1 Alignment

We have seen two symmetrical approaches to the problem of "comparing sequences": the edit distance measures how different two strings are, and the longest common subsequence, that measures how similar they are. **What if we combine the two approaches into a unique one?**

Idea: given two strings S and T, define a similarity measure where

- *differences* have a negative effect: define **negative scores** for the weight of **edit operations** (insertions, deletions, substitutions).
- *conserved letters* have a positive effect: define **positive scores** for the weight of **matches** between letters of the two strings.

The goal is now to **find the alignment that maximises the score** resulting by the weights just defined.

1.1 Global alignment

The goal of **global alignment** is to **try to align every character in every sequence**. We can find a solution for this problem once again using dynamic programming, in a similar way to the previous algorithms.

The rule for calculating the cell values is a combination of what we have seen so far:

$$E(S(i),T(j)) = \mathbf{max}\begin{cases} E(S(i),T(j-1)) + \mathbf{w_d} \\ E(S(i-1),T(j)) + \mathbf{w_d} \\ E(S(i-1),T(j-1)) + \mathbf{w_m} & \text{if} \quad s_i = t_j \\ E(S(i-1),T(j-1)) + \mathbf{w_s} & \text{if} \quad s_i \neq t_j \end{cases}$$

We are still looking for the "max": we're trying to find the alignment with the maximum score. As before, we can use pointers to keep track of the choice made for each cell.

For the example we set $\mathbf{w_d} = -\mathbf{2}, \mathbf{w_s} = -\mathbf{1}, \mathbf{w_m} = \mathbf{1}$.

	-	**H**	**O**	**M**	**E**
-	0	-2	-4	-6	-8
H	-2	1	-1	-3	-5
O	-4	-1	2	0	-2
U	-6	-3	0	1	-1
S	-8	-5	-2	-1	0
E	-10	-7	-4	-3	0

Backtracking as before we can build the alignment:

```
H O M - E
H O U S E
```

1.1.1 Needleman-Wunsch algorithm

The Needleman-Wunsch is the most famous global alignment algorithm and is exactly the same as above but formalized in a different way.

$$M(i,j) = \mathbf{max} \begin{cases} M(i,j-1) + \mathbf{g} \\ M(i-1,j) + \mathbf{g} \\ M(i-1,j-1) + \boldsymbol{\sigma}(\mathbf{s_i}, \mathbf{t_j}) \end{cases}$$

We no longer distinguish matches/mismatches: the score of aligning s_i and t_j **is already included in the substitution matrix $\boldsymbol{\sigma(s_i, t_j)}$**

Given an alphabet $\boldsymbol{\Sigma}$ **the substitution matrix** is a $|\Sigma| \times |\Sigma|$ matrix that for every pair of characters $a_i, a_j \in \Sigma$ gives us the **"weight" in the alignment of substituting $\mathbf{a_i}$ with $\mathbf{a_j}$**. It has the following properties:

- the matrix is symmetrical, i.e. $\sigma(a_i, a_j) = \sigma(a_j, a_i)$.
- given a_i, the values $\sigma(a_i, a_j)$ for all the characters a_j can be different in the matrix.
- values for "matches" $\sigma(a_i, a_i)$ can be different for every letter a_i.

The simplest substitution matrices are the ones used for DNA/RNA sequences. The scores for matches (positive) and substitutions (negative) do not change with nucleotides. For example:

	A	**C**	**G**	**T**
A	+2	-1	-1	-1
C	-1	+2	-1	-1
G	-1	-1	+2	-1
T	-1	-1	-1	+2

1.2 Local alignment