Project 2 - Global alignment with different gap costs

Group 7
Shakil Ahmed - 202402609
Swantje Katharina Hänsch - 202007286
Al Ashrif Bin Ahamed - 202402610

Introduction

The program works well with the characters A, G, C and T (DNA only), but does not handle other alphabets like RNA bases or amino acids. Also, score matrix input by file only allows for uppercase letters in the input file. Otherwise, the program works as expected.

Methods

Problem 1: Implementing alignment methods

The program is used by running the file main.py, which guides the user through the implementation choices that can be made. These are how to specify the two input sequences, how to specify the score matrix, which gap cost function (linear / affine) to use, and whether to output the optimal alignments. Our program works with the DNA bases A, G, C and T. For the input sequences, the user first is asked whether to input the sequences from a file or from the command line. If input by file is chosen, the program will ask for a file path for each sequence. To specify the character score matrix, the program will ask whether to input the score matrix as a file or on the command line. If the user choses input by file, they will be asked to input the file path on the command line The file should be in the "Phylip-like" format – here, the program i/o handles the scoring matrix in the sequential format (e.g. The first line is typically a header ("4" or "A, C, G, T") which is ignored and also each of next four lines must begin with a row label (one of "A", "C", "G", or "T") followed by exactly four integer values (one for each column corresponding to A, C, G, and T in that order). If manual input is chosen, the user will be asked to input the score for each pair of bases (guided input). For the gap score function, the program first asks whether to use a linear or an affine gap cost. If linear is chosen, the user specifies one value for the gap cost. If affine is chosen, the user specifies two values, one for the gap opening score and one for the gap extension score. At last, the user will be asked by the program whether or not the optimal alignments should be displayed and whether to save one optimal alignment (randomly) as a FASTA file.

Problem II: Experimenting with alignment methods

In testing_data.py, we construct test data by generating pairs of random sequences of bases, whose length is specified as a parameter to the function testing_data(). Therefore, we also considered the *project2_examples.txt* as a base test file and constructed a unit test (built-in test script class in python) to validate the program that yields a pass.

To illustrate the time consumption of the programs, we generate pairs of testing sequences of increasing length. This data is used to measure the running time of *global_linear_alignment()* by executing *linear_time.py* and *global_affine_alignment() by* executing *affine_time.py*, respectively. For each pair of a given sequence length, we repeated the measurement x times. The measured running times are then plotted against the sequence lengths to illustrate time consumption. Although, we also compared the time consumption of both linear and affine gap cost.

Test

In addition, we tested the program by giving it some values as inputs which might be invalid or not meaningful. The program handles invalid sequence alphabets and invalid file paths for sequence inputs and score matrices well, for instance by giving adequate error messages.

To further verify the correctness of the program, we answered the questions in *project2_eval.txt* by running our program on the sequences presented in the questions, using *evaluation.py*. The answers are as follows:

- Question 1: The optimal alignment score is 226.
- Question 2: The optimal alignment score is 266.
- Question 3: The optimal alignment scores for each pair of sequences are:

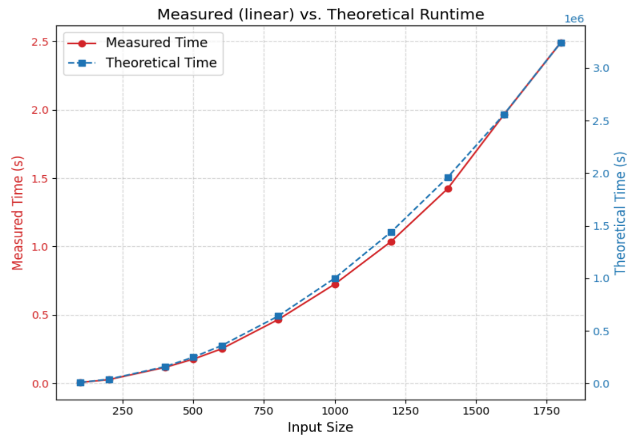
	seq1	seq2	seq3	seq4	seq5
seq1	0	226	206	202	209
seq2	226	0	239	223	220
seq3	206	239	0	219	205
seq4	202	223	219	0	210
seq5	209	220	205	210	0

- Question 4: The optimal alignment scores for each pair of sequences are:

	seq1	seq2	seq3	seq4	seq5
seq1	0	266	242	243	256
seq2	266	0	283	259	254
seq3	242	283	0	269	243
seq4	243	259	269	0	247
seq5	256	254	243	247	0

Experiments

The theoretical running time of a global alignment with linear cost is O(n^2) to find the optimal alignment score, and O(n) to backtrack. The same is the case for global alignment with affine gap cost. Thus, the overall running time for each program should be O(n^2) [O(n^2)+O(n) = O(n^2) – with backtracking]. In Figure 1, we can see that our program for both (linear and affine) cases follows the theoretical running time and the line graph lies on each other. So, it confirms the measured time follows the theoretical time.



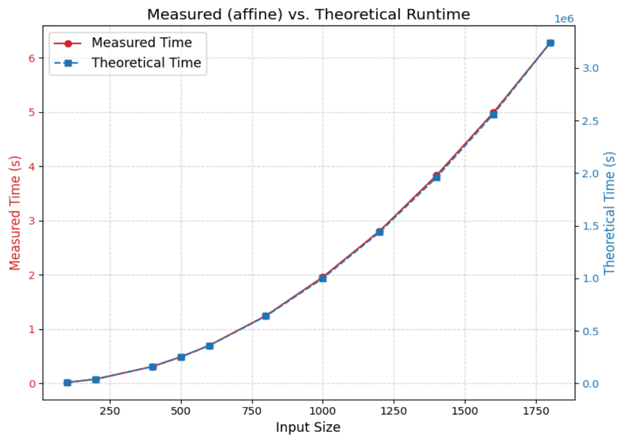


Figure 1: Measured time vs. theoretical time for global alignment with linear gap cost (left) and affine gap cost (right), respectively.

Moreover, we also compared the measured running time (*time_comparison.py*) between linear and affine gap cost with the same length of (random) sequences and we observed that although both are O(n^2), **affine gap cos**t has a larger constant factor because it does a bit more work at each cell (it updates 3 matrices each time, rather than 1). Therefore, if we run both on sequences of the same length – **Linear gap cost** is typically **faster** (smaller constant factor) whereas **Affine gap cost** is a bit **slower** but still scales in exactly the same quadratic manner as n grows (Figure 2).

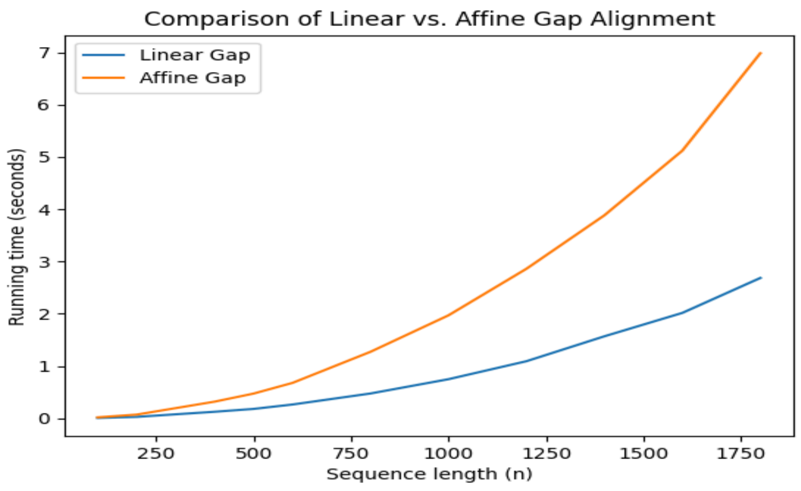
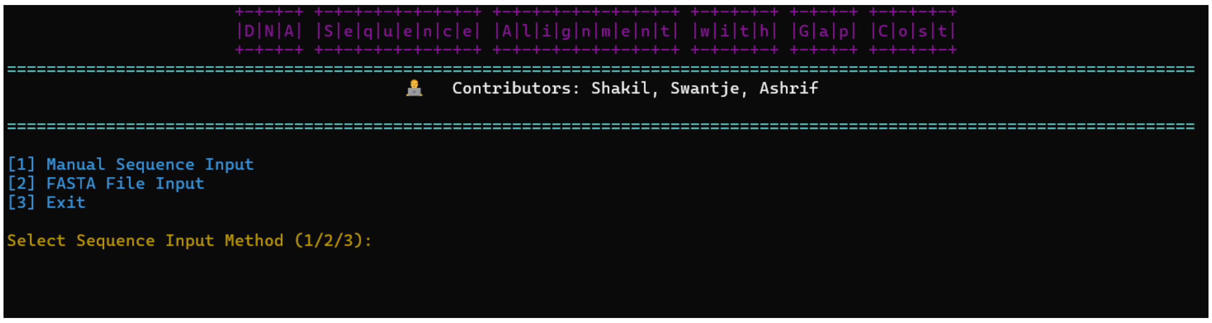


Figure 2: Measured time Linear vs Affine gap cost .

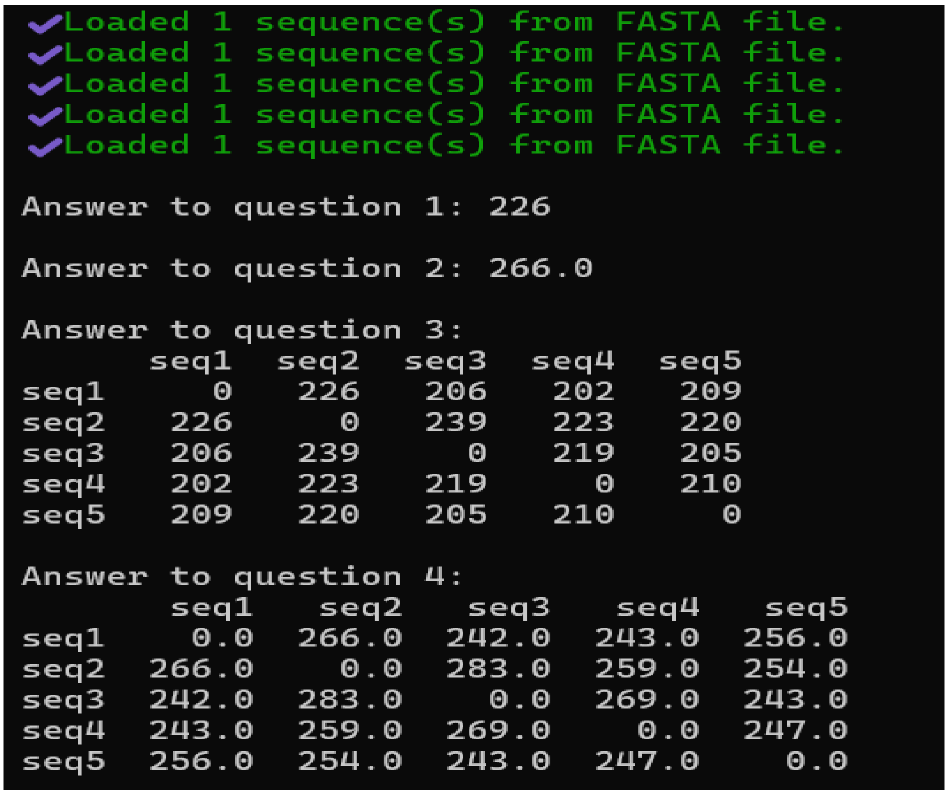
Appendix

By running *main.py,* we will get the following interface of the program.



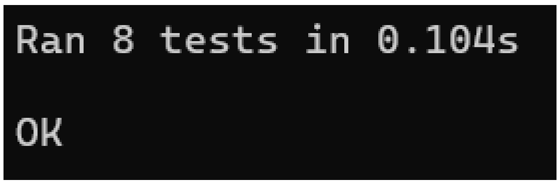
```
                          +-+-+-+ +-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+-+ +-+-+-+-+ +-+-+-+ +-+-+-+-+
                          |D|N|A| |S|e|q|u|e|n|c|e| |A|l|i|g|n|m|e|n|t| |w|i|t|h| |G|a|p| |C|o|s|t|
                          +-+-+-+ +-+-+-+-+-+-+-+-+ +-+-+-+-+-+-+-+-+-+ +-+-+-+-+ +-+-+-+ +-+-+-+-+
=========================================================================================================
                                      Contributors: Shakil, Swantje, Ashrif

=========================================================================================================

[1] Manual Sequence Input
[2] FASTA File Input
[3] Exit

Select Sequence Input Method (1/2/3):
```

By running *evaluation.py,* we will get the following validity and also the answers of Q1-Q4 stated in *project2_eval.txt*.



```
✔Loaded 1 sequence(s) from FASTA file.
✔Loaded 1 sequence(s) from FASTA file.
✔Loaded 1 sequence(s) from FASTA file.
✔Loaded 1 sequence(s) from FASTA file.
✔Loaded 1 sequence(s) from FASTA file.

Answer to question 1: 226

Answer to question 2: 266.0

Answer to question 3:
      seq1  seq2  seq3  seq4  seq5
seq1     0   226   206   202   209
seq2   226     0   239   223   220
seq3   206   239     0   219   205
seq4   202   223   219     0   210
seq5   209   220   205   210     0

Answer to question 4:
       seq1   seq2   seq3   seq4   seq5
seq1    0.0  266.0  242.0  243.0  256.0
seq2  266.0    0.0  283.0  259.0  254.0
seq3  242.0  283.0    0.0  269.0  243.0
seq4  243.0  259.0  269.0    0.0  247.0
seq5  256.0  254.0  243.0  247.0    0.0
```

By running the *test_program.py,* we will get the following test result stated in *project2_example.txt*



```
Ran 8 tests in 0.104s

OK
```