Final Project
Bioinformatics Pipeline

Objective

The objective of this project is to deepen your understanding of sequence analysis and manipulation in bioinformatics using the tools we have at our disposal. Specifically, we will be implementing a bioinformatics pipeline used to perform data reduction via clustering, data cleanup via chimera removal, and phylogenetic alignment of organisms sequenced by amplicon sequencing.

Instructions

You are tasked with implementing a small bioinformatics pipeline in Python that performs various operations on DNA sequences, including sorting, clustering, chimera detection, and multiple sequence alignment. This project will make use of the following programs you have previously written for this course:

- Assignment #1: Ability to read, write, and sort sequences in FASTA format.
- Assignment #2: Ability to perform global alignment via Needleman-Wunsch.
- Assignment #3: Ability to perform local alignment via Smith-Waterman.
- Assignment #4: Ability to perform multiple sequence alignment via Feng-Doolittle.

Your Python program will need to implement the following tasks, with each task representing a piece of the pipeline. Each task is additive meaning it uses the data from all stages before it. As such, even though your solution will run as a single program, each task will produce its own set of output for grading.

Task #1 – Reading and Sorting of FASTA formatted Sequence Data

In bioinformatics, length of the sequence often correlates with various biological functions, hence sorting sequences by length helps in downstream analysis. Like Assignment #1, this task requires that your solution be capable of accepting a FASTA file containing N sequences taken as input (-i). Once the sequences have been read in, it will need to go through and sort these sequences by length from longest to shortest. Sequences of equal length should be sorted by the order they were encountered in the original FASTA file.

Input

This task will receive the input sequence data via the command line argument '-i' which will inform your solution where the input sequence data is stored.

Output

This task will produce the intermediary file <output_file>.sorted.fna where <output_file> is the path and filename given by the command line argument '-o'.

Task #2 – Sequence Clustering

Amplicon sequencing generates an immense amount of data. However, because amplicon sequencing depends on PCR amplification to amplify a particular section of the genome repeatedly, we end up with large portions of the data being identical or near identical to one another. Given the nature of our alignment algorithms though, we would expect highly similar sequences to produce similar (if not identical) alignments. To reduce the complexity of downstream analysis, we perform a step known as clustering to group high similar sequences together and then use single sequence to represent them all - known as the representative sequence.

The algorithm to perform this process is as follows:

1. Implement an identity scoring function that can calculate the "percent identity" between two aligned sequences.
    a. The percent identity is the number of exact matches between two aligned sequences divided by the length of the sequences.
    b. Example: The aligned sequences below contain 4 bases that are identical between the alignments and the total length of each sequence is 7, so the percent identity would be $\frac{4}{7} = 0.5714\ldots$ or ~57.14%.
        i. Sequence 1: `AGC-TA-`
        ii. Sequence 2: `-GCATAA`
2. Using the sorted sequences from Task #1, perform the following:
    a. Mark the longest sequence as the representative sequence for the first cluster.
    b. For each remaining sequence, perform the following:
        i. Perform NW alignment of the sequence against each representative sequence.
        ii. After each alignment, calculate the percent identity.
            1. If the percent identity is >=97% add the sequence to the cluster that the representative sequence was representing, then return to 2b.
        iii. If the sequence does not join any cluster, then set it as a new cluster and set the sequence as the representative sequence for the new cluster.
3. You should end up with a list of representative sequences, where each sequence represents a cluster containing 1 to many member sequences.
    a. Write each of the representative sequences to a new file called <output_file>.clustered.fna (defined below). The sequence header should have "; size=##" appended to each representative sequence where ## is the number of member sequences in the cluster.
        i. *Example:* >Sequence 1; size=150

Input

This task will receive the sorted sequence data from Task #1 as well as the scoring matrix given via the command line argument '-s' which will define the substitution scores that should be utilized by the NW algorithm.

Output

This task will produce the intermediary file <output_file>.clustered.fna where <output_file> is the path and filename given by the command line argument '-o'.

Task #3 – Chimera Detection and Filtering

In biological sequence analysis, a chimera (also known as a chimeric sequence) refers to a sequence that is formed from the combination of two or more different parent sequences. This is commonly an artifact of amplification techniques like polymerase chain reaction (PCR), where sequences can be artificially joined together during the amplification process. The objective for this task is to identify and remove all possible chimera sequences from the data set, a process known as "chimera filtering".

Your solution must implement a form of de novo chimera detection and filtering. This can be achieved using the Smith-Waterman algorithm to compare each representative sequence to all others. If a sequence (from a cluster with fewer members) aligns to two or more different sequences (from clusters with a higher member count), it is likely a chimera. A high-level algorithm for this process is as follows:

1. For each representative sequence, calculate pairwise alignments using the Smith-Waterman algorithm with all other representative sequences.
    a. Reject and remove any alignment that fails to meet the following criteria:
        i. Good alignments must reach an alignment score ≥50.
        ii. Good alignments must be at least 60 bases in length.
2. For each representative sequence (A), determine if there exist good alignments (defined above) between it and two other representative sequences (B and C) such that the size of B > A and the size of C is >A.
    a. In other words, the number of sequences in clusters B and C are greater than the number in cluster A.
3. If such a match can be found, mark the cluster as possibly chimeric (is a chimera).
    a. Write all chimeric sequences to the output file <output_file>.chimeric.fna defined in the Output section below.
4. If no such match can be found, mark the cluster as likely nonchimeric (not a chimera).
    a. Write all non-chimeric sequences to the output file <output_file>.nonchimeric.fna defined in the Output section below.

Input

This task will receive the clustered sequences generated during Task #2 as well as the scoring matrix given via the command line argument '-s' which will define the substitution scores that should be utilized by the SW algorithm.

Output

This task will produce two intermediary files where <output_file> is the path and filename given by the command line argument '-o':

- **<output_file>.nonchimeric.fna**: A fasta file containing the representative sequences determined likely to not be chimeras.
- **<output_file>.chimeric.fna**: A fasta file containing the representative sequences determined likely to be chimeras.

Task #4 – Multiple Sequence Alignment

Multiple sequence alignment is a way of arranging the sequences to identify regions of similarity that could be a consequence of functional, structural, or evolutionary relationships between the sequences.

Using the non-chimeric sequences from Task #3, your solution should utilize the Fitch-Margoliash algorithm to determine a guide tree, then utilize the Feng-Doolittle algorithm to select and merge sequences until all sequences have been merged into a single MSA. The resulting aligned sequences will then be written to an output file named <output_file>.msa.fna as defined below in the output section. Similar to Assignment #5, your final output should have the score appended to each sequence header as follows:

1) The sequence header for each sequence will have a semicolon ";" appended to the end of it as well as a "score=***S***" where ***S*** represents the alignment score.
2) The sequence will now be the aligned sequence containing any gaps being represented by hyphens "-".
3) Example:
    i. Input

```
>Input Sequence 1; size=134
ACGTAGCTAGCTAGG
```

    ii. Output

```
>Input Sequence 1; size=134; score=13
ACGTAG-CTAG--CTAGG
```

Input

This task will receive the non-chimeric representative sequences from Task #3 as well as the scoring matrix given via the command line argument '-s' which will define the substitution scores that should be utilized by the NW algorithm.

Output

This task will produce the intermediary file <output_file>.msa.fna where <output_file> is the path and filename given by the command line argument '-o'.

Python Program Specifications

Your program must be compatible with Python 3.11.3. The grading script will enforce testing against that version of Python using only the built-in modules as well as access to numpy, scipy, and matplotlib. No other modules are available for use.

Your program must be capable of accepting command line arguments as described below:

- -i <path_to_input_FASTA_file>
  - Purpose: This option retrieves the file path to the starting FASTA file.
  - Input Type: String
  - Validation: Entire file path must exist, otherwise error.
  - Required: Yes
- -o <path_to_output_FASTA_file>
  - Purpose: This option will be utilized to determine the path and name for all output files.
  - Input Type: String
  - Validation: The string must be alphanumeric.
  - Required: Yes

- -s <path_to_score_matrix_file>
  - Purpose: This option retrieves the file path for the scoring matrix.
  - Input Type: String
  - Validation: Entire file path must exist, otherwise error.
  - Required: Yes

Example execution:

```
python3 Eric_Rees_R123456_final_project.py -i in.fna -o output -s nucleotide.mtx
```

- Instructs python to run Eric_Rees_R123456_final_project.py
- Sets the input file option to “in.fna”.
- Sets the output file option to “output”.
  - This will generate files such as output.sorted.fna, output.clustered.fna, etc…
- Sets the score matrix file option to “nucleotide.mtx”

Scoring Matrix File Specifications

The scoring matrix file contains the nucleotide or amino acid scores for each possible pair of bases. This file follows the same specifications as were written in Assignment #2.

Additional Rules and Hints

Program Output

Your program may print as much information as you wish to stdout / stderr during execution. The first line of output printed to stdout during execution **must** be “Final Project :: R#” where R# is your specific TTU R#. That is the only **required** item it must print and is the only line the grading script cares about. Your program correctness grade will be determined by the output found in the file specified by the -o argument.

Hints for getting started

1) Make sure your Assignment #1, #2, #3, and #5 are fully functional.
    a. Be sure these assignments can pass their respective grading scripts at 100%.
    b. Any errors, even small ones, will cause your project to skew wildly from the correct answer.
2) Go task by task in order.

What to turn in to Blackboard

A zip archive (.zip) whose name does not matter which contains your source code files such that:

- The primary python file for your project is named <FirstName>_<LastName>_R<#>_final_project.py
    - Example: Eric_Rees_R123456_final_project.py
- Any additional files necessary for your program to execute should be included.

Keep in mind the grading program will require that your python file be named appropriately. Failure to submit your code as a zip archive will result in point deductions. Failure to submit your code using the correct naming scheme will result in point deductions.