Running Exercise II – Develop a sequence alignment ToolboxData

files. *Running Exercise 2-1.tgz* and *Running Exercise 2-2.tgz*.

Background. REII is designed for two students and is divided into two parts accordingly. Unpaired students should complete only the first part. REII aims to build A Python application that aligns DNA sequences (part 1), converts them to amino acids, and analyzes the output (part 2).

Part one (Student I)

Goal. We want to calculate how well two DNA sequences align with each other, measure, and visualize the results. Below are more specific instructions:

1. How well do two DNA sequences align with each other?

Scoring (default): If we have a match between two nucleotides in a given position, it will be scored as +1. If we have a transition, a<->g or c<->t, the penalty will be -1, and if it is a transversion (any other substitution), the penalty will be -2. A gap penalty is -1. A figure showing transitions/transversions can be found here.

For example, for this input:

```
>id1
a   c   g   t   a   g   g   t   t   t   t   a
>id2
a   t   g   t   -   -   g   t   t   t   -   -
```

The alignment score is 3:

```
a   c   g   t   a   g   g   t   t   t   t   a
a   t   g   t   -   -   g   t   t   t   g   a
1  -1  +1  +1  -1  -1  +1  +1  +1  +1  -2  +1  = 3
```

Write a program that reads a FASTA file with two or more sequences and outputs the IDs of the aligned sequences, their identity, gaps, and alignment score. If the fasta file includes non-ACTG characters, remove them. Your program should work for both lower and upper case characters. If the fasta file contains multiple sequences, do anall-against-all calculation. Your program should output the results to the screen and a file. For example, if the file has three sequences, print:

```
Id1-id2: Identity: 8/12 (66.7%), Gaps: 2/12 (16.7%), Score=3
Id1-id3: Identity: 4/12 (33.3%), Gaps: 2/12 (16.7%), Score=1
Id2-id3: Identity: 12/12 (100%), Gaps: 0/12 (0%), Score=12
```

Allow the user to provide a file parameters.txt with alternative scoring.

Data. Data for the assignment can be downloaded from Canvas. The archive file *RunningExercise2-1.tgz* includes several examples of FASTA files, but you should create your own examples to test your code.

Execution. Your program should take one FASTA file with two or more sequences and output a single filewith the alignment scores for each pair. Your code should run like this:

```
python FastaAligner.py input_fasta.fna parameters.txt [optional] output_fasta.txt [optional]
```

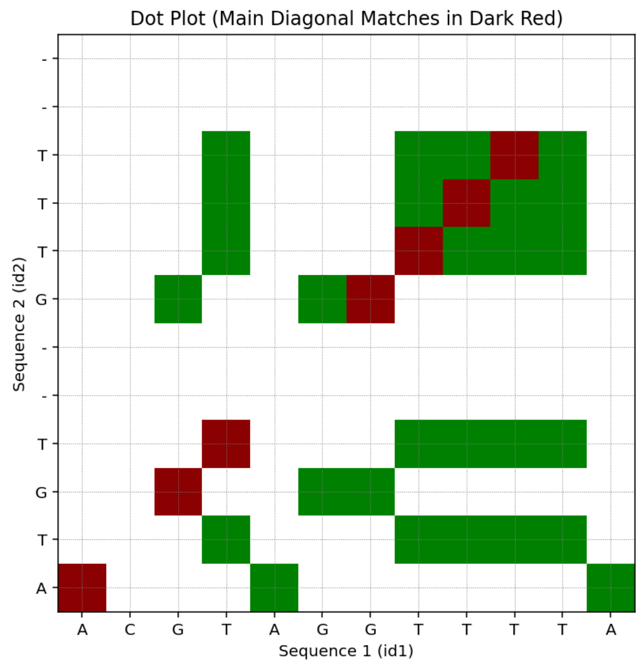
2. Visualize the alignment

Write Python code to visualize the alignment using a dot plot, allowing the user to evaluate its accuracy visually.

For example, for this input:

```
>id1
a  c  g  t  a  g  g  t  t  t  t  a
>id2
a  t  g  t  -  -  g  t  t  t  -  -
```

The output would be:



Here's an updated version of your description that reflects the new orientation and coloring scheme (bottom-left → top-right diagonal, dark red for main-diagonal matches):

In this dot plot, the two aligned sequences are displayed along the **X-axis** (sequence 1) and the **Y-axis** (sequence 2). Each cell in the grid represents a comparison between a nucleotide from sequence 1 and a nucleotide from sequence 2.

- **Dark-red squares** indicate **matches along the main diagonal** — positions where identical nucleotides occur at the same index in both sequences. These reflect perfectly aligned bases.

- **Green squares** represent **matches off the main diagonal**, showing regions of local similarity between non-corresponding positions.
- **White cells** indicate **mismatches or gaps**, where the nucleotides differ or one sequence contains a missing base (“–”).

Because the Y-axis increases from bottom to top, the main diagonal of perfect alignment runs from the bottom-left to the top-right corner of the plot. Breaks or shifts in this diagonal highlight substitutions, insertions, or deletions, while parallel or crossing diagonals can reveal repeated or inverted sequence segments.

Execution. Your program should take one FASTA file with two or more sequences and output a single filewith the alignment scores for each pair. Your code should save the result as a combination of the names of the 2 fasta headers separated by _. In the above example: `id1_id2.png`.

Your code should run like this:

```
python FastaAlignerPlotter.py input_fasta.fna parameters.txt [optional]
```

Note, you should make use of **matplotlib** and **NumPy** modules to generate the plot efficiently.

Submission. Compress ALL your scripts, input, and output files, and call them *FastaAlignerProject-1.zip.*

Upload this single compressed file to Canvas by the deadline.

Important notes.

1. Assume the sequences are aligned (i.e., have the same length). You are not asked to write an alignment program, only to evaluate the alignment.
2. Your program should be completely robust to prevent users from trying to crash it (us). In your documentation, write all the cases your code can handle and the decisions you made.
3. You are NOT allowed to use BioPython or similar modules that were not studied in class (unless specified in this exercise).
4. When you test new sequences on your algorithm, they must also be aligned. But how do we align sequences? For that, you can use these two aligner tools. This is a quick aligner: https://www.bioinformatics.org/sms2/pairwise_align_dna.htm. This one is slow, but it will give you the similarity and gap scores that you need to output, so you can check your code https://www.ebi.ac.uk/Tools/psa/emboss_needle/

Part two (Student II)

Goal. Write a Python application that supports the following options (for each sequence in the fasta file):

1. Read a DNA file and convert the DNA to amino acids.
2. Read a file with amino acids and print the absolute abundances of each amino acid to a file.
3. Read a DNA file, read a barcode file, trim the barcodes, and write the final DNA to a file.

Below are more specific instructions:

1. Converting a multi-FASTA DNA sequence to amino acids. The application will accept a DNA file in a fasta format, with one or more sequences, convert T->U, and convert it to amino acids using the standard genetic code (we will skip the translation to mRNA step this time). Stop codons should appear as *. Remember to check that the input file consists ONLY of DNA letters.

For example, when receiving the following input file:

```
>sequenceID-001 description
AAGTAGGAATAATATCTTATCATTATAGATAAAAACCTTCTGAATTTGCTTAGTGTGTAT
ACGACTAGACATATATCAGCTCGCCGATTATTTGGATTATTCCCTG
>sequenceID-002 description
CAGTAAAGAGTGGATGTAAGAACCGTCCGATCTACCAGATGTGATAGAGGTTGCCAGTAC
AAAAATTGCATAATAATTGATTAATCCTTTAATATTGTTTAGAATATATCCGTCAGATAA
TCCTAAAAATAACGATATGATGGCGGAAATCGTC
>sequenceID-003 description
CTTCAATTACCCTGCTGACGCGAGATACCTTATGCATCGAAGGTAAAGCGATGAATTTAT
CCAAGGTTTTAATTTG
```

The output will be:

```
>sequenceID-001 description
K*E*YLIIIDKNLLNLLSVYTTRHISARRLFGLFP

>sequenceID-002 description
Q*RVDVRTVRSTRCDRGCQYKNCIIID*SFNIV*NISVR*S*K*RYDGGNR

>sequenceID-003 description
LQLPC*REIPYASKVKR*IYPRF*F
```

You can start translating from the start of the sequence (don't look for the start codon). You may check your solution at: https://www.bioinformatics.org/sms2/translate.html

Execution. Your program should take a FASTA file with two or more sequences and output a single file with the relevant AA. Your code should run like this:

```
python dna2aa.py DNA.fna output_file.txt [optional]
```

Note, DNA.fna is NOT provided. You need to create it yourself.

2. Counting amino acid abundances. Your code should accept an input file from the user containing amino acids and provide an optional output filename. It will then print the absolute combined abundances of allthe amino acids in the input file to the output file. Example input:

```
>id1
ACQWRDB
FHIGLSB
>id2
ACDWRDJ
WHIGLSJ
```

Example output:

```
A 2
C 2
D 3
E 0
F 1
...
W 1
Y 0
X 4
```

The two fields should be separated with a space or tab. The script should handle both lower- and upper-case amino acids. The 20 amino acids have the abbreviations:

```
A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y
```

The application should take a multi-FASTA file with possibly interleaved sequences. Only the 20 standardamino acids should be printed; otherwise, denote the letter as X and show it at the bottom of the list.

Execution

```
python amino_count.py amino.faa output_file.txt [optional]
```

3. Trimming the barcodes. A common procedure before sequencing is to put a barcode (a predefined oligonucleotide) in the 5'-end of the sequences (it can also be used in the 3'-end). **The barcodes often represent different samples**. Your script should accept an input file in fasta format with one or more sequences and an optional output filename, which is a fastq file *barcode.fastq* (=default name, read about this format), which is available to you. Filter the fasta file from the barcodes (exact matches only) and print each sample (represented by the same barcodes) to a single fastq file after removing the barcode. Thebarcodes are TATCCTCT, GTAAGGAG, and TCTCTCCG. If no barcodes are found, print the entry (sequences or sequence + quality value) to a second fastq file.

For example (in a fasta file, but the input will be in fastq), if you get one file with three sequences, but only 2have barcodes:

```
>1
ACCGAGCGACGAGCAGTATCCTCT
>2
ACCGAAGAGAGAGAGCGACGAGCAGTATCCTCT
>3
ATTTAACCGAGCGACGAGCAGGTAAGGAG
>4
ATTTAACCGAGCGACGAGCAGTTTTTTTTTT
```

You should print the results into three files (because there are 2 barcodes + 1 file without matches):

First output file (corresponding to TATCCTCT barcode**)**: sequences that originally were with barcode (after you removed the barcode)

```
>1
ACCGAGCGACGAGCAG
>2
ACCGAAGAGAGAGAGCGACGAGCAG
```

Second output file (corresponding to GTAAGGAG barcode): sequences that originally were with a barcode (after you removed the barcode)

```
>3
ATTTAACCGAGCGACGAGCAG
```

Third output file: sequences that originally were without a barcode:

```
>4
ATTTAACCGAGCGACGAGCAGTTTTTTTTTT
```

Data. The data for the assignment can be downloaded from Canvas. The name of the archive file is *RunningExercise2-2.tgz*. The input file should be in fastq format. The corresponding sample names are *sample1* and *sample2*. The output files should be named consecutively (e.g., *sample1.fastq, sample2.fastq)* (unless the user chose a different output filename). Not all sequences will contain these three barcodes. Save these barcodes to the file *undetermined.fastq* (unless the user chose a different output filename).

Execution

```
python barcode.py input_filename output_file1.txt [optional] output_file2.txt [optional]
```

Submission. Name the program *barcode.py* and provide the output files for *barcode.fastq*.

Compress your code and output files and call them *FastaAlignerProject-2.zip.*

Upload the compressed file to Canvas by the deadline.

Notes

1. Tip: More than one file can be opened using the same *with* statement.
2. Double-check at least parts of your results with bash.
3. You are NOT allowed to use BioPython or similar modules that were not studied in class.
4. You can assume that input_filename is in fastq format.

Deadline

21/10/2025 at 12:00 (noon) with late submissions allowed until 13:00.

Team Bonus (15/100 points for each student)

Provide all your codes in Google Colab. Provide a single IPython notebook file per team.
– Upload the sequences as FASTA files and visualize them interactively.
– Add markdown cells explaining the steps.

Final notes

- The grading is independent for both parts, so please **mark which part is yours**. You are **allowed** to help your partner out, review their code, and test their code.You are **not allowed** to write their code.
- You are **not allowed** to share your code with anyone.
- We will emphasize documentation and robustness (e.g., handling errors).
- We will put emphasis on documentation and robustness (e.g., handling errors). That includes:
    - Following up our specific requirements for the documentation block at the beginning of your script (Lecutre 1, *Documentation* slide)
    - Comments throughout the code
    - QC – how well you tested your code. Provide your driver\testing units.
    - Using try-except, FileNotFound, incorrect file content, etc.