3.1 Global Alignment

3.1.1 Background Recap

Recall from the previous lecture that an alignment for strings $x$ and $y$ over alphabet $\Sigma$ is a sequence over pairs of elements of $\Sigma$, no more than one of which may be replaced by the special gap character ‘-’. Taking the first member of every pair yields the first sequence, with gaps inserted, and similarly taking the second member of every pair yields the second sequence, gaps inserted.

Recall also that a similarity matrix $\delta$ is used to score an alignment, and these scores are simply the sums of similarities of value pairs in the alignment.

We also discussed the edit graph and gave a recursive formulation for the computation of the global alignment score between two sequences. Now, we are ready to give a dynamic programming algorithm to efficiently compute this score.

3.1.2 Pseudocode for the Needleman-Wunsch Algorithm for Global Alignment

GLOBAL PAIRWISE ALIGNMENT:

**Given:**
1. Sequence $\vec{x} \in \Sigma^m$.
2. Sequence $\vec{y} \in \Sigma^n$.
3. Similarity matrix $\delta$.

**Produce:** An alignment $\vec{a}$ of length $l$ of $\vec{x}$ and $\vec{y}$ that maximizes the alignment similarity score $\sum_{i=1}^{l} \delta(\vec{a}, 0, \vec{a}_{i,1})$.

```
1: function GLOBAL ALIGNMENT($x \in \Sigma^m, y \in \Sigma^n$)
2:     $S_{0,0} \leftarrow 0$
3:     for $i \in \{1,2,\ldots,m\}$ do
4:         $S_{i,0} \leftarrow S_{i-1,0} + \delta(x_i,-)$
5:     end for
6:     for $j \in \{1,2,\ldots,n\}$ do
7:         $S_{0,j} \leftarrow S_{0,j-1} + \delta(-,y_j)$
8:     for $i \in \{1,2,\ldots,m\}$ do
9:         $S_{i,j} \leftarrow \max \left\{ S_{i-1,j-1} + \delta(x_i,y_j) \right\}$
10:             $S_{i-1,j} + \delta(x_i,-)$
11:             $S_{i,j-1} + \delta(-,y_j)$
12:     end for
13: end for
```

3-1
Recall the recursive definition of alignment scores from the previous lecture:

\[ S_{0,0} = 0 \]

\[ S_{i,j} = \max \begin{cases} S_{i-1,j-1} + \delta(x_i, y_j) \\ S_{i-1,j} + \delta(x_i, -) \\ S_{i,j-1} + \delta(-, y_i) \end{cases} \]

The pseudocode above correctly implements this recurrence by handling the edge cases (lines 2, 3, 4, and 7), and then every time \( S_{i,j} \) is evaluated for \( i,j > 0 \), the algorithm ensures that \( S_{i-1,j}, S_{i,j-1}, \) and \( S_{i-1,j-1} \) have already been calculated.

There are other orderings in which we could fill the matrix, for instance we could fill rowwise or diagonally, and the result would be identical. Any ordering that ensures that all recursive dependencies of a given cell are calculated before the cell in question produces an identical solution.

### 3.1.3 On Edit Graphs and Dynamic Programming Matrices

Recall that the edit graph has three types of edges:

1. Horizontal edges, represent a gap in \( x \): \((i,j) \rightarrow (i - 1, j)\), for \( i \in \{1, 2, \ldots, n\} \uparrow \)
2. Vertical edges, represent a gap in \( y \): \((i,j) \rightarrow (i, j - 1)\), for \( j \in \{1, 2, \ldots, m\} \leftarrow \)
3. Diagonal edges, represent a match or a mismatch: \((i - 1, j - 1) \rightarrow (i,j)\), diagonal.

Interpreting the edges traversed in the path through the edit graph in the optimal alignment can be used to reconstruct the alignment itself.

### 3.1.4 A Brief Note on Multiple Sequence Alignment

In general, we don’t know how to efficiently perform multiple sequence alignment. In this problem, we want to align more than 2 strings.

To align \( k \) strings, the edit graph can be thought of as nodes in \( k \) dimensional space. Any combination of gaps and aligned characters for the \( k \) strings is possible, again with the exception of all gaps.

In the binary case, we have 3 different edges in the alignment graph, representing a match/mismatch, insertion, or deletion. More generally, we have \( 2^k - 1 \) different types of edge, as there are \( 2^k \) ways to pick between a gap or a character, minus 1 for the all gap case.

An edit graph for 3 strings is given in Figure 3.1.
3.1.5 Informal Introduction to Alignment Statistics

Maximum similarity alignments make intuitive sense, as we can use them to select for alignments that try to match similar reasons. However, we can also interpret them from a much deeper statistical framework. We get an informal look at this here.

Consider the following alignment:

```
GT-ACGTT
ACGAC-TT
```

Here we have 4 matches, 2 gaps, and 2 mismatches.

Let $p$ be the probability of a match under alignment of uniformly random sequences, $q$ be the probability of a mismatch, and $r$ the probability of an indel.

The probability of the above alignment occurring would then be $p^4 q^2 r^2$, and thus the log probability is $\log(p^4 q^2 r^2) = 4 \log(p) + 2 \log(q) + 2 \log(r)$. 

Figure 3.1: Edit Graph Diagram for AC, AGC, and AT.
Now select $k$ such that $\log \frac{k}{k} = 1$. Let

$$s' = \log (P(a\text{lignment})) = 4 \log(p) + 2 \log(q) + 2 \log(r)$$

$$s = s' - 8 \ln(k)$$

Now, $s$ represents our alignment score. Algebraically, we have the following (introducing variables $\mu = -\log \left( \frac{q}{k} \right)$, $\epsilon = -\log \left( \frac{r}{k} \right)$).

$$s = 4 \log(p) + 2 \log(q) + 2 \log(r) - 8 \log(k)$$

$$= 4 \log \left( \frac{p}{k} \right) + 2 \log \left( \frac{q}{k} \right) + 2 \log \left( \frac{r}{k} \right)$$

$$= 4 \mu + 2 \epsilon$$

In this framework, we have match score 1, mismatch score $-\mu$, and gap score $-\epsilon$.

We see here that maximizing the alignment score corresponds to determining the most likely alignment under this simple probabilistic model.