Sunday, 17 December 2023

Assignment 2: Dynamic Programming

My Approach

The solution uses a `LetterMatrix` to store the scores of aligning different pairs of nucleotides (A, C, G, T) and gaps (`-`). This matrix is used to calculate the alignment score based on specific rules (e.g., matching, mismatching, and gaps). The alignment process is executed by the `alignment` method, which computes the optimal alignment score recursively.

Dynamic programming is applied to avoid redundant calculations. This is evident in the use of `memoization` (a 2D array) to store intermediate results and the `track` array to trace back the alignment path. The `memoization` array holds the best scores for subproblems, while `track` records decisions made at each step to enable reconstruction of the alignment.

The `alignment` function computes scores for three cases at each step: aligning the current characters of both strings, aligning the current character of the first string with a gap, and aligning the current character of the second string with a gap. It then selects the option with the highest score.

Finally, the `reconstruct` method backtracks from the end of the strings to the beginning, using the `track` array to build the aligned sequences.

Time Analysis

The time complexity of this solution is primarily dictated by the `alignment` method.

1. ***Initialization***: The initialization of the `memoization` and `track` arrays is O(nm), where `n` is the length of the first string and `m` is the length of the second string.

2. ***Alignment Computation***: The recursive `alignment` method is called for each pair of indices in the two strings. In the absence of memoization, this would lead to exponential time complexity. However, with memoization, each pair of indices is computed at most once, leading to a total of O(nm) subproblems.

3. ***Reconstruction***: The `reconstruct` method iterates over the lengths of the two strings in the worst case, which is also O(n + m). However, this does not change the overall time complexity.

Overall, the time complexity of the algorithm is O(nm), making it efficient for typical sequence alignment tasks. The use of dynamic programming converts an exponential brute-force problem into a polynomial-time one.