1 Scoring model

When we compare sequences we want to consider substitutions, insertions or deletions which add or remove residues. The total score assign to an alignment will be the sum of terms for each aligned pair of residues plus terms for each gap. Consider the following scoring model

\[ \sigma(x, y) = \begin{cases} 
+1 & x = y \\
-1 & x \neq y \\
-2 & \text{otherwise}
\end{cases} \]

let \( a \) be an alignment. The total score of an alignment is \( \text{Score}(a) = \sum_i \sigma(a_i,1, a_i,2) \)

Example 1.1 let \( \bar{s} \) and \( \bar{t} \) be two sequences, \( \bar{s} = \text{AGT} \), \( \bar{t} = \text{AACT} \) and \( a \) is the following alignment

\[
a = \begin{array}{cccc}
A & - & G & T \\
A & A & C & T
\end{array}
\]

the score for this alignment will be

\[ \text{Score}(a) = \sigma(A, A) + \sigma(\text{-}, A) + \sigma(G, C) + \sigma(T, T) = 1 - 2 - 1 + 1 = -1 \]

2 Global alignment: Needleman-Wunsch algorithm [1970]

We will now consider the problem of obtaining the optimal global alignment between two sequences, allowing gaps.(by optimal we mean the alignment with the maximal score) This problem is solved by the dynamic programming algorithm known as Needleman Wunsch algorithm. The idea is to build up an optimal alignment using previous solutions for optimal alignment of smaller subsequences.

Let \( a(\bar{s}, \bar{t}) \) be an alignment, at the \( i \) step we consider the optimal alignment of length \( i \) (starting right to left). at the first step (\( i = 1 \)) our alignment options are
• \( \bar{s}_n \), aligning \( \bar{s}_n \) with \( \bar{t}_n \), and we are left with the alignment of \( \bar{s}_{1,n-1} \) and \( \bar{t}_{1,n-1} \) or \( a(\bar{s}_{1,n-1}, \bar{t}_{1,n-1}) \)

• \( \bar{s}_n \), aligning \( \bar{s}_n \) with a gap, and we are left with the alignment of \( \bar{s}_{1,n-1} \) and \( \bar{t}_{1,n} \) or \( a(\bar{s}_{1,n-1}, \bar{t}_{1,n}) \)

• \( \bar{t}_n \), aligning \( \bar{t}_n \) with a gap, and we are left with the alignment of \( \bar{s}_{1,n} \) and \( \bar{t}_{1,n-1} \) or \( a(\bar{s}_{1,n}, \bar{t}_{1,n-1}) \)

In order to solve our problem we construct the matrix \( V \) where the value \( V_{i,j} \) is the score of the best alignment between the initial segment \( \bar{s}_{1..i} \) of \( \bar{s} \) and the initial segment \( \bar{t}_{1..j} \) of \( \bar{t} \).

\[
V = \begin{array}{cccccc}
0 & 1 & \ldots & i-1 & i & \ldots & n \\
0 & & & & & & \\
1 & & & & & & \\
\vdots & & & & & & \\
j-1 & & & & & & \\
j & V_{i-1,j-1} & V_{i,j-1} & & & & \\
\vdots & & V_{i-1,j} & \rightarrow & V_i,j & & \\
im & & & & & & \quad V_{n,m}
\end{array}
\]

The bottom right corner value \( V_{n,m} \) is the score of the optimal alignment. Each matrix cell value is calculated using its 3 adjacent cells according to the following update rules

\[
V_{i-1,j-1} \rightarrow V_{i,j-1} \downarrow \quad V_{i-1,j} \rightarrow V_{i,j}
\]

\[
V_{i,j} = \max \left\{ \begin{array}{c}
V_{i-1,j-1} + \sigma(\bar{s}_i, \bar{t}_j) \\
V_{i-1,j} + \sigma(\bar{s}_i, -) \\
V_{i,j-1} + \sigma(-, \bar{t}_i)
\end{array} \right\}
\]

Handling the boundary conditions:

• along the top row, where \( j = 0 \), the value of \( V_{i,0} \) is not defined, so we will defined it as \( V_{i,0} = \sum_i \sigma(\bar{s}_i, -) \)

• Likewise the left column cells \( V_{0,j} \) will be defined as \( V_{0,j} = \sum_j \sigma(-, \bar{t}_j) \).

**Example 2.1** Filling the matrix for \( \bar{s} \) and \( \bar{t} \)
At each matrix cell we need to remember where we came from by saving a pointer to our father cell. We can do this by using another matrix in which we will store the father for each cell (one out of 3 options). Using the father pointer matrix we can restore the optimal alignment by going backward from the bottom right corner.

There may be many alignments with an optimal score, actually there can be exponentially large number of such alignments. In the above example two such alignments with optimal score of -1 are

- A G T
  A A C T

or

A - G T
A A C T

Note that if we will update the matrix cells top to bottom and left to right is it enough to save only the 2 left most columns.

So by now we found the optimal alignment given our model and a score function. This algorithm is called a global alignment because we are aligning one sequence against another sequence start to end, if say, one is a subsequence of the other we will pay a penalty for the missing edges.

Algorithmic time complexity, at each step we have an update operation that is based on 3 matrix cells and each such update is $O(1)$ and we have $n \cdot m$ such updates so overall we got a total of $O(mn)$ complexity.

Algorithmic space complexity, we need to store a total of $m \cdot n$ cells so we have a space complexity of $O(mn)$. remember that if we are only interested in the optimal score without the actual alignment we can pass on the father pointer matrix and save only the 2 left most columns of the matrix and so reduce the space complexity to $O(\min(m, n))$ or in other words get a linear space complexity.

Now let consider the problem of finding the actual alignment but with linear space complexity.

let $(u = \lceil \frac{n}{2} \rceil, v)$ be the indexes that are in an optimal alignment, where $u$ is the mid point of the sequence $\bar{s}$. $(u, v) \in a^* = \text{argmax}_{a \in A(\bar{s}, \bar{t})} \{\text{Score}(a)\}$. The problem of finding the optimal alignment can now be break down to finding the alignment of
a^*_1 and a^*_2 where a^*_1 is the optimal alignment of \( \bar{s}_{1..u} \) and \( \bar{t}_{1..v} \) and a^*_2 is the optimal alignment of \( \bar{s}_{u..n} \) and \( \bar{t}_{v..m} \).

\[
a^*_1 = \arg\max_{a \in A} \{ \text{Score}(a) \}
a^*_2 = \arg\max_{a \in A} \{ \text{Score}(a) \}
\]

Let \( T_1 \) be the memory complexity of the original problem. \( T_1 = c \cdot n \cdot m \) (where \( c \) is some constant). Assuming we somehow know \( u \) and we are left with finding the alignment of \( a^*_1 \) and \( a^*_2 \). The space complexity for solving \( a^*_1 \) will be \( c \cdot n^2 \cdot v \), and for \( a^*_2 \) we will get \( c \cdot \frac{n^2}{2} \cdot (m - v) \) so the total space complexity is \( c \cdot \frac{n^2}{2} \cdot m \) which is exactly \( \frac{1}{2} \cdot T_1 \).

By now we have break down the problem into 2 smaller problems and used only half of the memory of the original solution. This is an example of a [Divide and conquer algorithm](https://en.wikipedia.org/wiki/Divide_and_conquer). The only problem is that we don’t know \( v \). However we can remember \( v \) in a similar way of saving the father pointer. At each cell update we will also update the what was the point \( v \) where we crossed \( u \). This way the bottom right corner cell will have \( v \) without any additional calculation and with a linear memory complexity.

Formally, let \( C \) be a matrix where \( C_{i,j} = k : (u,k) \in a^* = \arg\max_{a \in A} \{ \text{Score}(a) \} \) for \( i \geq u \). Note that finding \( v \) is done in linear space complexity, as before we need to save only the 2 left most columns. We can keep breaking the subproblems into smaller problems and get a total space complexity of \( T = T_1 + T_2 + T_3 + \ldots = n \cdot m + \frac{n \cdot m}{2^2} + \frac{n \cdot m}{4} + \ldots = n \cdot m \cdot (1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots) \leq 2 \cdot n \cdot m \). summing it all up, we found an optimal global alignment with \( O(n \cdot m) \) time complexity and linear space complexity.

### 3 Local alignment: Smith Waterman algorithm

This time we will consider finding the best alignment between the subsequences \( \bar{s} \) and \( \bar{t} \). There are few modifications to the Global alignment algorithm so it will fit to the problem of finding the best local alignment. we need to somehow model start and end states. so we could ignore the starting and closing gaps penalties. We can achieve this by

- Taking the maximal score in the whole matrix \( V \) instead of the bottom right corner. The alignment can now end anywhere, so the best alignment score can also be found anywhere in the matrix.
- We will have to modify a bit the update operation of the matrix. We will ignore negative values, that is

\[
V_{i,j} = \max \begin{cases} 
V_{i-1,j-1} + \sigma(\bar{s}_i,\bar{t}_j) \\
V_{i-1,j} + \sigma(\bar{s}_i, -) \\
V_{i,j-1} + \sigma(-,\bar{t}_i) \\
0
\end{cases}
\]

The new option 0 corresponds to starting new alignment at any starting point. If the best alignment up to some point has a negative score, it is best to start a new one, rather than extending the alignment.
In order to restore the best local alignment one can start at the highest matrix score cell and start backtracking using the father pointers back to a 0 value cell. This path corresponds to the best local alignment.