Implementation:

read_file(filename)

The `read_file` function is designed to read a dataset file and extract the sequence information needed for analysis. It takes the filename as input, opens the file, and reads all lines while removing any extra whitespace. The first two lines of the file are converted to integers representing the number of sequences and the sequence length (or window size), while the remaining lines are returned as a list of sequences. The function outputs the number of lines, the length of each sequence, and the list of sequences, providing a structured format for further processing.

smith_waterman(sequence1, sequence2, match, mismatch, gap)

The `smith_waterman` function implements the Smith–Waterman algorithm to compute the local alignment score between two sequences. It takes two sequences along with scoring parameters for matches, mismatches, and gaps as input. The function initializes a scoring matrix of size `(len(sequence1)+1) x (len(sequence2)+1)` with zeros and fills it using dynamic programming: the diagonal cells are updated with the match or mismatch score, and the up and left cells are updated with the gap penalty. At each step, the maximum of these values and zero is chosen to ensure local alignment. The function tracks the maximum score in the matrix and its position, returning the matrix, the highest alignment score, and the coordinates of the maximum score.

find_motifs(sequences, window_size)

The `find_motifs` function identifies recurring motifs across multiple sequences using a sliding window approach combined with Smith–Waterman local alignment. The function takes a list of sequences and a window size as input and generates candidate motifs by sliding a window of the specified length across each sequence, storing the subsequence along with its sequence index and starting position. It then compares all windows from different sequences using Smith–Waterman and keeps only those with scores above a defined threshold. The results are stored in a dictionary mapping each motif to a list of alignment scores. Finally, motifs are ranked by two criteria: frequency, counting how many high-scoring alignments each motif has, and score, taking the maximum alignment score for each motif. Both rankings are sorted and returned to highlight the most significant motifs.

main()

The `main` function orchestrates the entire motif discovery pipeline, coordinating data input, motif identification, and results output. It first ensures that an output directory exists and then loops through each dataset. For each dataset, sequences are read using the `read_file` function, motifs are identified using `find_motifs`, and the top motifs are printed and saved to text files. The results include the top 10 motifs by both score

and frequency and are separated by headers for clarity. This function allows the analysis of multiple datasets in an automated manner and produces organized output suitable for reporting or further examination.

Results

Example:

4.1 Dataset 1

=== RESULTS FOR DATASET1 ===

:Top 10 motifs by Score
Motif: ATGCA, Score: 10
Motif: CATGC, Score: 10
Motif: ACGTA, Score: 10
Motif: TACGT, Score: 10
Motif: GTACG, Score: 10
Motif: CGTAC, Score: 10
Motif: GGCAT, Score: 8
Motif: CCATG, Score: 8
Motif: GGTAC, Score: 8
Motif: TGCAA, Score: 8

:Top 10 motifs by Frequency
Motif: ATGCA, Count: 11
Motif: ACGTA, Count: 8
Motif: CATGC, Count: 7
Motif: TACGT, Count: 5
Motif: GGCAT, Count: 4
Motif: GTACG, Count: 4
Motif: CGTAC, Count: 4
Motif: CCATG, Count: 3
Motif: GGTAC, Count: 3
Motif: TGCAA, Count: 2

Conclusions:

The motifs `ATGCA`, `ACGTA`, `CATGC`, `TACGT`, `GTACG`, and `CGTAC` are highly conserved and frequently appear across sequences, indicating they are key recurring patterns.