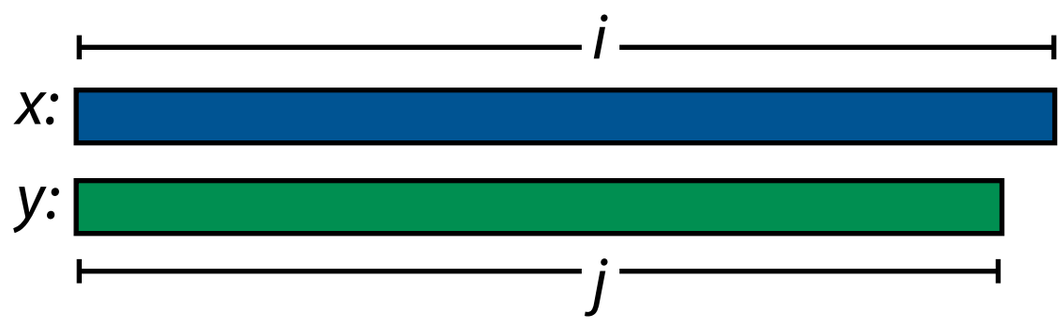
```
x: the longest----
        |||||||
y: ----longest day
```

DDDDMMMMMMMMIIII

Distance = 8

Edit distance

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



Think in terms of edit transcript. Optimal transcript for D[*i*, *j*] can be built by extending a shorter one by 1 operation. Only 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]

D[*i*, *j*] is minimum of the three, and D[|*x*|, |*y*|] is the overall edit distance

Edit distance

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

Otherwise, let $D[i, j] = \min \begin{cases} D[i-1, j] + 1 & \text{D} \\ D[i, j-1] + 1 & \text{I} \\ D[i-1, j-1] + \delta(x[i-1], y[j-1]) & \text{M or R} \end{cases}$

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

Edit distance

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

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

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

A simple recursive algorithm:

prefixes of *x* and *y* currently under consideration

```
def edDistRecursive(x, y):
    if len(x) == 0: return len(y)
    if len(y) == 0: return len(x)
    delt = 1 if x[-1] != y[-1] else 0
    diag = edDistRecursive(x[:-1], y[:-1]) + delt
    vert = edDistRecursive(x[:-1], y) + 1
    horz = edDistRecursive(x, y[:-1]) + 1
    return min(diag, vert, horz)
```

Recursively solve smaller problems

Python example: http://bit.ly/CG_DP_EditDist

Edit distance

```
def edDistRecursive(x, y):
    if len(x) == 0: return len(y)
    if len(y) == 0: return len(x)
    delt = 1 if x[-1] != y[-1] else 0
    diag = edDistRecursive(x[:-1], y[:-1]) + delt
    vert = edDistRecursive(x[:-1], y) + 1
    horz = edDistRecursive(x, y[:-1]) + 1
    return min(diag, vert, horz)
```

```
>>> import datetime as d
>>> st = d.datetime.now(); \
... edDistRecursive("Shakespeare", "shake spear"); \
... print (d.datetime.now()-st).total_seconds()
3
31.498284
```

Simple, but takes >30 seconds for a small problem

Edit distance: dynamic programming

Subproblems (D[*i*, *j*]s) can be reused instead of being recalculated:

```
def edDistRecursive(x, y):
    if len(x) == 0: return len(y)
    if len(y) == 0: return len(x)
    delt = 1 if x[-1] != y[-1] else 0
    diag = edDistRecursive(x[:-1], y[:-1]) + delt
    vert = edDistRecursive(x[:-1], y) + 1
    horz = edDistRecursive(x, y[:-1]) + 1
    return min(diag, vert, horz)
```

Reusing solutions to subproblems is *memoization:*

Return memoized answer, if avaialable

Memoize D[*i*, *j*]

```
def edDistRecursiveMemo(x, y, memo=None):
    if memo is None: memo = {}
    if len(x) == 0: return len(y)
    if len(y) == 0: return len(x)
    if (len(x), len(y)) in memo:
        return memo[(len(x), len(y))]
    delt = 1 if x[-1] != y[-1] else 0
    diag = edDistRecursiveMemo(x[:-1], y[:-1], memo) + delt
    vert = edDistRecursiveMemo(x[:-1], y, memo) + 1
    horz = edDistRecursiveMemo(x, y[:-1], memo) + 1
    ans = min(diag, vert, horz)
    memo[(len(x), len(y))] = ans
    return ans
```

Python example: http://bit.ly/CG_DP_EditDist

Edit distance: dynamic programming

```
def edDistRecursiveMemo(x, y, memo=None):
    if memo is None: memo = {}
    if len(x) == 0: return len(y)
    if len(y) == 0: return len(x)
    if (len(x), len(y)) in memo:
        return memo[(len(x), len(y))]
    delt = 1 if x[-1] != y[-1] else 0
    diag = edDistRecursiveMemo(x[:-1], y[:-1], memo) + delt
    vert = edDistRecursiveMemo(x[:-1], y, memo) + 1
    horz = edDistRecursiveMemo(x, y[:-1], memo) + 1
    ans = min(diag, vert, horz)
    memo[(len(x), len(y))] = ans
    return ans
```

```
>>> import datetime as d
>>> st = d.datetime.now(); \
... edDistRecursiveMemo("Shakespeare", "shake spear"); \
... print (d.datetime.now()-st).total_seconds()
3
0.000593
```

Much better

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Edit distance: dynamic programming

`edDistRecursiveMemo` is a *top-down* dynamic programming approach

Alternative is *bottom-up*. Here, bottom-up recursion is pretty intuitive and interpretable, so this is how edit distance algorithm is usually explained.

Fills in a table (matrix) of D(*i*, *j*)s:

```
import numpy
```

`numpy`: package for matrices, etc

```
def edDistDp(x, y):
    """ Calculate edit distance between sequences x and y using
        matrix dynamic programming.  Return distance. """
    D = numpy.zeros((len(x)+1, len(y)+1), dtype=int)
    D[0, 1:] = range(1, len(y)+1)
    D[1:, 0] = range(1, len(x)+1)
    for i in xrange(1, len(x)+1):
        for j in xrange(1, len(y)+1):
            delt = 1 if x[i-1] != y[j-1] else 0
            D[i, j] = min(D[i-1, j-1]+delt, D[i-1, j]+1, D[i, j-1]+1)
    return D[len(x), len(y)]
```

Fill 1st row, col

Fill rest of matrix

Edit distance: dynamic programming

ϵ is empty string

D: x — y

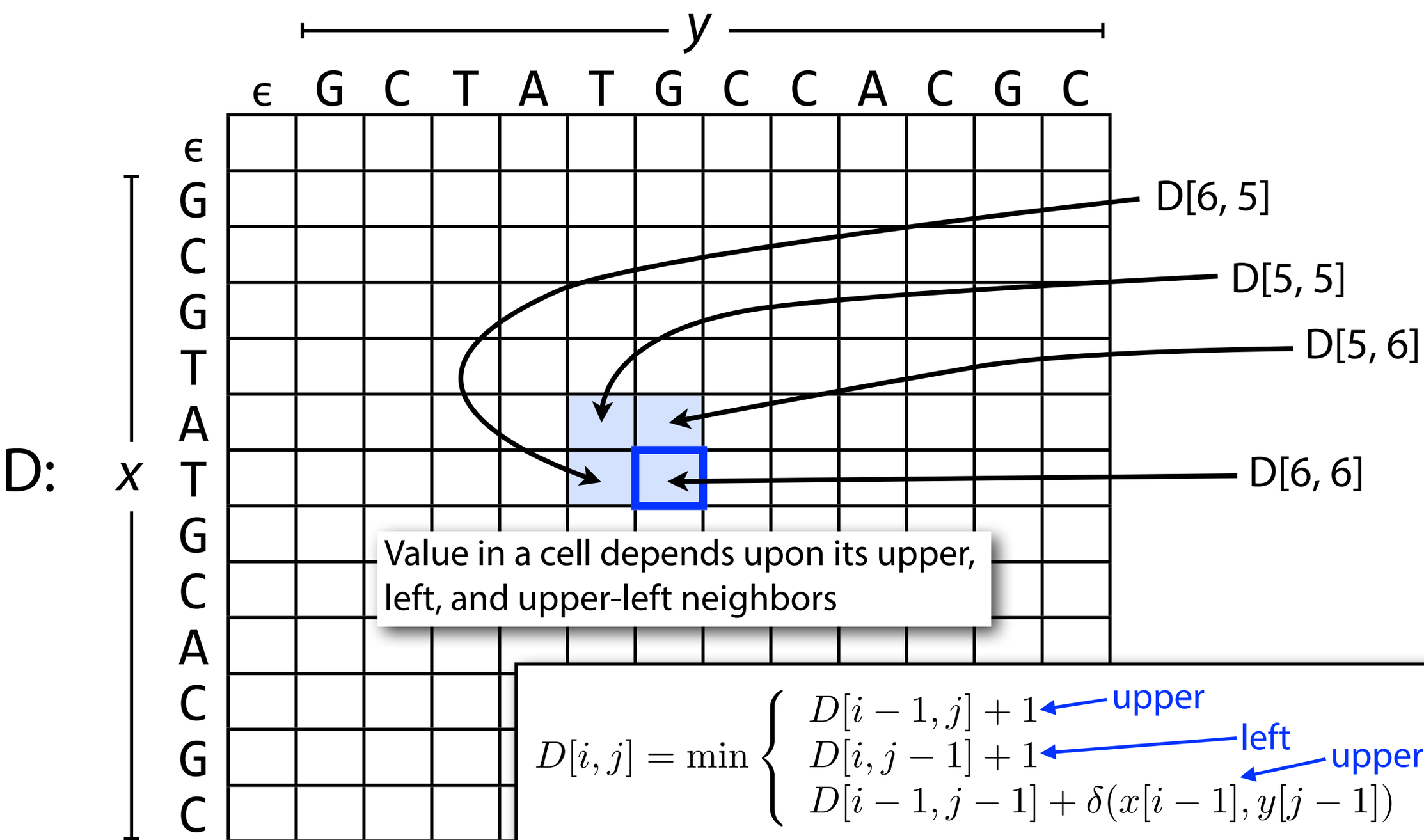
	ϵ	G	C	T	A	T	G	C	C	A	C	G	C
ϵ													
G													
C													
G													
T													
A													
T													
G													
C													
A													
C													
G													
C													

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

D: $(n+1) \times (m+1)$ matrix

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

Edit distance: dynamic programming



Value in a cell depends upon its upper, left, and upper-left neighbors

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

Edit distance: dynamic programming

First few lines of `edDistDp`:

```
D = numpy.zeros((len(x)+1, len(y)+1), dtype=int)
D[0, 1:] = range(1, len(y)+1)
D[1:, 0] = range(1, len(x)+1)
```

	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												

Initialize D[0, *j*] to *j*, D[*i*, 0] to *i*

Edit distance: dynamic programming

Loop from **edDistDp**:

```
for i in xrange(1, len(x)+1):
    for j in xrange(1, len(y)+1):
        delt = 1 if x[i-1] != y[j-1] else 0
        D[i, j] = min(D[i-1, j-1]+delt, D[i-1, j]+1, D[i, j-1]+1)
```

	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						etc						
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

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

Edit distance: dynamic programming

Loop from **edDistDp**:

```
for i in xrange(1, len(x)+1):
    for j in xrange(1, len(y)+1):
        delt = 1 if x[i-1] != y[j-1] else 0
        D[i, j] = min(D[i-1, j-1]+delt, D[i-1, j]+1, D[i, j-1]+1)
```

	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												

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

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

`x[i-1]` = `y[j-1]` = '`G`',

SO `delt = 0`

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

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Edit distance: dynamic programming

Loop from **edDistDp**:

```
for i in xrange(1, len(x)+1):
    for j in xrange(1, len(y)+1):
        delt = 1 if x[i-1] != y[j-1] else 0
        D[i, j] = min(D[i-1, j-1]+delt, D[i-1, j]+1, D[i, j-1]+1)
```

	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: dynamic programming

Loop from **edDistDp**:

```
for i in xrange(1, len(x)+1):
    for j in xrange(1, len(y)+1):
        delt = 1 if x[i-1] != y[j-1] else 0
        D[i, j] = min(D[i-1, j-1]+delt, D[i-1, j]+1, D[i, j-1]+1)
```

	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						etc						
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

Could we have filled the cells in a different order?

Edit distance: dynamic programming

Switched

```
for j in xrange(1, len(y)+1):
    for i in xrange(1, len(x)+1):
        delt = 1 if x[i-1] != y[j-1] else 0
        D[i, j] = min(D[i-1, j-1]+delt, D[i-1, j]+1, D[i, j-1]+1)
```

	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					etc							
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

Yes: e.g. invert the loops

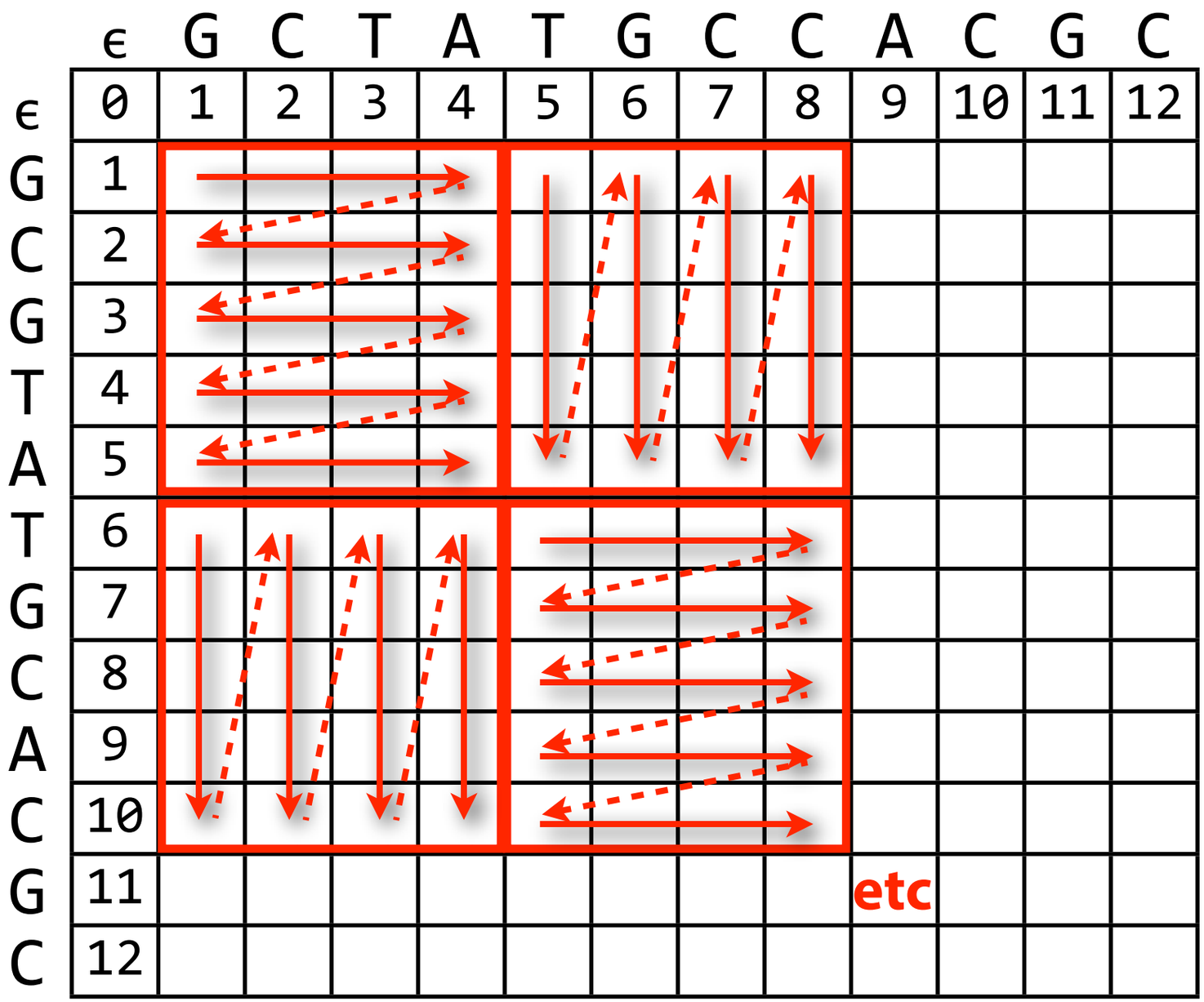
JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Edit distance: dynamic programming

	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				etc								
A	5												
T	6												
G	7												
C	8												
A	9												
C	10												
G	11												
C	12												

Or by anti-diagonal

Edit distance: dynamic programming



Or blocked

Edit distance: getting the alignment

Full backtrace path corresponds to an optimal alignment / edit transcript:

Start at end; at each step, ask: which predecessor gave the minimum?

	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

A: From here

Q: How did I get here?

Edit distance: getting the alignment

Full backtrace path corresponds to an optimal alignment / edit transcript:

Start at end; at each step, ask: which predecessor gave the minimum?

	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

A: From here

Q: How did I get here?

Edit distance: getting the alignment

Full backtrace path corresponds to an optimal alignment / edit transcript:

Start at end; at each step, ask: which predecessor gave the minimum?

	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

A: From here

Q: How did I get here?

Edit distance: getting the alignment

Full backtrace path corresponds to an optimal alignment / edit transcript:

Start at end; at each step, ask: which predecessor gave the minimum?

	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
```

Edit transcript:

MMDMMMMIMMMMM

JOHNS HOPKINS
WHITING SCHOOL of ENGINEERING

Edit distance: summary

Matrix-filling dynamic programming algorithm is $O(mn)$ time and space

- FillIng matrix is $O(mn)$ space and time, and yields edit distance
- Backtrace is $O(m + n)$ time, yields optimal alignment / edit transcript