# Algorithms in Bioinformatics

## More dynamic programming

## Recap

- HW - up after class
  due on Tuesday the 2nd
- No class next Thursday

# Dynamic Programming:
aka "Smart Recursion"

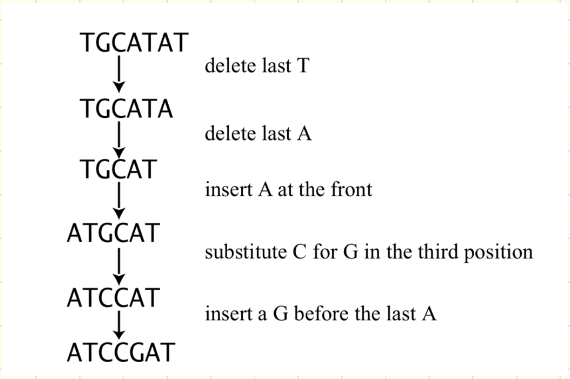
High level idea:

- Formulate recursion
- Notice recursive alg. is slow
- Store recursive call answers for lookup

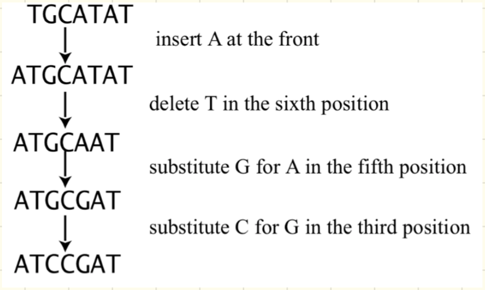
$\rightsquigarrow$ Wait, could just do this iteratively

# Last time: Edit distance

Example:

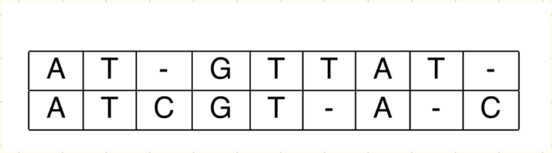


$$\begin{tikzcd}
\text{TGCATAT} \arrow[d, "\text{delete last T}"] \\
\text{TGCATA} \arrow[d, "\text{delete last A}"] \\
\text{TGCAT} \arrow[d, "\text{insert A at the front}"] \\
\text{ATGCAT} \arrow[d, "\text{substitute C for G in the third position}"] \\
\text{ATCCAT} \arrow[d, "\text{insert a G before the last A}"] \\
\text{ATCCGAT}
\end{tikzcd}$$



$$\begin{tikzcd}
\text{TGCATAT} \arrow[d, "\text{insert A at the front}"] \\
\text{ATGCATAT} \arrow[d, "\text{delete T in the sixth position}"] \\
\text{ATGCAAT} \arrow[d, "\text{substitute G for A in the fifth position}"] \\
\text{ATGCGAT} \arrow[d, "\text{substitute C for G in the third position}"] \\
\text{ATCCGAT}
\end{tikzcd}$$

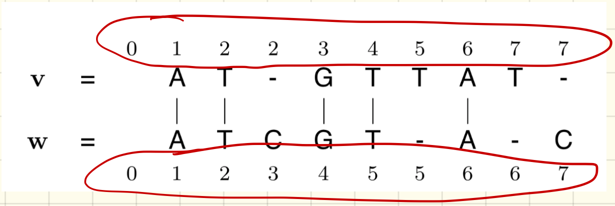
Alignment matrix:



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

(at most m+n columns)

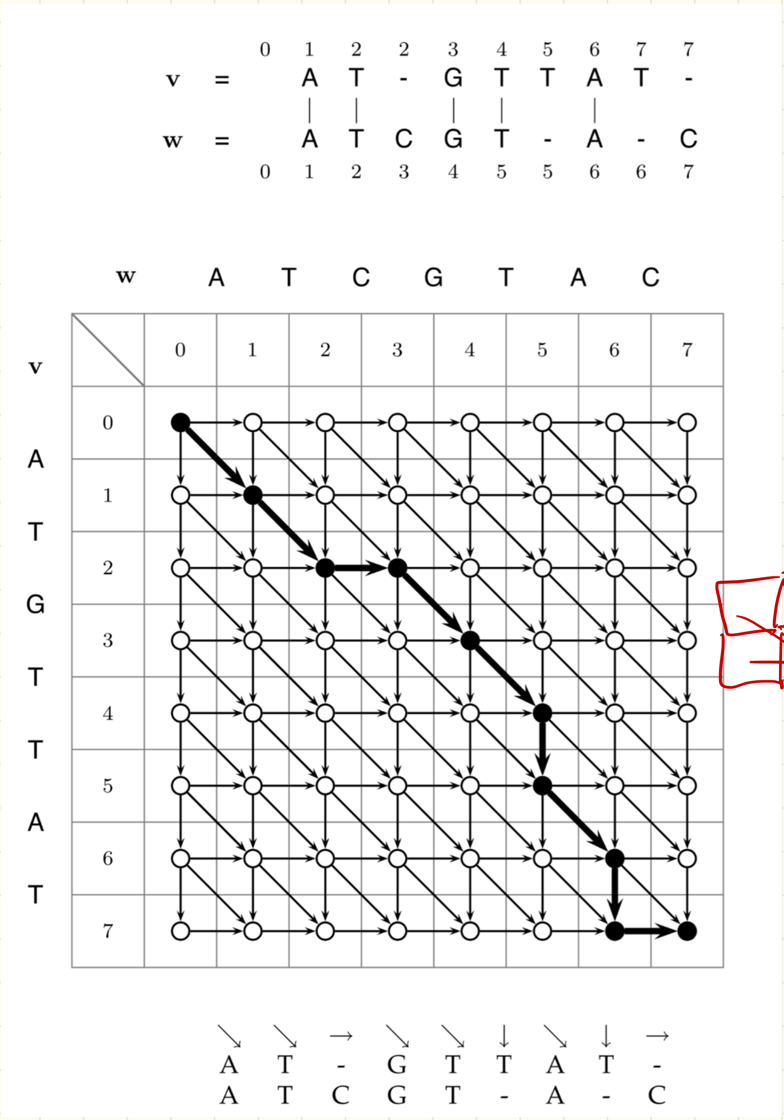
Another way:
Write # of ~~repititions~~ repititions:

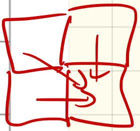


$$\begin{array}{cccccccccccc}
 & & 0 & 1 & 2 & 2 & 3 & 4 & 5 & 6 & 7 & 7 \\
v & = & & A & T & - & G & T & T & A & T & - \\
 & & & | & | & & | & | & & | & & \\
w & = & & A & T & C & G & T & - & A & - & C \\
 & & 0 & 1 & 2 & 3 & 4 & 5 & 5 & 6 & 6 & 7
\end{array}$$

This leads to formulation from last time.

Book's view:





# Now: Longest Common Subsequence

**Longest Common Subsequence Problem**:
*Find the longest subsequence common to two strings.*

**Input:** Two strings, v and w.

**Output:** The longest common subsequence of v and w.

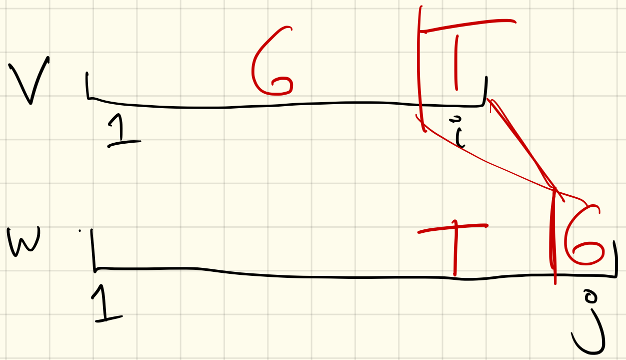
ATGAA~~T~~

GTAAA~~G~~

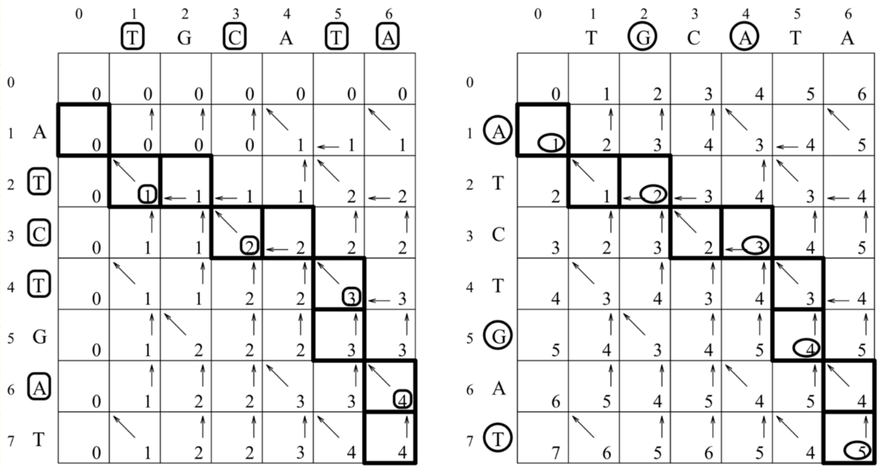
Let $s_{i,j}$ = length of LCS between $v[1..i]$ and $w[1..j]$.

If I look at last letter(s), 2 options:

- last letter is common
- last letter isn't



Aside: LCS & Edit distance are different!



| | | 0 | 1 T | 2 G | 3 C | 4 A | 5 T | 6 A |
|---|---|---|---|---|---|---|---|---|
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | T | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| 3 | C | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| 4 | T | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| 5 | G | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| 6 | A | 0 | 1 | 2 | 2 | 3 | 3 | 4 |
| 7 | T | 0 | 1 | 2 | 2 | 3 | 4 | 4 |

Computing similarity s(V,W)=4
V and W have a subsequence TCTA in common

| | | 0 | 1 T | 2 G | 3 C | 4 A | 5 T | 6 A |
|---|---|---|---|---|---|---|---|---|
| 0 | | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | A | 1 | 2 | 3 | 4 | 3 | 4 | 5 |
| 2 | T | 2 | 1 | 2 | 3 | 4 | 3 | 4 |
| 3 | C | 3 | 2 | 3 | 2 | 3 | 4 | 5 |
| 4 | T | 4 | 3 | 4 | 3 | 4 | 3 | 4 |
| 5 | G | 5 | 4 | 3 | 4 | 5 | 4 | 5 |
| 6 | A | 6 | 5 | 4 | 5 | 4 | 5 | 4 |
| 7 | T | 7 | 6 | 5 | 6 | 5 | 4 | 5 |

Computing distance d(V,W)=5
V can be transformed into W by deleting A,G,T and inserting G,A

Alignment:
```
A T - C - T G A T
- T G C A T - A -
```

(But connected still...)

see HW..

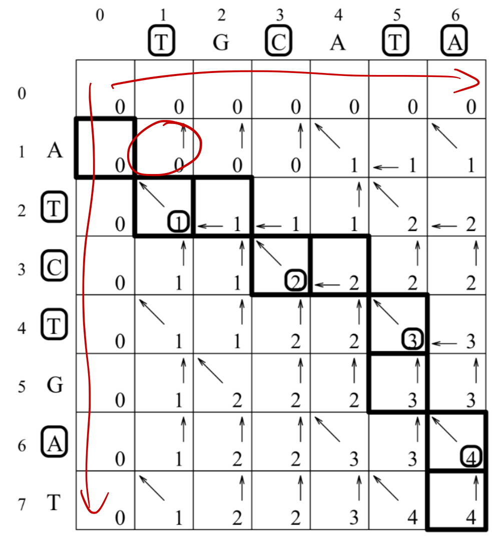
So: Try them all!

V[1..i], W[1..j]

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, & \text{if } v_i = w_j \end{cases}$$

↑ these are in LCS

Then think about not recomputing - so store in big table again



| | | 0 | 1 T | 2 G | 3 C | 4 A | 5 T | 6 A |
|---|---|---|---|---|---|---|---|---|
| 0 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | T | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| 3 | C | 0 | 1 | 1 | 2 | 2 | 2 | 2 |
| 4 | T | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| 5 | G | 0 | 1 | 2 | 2 | 2 | 3 | 3 |
| 6 | A | 0 | 1 | 2 | 2 | 3 | 3 | 4 |
| 7 | T | 0 | 1 | 2 | 2 | 3 | 4 | 4 |

Computing similarity s(V,W)=4
V and W have a subsequence TCTA in common

Alignment:

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

## Pseudo code:

```
LCS($\mathbf{v}, \mathbf{w}$)
for $i \leftarrow 0$ to $n$
    $s_{i,0} \leftarrow 0$
for $j \leftarrow 1$ to $m$
    $s_{0,j} \leftarrow 0$
for $i \leftarrow 1$ to $n$
    for $j \leftarrow 1$ to $m$
        $s_{i,j} \leftarrow \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1, & \text{if } v_i = w_j \end{cases}$
        $b_{i,j} \leftarrow \begin{cases} \text{“}\uparrow\text{”} & \text{if } s_{i,j} = s_{i-1,j} \\ \text{“}\leftarrow\text{”} & \text{if } s_{i,j} = s_{i,j-1} \\ \text{“}\nwarrow\text{”}, & \text{if } s_{i,j} = s_{i-1,j-1} + 1 \end{cases}$
return $(s_{n,m}, \mathbf{b})$
```

```
PrintLCS($\mathbf{b}, \mathbf{v}, i, j$)
if $i = 0$ or $j = 0$
    return
if $b_{i,j} =$ “$\nwarrow$”
    PrintLCS($\mathbf{b}, \mathbf{v}, i-1, j-1$)
    print $v_i$
else
    if $b_{i,j} =$ “$\uparrow$”
        PrintLCS($\mathbf{b}, \mathbf{v}, i-1, j$)
    else
        PrintLCS($\mathbf{b}, \mathbf{v}, i, j-1$)
```

Now: Back to some biology!

LCS is a way to score similarity:

- +1 for a match
- +0 for a mismatch (indel)

Edit distance is too!

- insertion, deletion + substitution all cost +1.

Biology changes are more complex...

Generalize:

Make a scoring matrix $\delta$:

Ex: $\delta$:

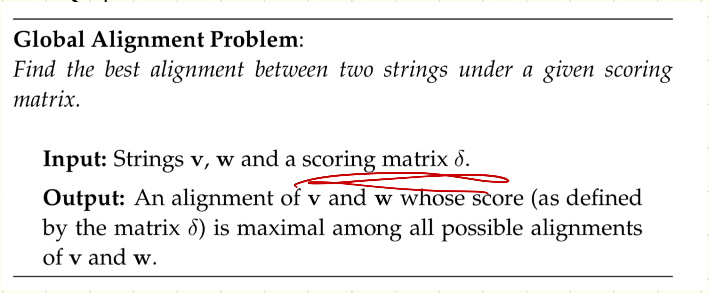
$$\begin{array}{c|ccccc} & G & C & T & A & - \\ \hline G & 1 & & & & \\ C & & \boxed{1} & \boxed{-3} & & \\ T & & & 1 & & \\ A & & & & 1 & \\ - & & & \boxed{+\frac{1}{2}} & & \end{array}$$

C-T mutation charge

T insertion deletion charge

usually negative

New goal:



**Global Alignment Problem**:
*Find the best alignment between two strings under a given scoring matrix.*

**Input:** Strings **v**, **w** and a scoring matrix $\delta$.

**Output:** An alignment of **v** and **w** whose score (as defined by the matrix $\delta$) is maximal among all possible alignments of **v** and **w**.

Same type of recursion:

When looking at $v[1..i]$ and $w[1..j]$

- could match ~~them~~ $v[i] + w[j]$

score for $v[1..i-1]$ $w[1..j-1]$ $\rightarrow$ $T[i-1, j-1] + \delta(v[i], w[j])$

- could match $v[i]$ to $-$

$T[i-1, j] + \delta(v[i], -)$

- could match $w[j]$ to $-$

$T[i, j-1] + \delta(-, w[j])$

End recurrence:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \delta(v_i, -) & \text{remove } v[i] \text{ + recurse} \\ s_{i,j-1} + \delta(-, w_j) & \text{remove } w[j] \text{ + rec.} \\ s_{i-1,j-1} + \delta(v_i, w_j) & \text{match last two} \end{cases}$$

Example:

- charge mismatches by $-\mu$
- indels by $-\sigma$
- and add $+1$ for matches

Get score = #matches $- \mu$(#mismatches) $- \sigma$(# indels)

Then:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} - \mu, \text{ if } v_i \neq w_j \\ s_{i-1,j-1} + 1, \text{ if } v_i = w_j \end{cases}$$

(Note: LCS is example of this! )
$\sigma = 0$

Runtime: Same alg! $O(mn)$

## How to get $\delta$?

- For DNA, usually just ask for $\mu$ & $\sigma$ as part of input.

- For amino acids - harder!
  - Point Accepted Mutations (PAM)
  - Block Substitution (BLOSUM)

(Two most common ones)

Reason: probability that Ser mutates to Phe is $\approx 3$ times higher than Trp to Phe.

But - gets complex!

(Read 6.7 for details - it's constructed iteratively.)

# Local alignment

Instead of entire string, might want to find substrings with good alignment.

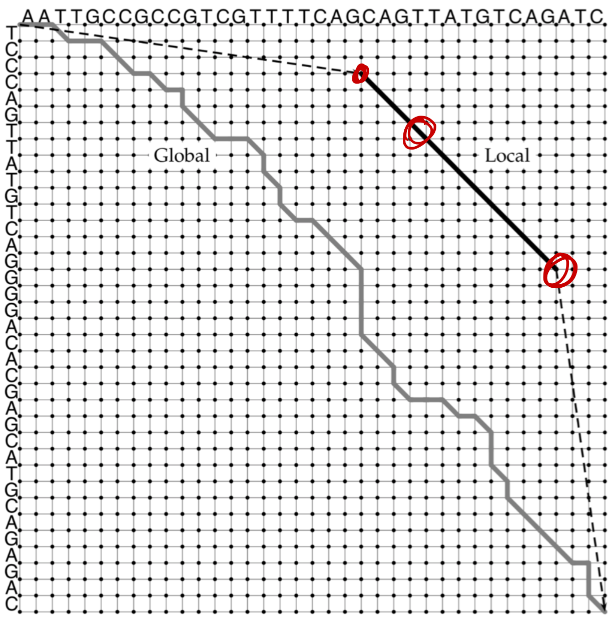
(Book's example: homeobox genes)

High level issue:

431

```
--T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
  |   || |   ||   | | | |||     || |   | |   | ||||    |
AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C

                    tccCAGTTATGTCAGgggacacgagcatgcagagac
                       ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```



AATTGCCGCCGTCGTTTTCAGCAGTTATGTCAGATC
TCCCAGTTATGTCAGGGGACACGAGCATGCAGAGAC
Global
Local

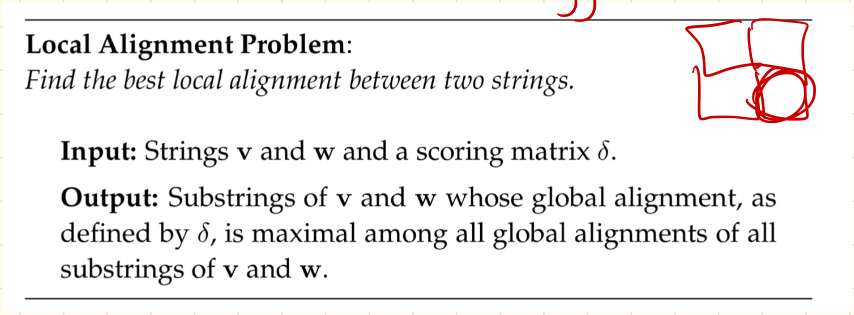
Essentially, since we charge:

- $-\mu$ for mismatches
- $-\sigma$ for indels,

Global alignment will likely miss this good substring entirely.

So:

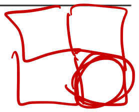
$S_{i,j} =$



**Local Alignment Problem**:
*Find the best local alignment between two strings.*

**Input:** Strings **v** and **w** and a scoring matrix $\delta$.

**Output:** Substrings of **v** and **w** whose global alignment, as defined by $\delta$, is maximal among all global alignments of all substrings of **v** and **w**.



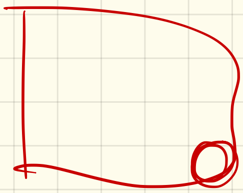
However, this is still pretty close -

- instead of best $(0,0)$ to $(m,n)$ path
- Want best $(i,j)$ to $(i',j')$ path, for <u>any</u> $i,j,i',j'$.

Really – just need to let you start match at any $(i,j)$ if matching $v[1..i]$ + $w[1..j]$ is ~~bad~~

So:

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \\ s_{i-1,j-1} + \delta(v_i, w_j) \end{cases}$$



Then:
- Build a matrix
- Fill it in
- Look at entire
→ $m \times n$ matrix for highest score

```
→ for i ← 1 to n
    → for j ← 1 to m
        4 lookups, 3 additions + take max
```

Runtime:

$O(mn) + O(mn) \overset{O(1)}{} \Rightarrow \boxed{O(mn)}$

also $O(mn)$ space

Next time:

- Alignment w/ Gap penalties
- Multiple alignment

Read 6.3
for next time