1. Understanding costs and scores.

Consider the alignment below:

\[
\begin{align*}
\text{ACGATGAT} & \quad \text{--CT} \\
\text{A-GA-CATAAAT} &
\end{align*}
\]

What is the cost of the alignment in the unit cost edit distance model? What is the global alignment score the alignment defines, with the mismatch and indel penalties \(-1\) and match premium \(+1\)? What is the best local alignment score inside the given global alignment?

2. Understanding matrix filling.

Compute the edit distance between \text{ACGTA} and \text{AGAA} by filling the dynamic programming matrix, and output the optimal alignment(s).

3. Implementing approximate string matching.

Write a python program that implements the approximate string matching algorithm (page 19 of the lecture slides).

4. Overlap alignments: tricks with zeros.

We are interested in overlap alignments of strings \(A\) and \(B\) such that suffix of \(A\) is aligned against prefix of \(B\). For example, an overlap alignment of \text{ACGATGAT} and \text{GACATAAAT} is

\[
\begin{align*}
\text{ACGATGAT} & \quad \text{GA-CATAAAT} \\
\end{align*}
\]

a) Derive a variant of global alignment recurrence that gives the best scoring overlap alignment of \(A\) and \(B\).

a) Derive a variant of edit distance recurrence that gives the overlap alignment of \(A\) and \(B\) with minimum cost, with the restriction that overlap should be at least of length \(\ell\). (Why is such restriction required?)

5. Developing a dynamic programming recurrence.

The Change Problem is to convert some money \(M\) into given denominations, using the smallest possible number of coins. For example, given the euro cent denominations \(\{50, 20, 10, 5, 2, 1\}\), the smallest number of coins to make up 46 cents is \(\{20, 20, 5, 1\}\). More formally:

\[\text{Input:}\ \text{An amount of money } M \text{ and an array of } d \text{ denominations } c = \{c_1, c_2, \ldots, c_d\} \text{ in decreasing order of value } \{c_1 > c_2 > \ldots > c_d\}.\]

\[\text{Output:}\ \text{A list of } d \text{ integers } i_1, i_2, \ldots, i_d \text{ such that } c_1i_1 + c_2i_2 + \ldots + c_di_d = M \text{ and } i_1 + i_2 + \ldots + i_d \text{ is as small as possible.}\]

Show how dynamic programming can be used to solve the Change Problem.

\[\text{Hint.}\ \text{Fill an array of size } M \text{ from left to right.}\]