Dynamic Programming

$\star \sum\prod \longrightarrow \prod\sum$

String (DNA) Matching

A DNA strand can be thought of simply as a sequence of *nucleotides*, where a nucleotide has one of four types, denoted by the letters A,C,G, and T. One problem of great interest is to find a sequence of DNA that is common to two individuals.

$\star$ Let the two individuals (X and Y) have the following DNA sequences:

$$\begin{aligned} X &= TGCATA \\ Y &= ATCTGAT \end{aligned}$$

We want to compute the longest common subsequence, Z, of X and Y.

We formalise the problem as follows. Given two sequences or strings $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, we want to find the *longest* common subsequence (LCS) of X and Y.

We say that a sequence $Z = \{z_1, z_2, \ldots, z_k\}$ is a subsequence of X if there is some sequence of integers $i_1, i_2, \ldots, i_k$ where $1 \leq i_1 < i_2 < \ldots < i_k \leq m$, such that $z_j = x_{i_j}, 1 \leq j \leq k$. If Z is a subsequence of both X and Y, we say that Z is a common subsequence of X and Y.

⋆

1 A brute force solution

We enumerate all subsequences of X and check each subsequence to see if it is also a subsequence of Y, while keeping track of the longest sequence found. Each subsequence of X corresponds to a subset of the indices $\{1, 2, \ldots, m\}$. The total number of subsequences of X is:

⋆

This calls for a different approach. The cost is too large so we need to consider reduction techniques.

The LCS problem has an optimal-substructure property:

Claim 9 *Let* $X = \{x_1, x_2, \ldots, x_m\}$ *and* $Y = \{y_1, y_2, \ldots, y_n\}$ *be subsequences, and let* $Z = \{z_1, x_2, \ldots, z_k\}$ *be any LCS of* X *and* Y.

1. *If* $x_m = y_n$, *then* $z_k = x_m = y_n$ *and* Z_{k-1} *is an LCS of* X_{m-1} *and* Y_{m-1}.
2. *If* $x_m \neq y_n$, *then* $z_k \neq x_m$ *and* Z *is an LCS of* X_{m-1} *and* Y.
3. *If* $x_m \neq y_n$, *then* $z_k \neq y_n$ *and* Z *is an LCS of* Y_{n-1} *and* X.

⋆

This claim allows us to reduce the size of the problem. Let us define $c(i, j)$ to be the length of an LCS of the sequences X_i and Y_j. If either $i = 0$ or $j = 0$ then $c(i, j) = 0$. Otherwise, the optimal substructure of the LCS problem yiels the following recursive formula

⋆

$$c(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max[c(i, j-1), c(i-1, j)] & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Note that the subproblems are not independent. Divide and conquer algorithms partition the problem into independent subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem. However, in our case, we need a different approach: dynamic programming. Dynamic programming solves every subproblem just once and then saves its answer in a table, thereby avoiding the problem of recomputing the answer every time the subproblem is encountered. Here is the solution to our problem:

- Construct a table c with $m+1$ rows and $n+1$ columns. The row index goes from 0 to m and the column index from 0 to n. The table stores $c(i,j)$. The first row of c is filled from left to right (using our recurrence relation), then the second row, and so on.
- Construct a table b with $m+1$ rows and $n+1$ columns. $b(i,j)$ points to the table entry corresponding to the optimal subproblem solution chosen when computing $c(i,j)$.

⋆

⋆

The pseudo-code for this algorithm is in Section 15.4 of the textbook.

⋆ What's the cost of the new algorithm?

Sometimes, we don't need to compute a LCS, but only the length of the LCS. A cheaper solution is to consider only two rows at a time: the row being computed and the previous row. The storage cost of this is

⋆