9/26

## LOCAL ALIGNMENT ~ SMITH-WATERMAN ALGORITHM

**Input:**

- $X = x_1 x_2 \dots x_m$
- $Y = y_1 y_2 \dots y_n$
- $x_i\, y_j \in \Sigma$ ($\Sigma$ = an alphabet)
- $\delta$ : scoring scheme

**compute:**

- Do global alignment of every substring of X and every substring of Y → return pair of substrings w/ max global alignment
- $V(i,j)$ = value of optimal global alignment btwn all pairs of suffixes of $X_i$ (prefix length $i$ of X) and $Y_j$ (prefix length $j$ of Y)
- $V^*$ = OPTIMAL LOCAL ALIGNMENT
  $= \max(V(i,j)),\ 0 \le i \le m,\ 0 \le j \le n$
  = max value in DP table

**The algorithm:**

$V(i,0) = 0,\ \forall i,\ 0 \le i \le m$

$V(0,j) = 0,\ \forall j,\ 0 \le j \le n$

For $i > 0$ and $j > 0$:

$$V(i,j) = \max \begin{cases} 0, \\ V(i-1,j) + \delta(x_i, -) \\ V(i, j-1) + \delta(-, y_j) \\ V(i-1, j-1) + \delta(x_i, y_j) \end{cases}$$

- there are $m^2$ substrings in string of length $m$:
  → brute force comparison of every substring in strings length $m$ and $n$ : $O(m^2 \cdot n^2)$
- However, Smith-Waterman = $O(mn)$ ~ quadratic    YAY ♥ Beautiful algorithm ♥

example:

$X$ = ACACTC

$Y$ = ACTCCA

$\delta$: identity: +1

mismatch: -1

indel: -2

| | | A | C | T | C | C | A |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| C | 0 | 0 | 2 | 0 | 1 | 1 | 0 |
| A | 0 | 1 | 0 | 1 | 0 | 0 | 2 |
| C | 0 | 0 | 2 | 0 | 2 | 1 | 0 |
| T | 0 | 0 | 0 | 3 | 1 | 1 | 0 |
| C | 0 | 0 | 1 | 1 | 4 | 2 | 0 |

→ max local alignment:

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

(start backtrack @ $V^*$, stop when reach a zero)

- The 0 in the recursion-relation essentially gives you a restart
- For global alignment, the optimal alignment length for $|X| = m$ and $|Y| = n$ is $O(n)$
- For local alignment, the length of the alignment = $O(\log(n))$
- Thus, the scoring scheme for local alignment has to be designed in a very sophisticated manner
  - if the scoring scheme doesn't penalize appropriately, you would end up w/ a very long alignment, which might not achieve our goal.

## INTRO TO GAP ALIGNMENT

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

} biologists tend to prefer larger gaps as opposed to lots of little gaps

- A little bit of biological context:
  - DNA is transcribed into RNA, which is translated into proteins
  - RNA is spliced (non-coding introns are removed) before translation
  - so, if you have spliced RNA and you want to align it with the genome, you want to use an alignment algorithm that is tolerant of long gaps to account for the RNA's missing introns
- How do we favor longer gaps algorithmically?
- Affine Gap Alignment has 2 parameters:
  - penalty for opening a gap: $\alpha$
  - penalty for continuing a gap: $\beta$

  } $\alpha > \beta$
- How do we score a gap cluster of size $n$?
  - logarithmic in $n$? Quadratic in $n$?

$\hookrightarrow$ most useful = linear in $n$:

score of gap cluster = $\alpha + n(\beta)$

This algorithm is gonna have 4 matrices instead of 1... get excited!