Programming for Bioinformatics | BIOL 7200

Exercise 9

Submission specifications

1. Your module ***does*** need to be generalizable to any FNA input files. We're going to be using this code to analyze different files moving forward. Don't hard code anything to the provided example files
2. You may only use Python standard library modules

Assignment description

This week's assignment is a similar format to last week. I have provided you with some data files and some scripts which will call a package that you write. This week we are going to extend the isPCR package that you wrote for the last exercise and add additional functionality that will start to take the shape of your magnum opus for this course.

More details of your assignment are provided below. Briefly, you will perform the following:

1. Slightly refactor your isPCR package so that it runs from start to finish in a single call
2. Write a Python module that includes an implementation of the Needleman-Wunsch algorithm described in class
3. Set up your isPCR and NW modules so that they can be run in series to go from assemblies to alignment using a single script.

The submission format this week is similar to last week. However, you will be combining both the isPCR and Needleman-Wunsch functionality into a single package named "magnumopus" (it will earn that name over the next few weeks). You will again submit either a single script or (probably the better way to organize things now) multiple scripts and an __init__.py file. If you include a single script, call it magnumopus.py. If you submit a __init__.py file, the autograder will create a mangumopus directory and copy all submitted *.py files (except amplicon_align.py written for question 3) into that directory, thus creating a magnumopus package from your code. The autograder will then import your magnumopus and test its functionality. Please ask on Ed Discussion if you would like any clarification.

1. Unifying the steps of your isPCR package (20 points)

Instructions

Last week's assignment required that you maintained the different steps of isPCR as separate functions. However, it would be strange to want to use an isPCR program in a stepwise manner like that. A more sensible usage would be to provide the software with primers and an assembly and get back predicted amplicons in a single step.

For this question, your task is to unify the steps in your isPCR module into a single function named "ispcr". The function should take as input the path to a primer file, the path to an assembly file, and a maximum amplicon size. i.e., it should have the following definition line:

```
def ispcr(primer_file: str, assembly_file: str, max_amplicon_size: int) -> str:
```

Your package will be tested by running the included "q1.py" script to import your "magnumopus" package and call your function.

Expected output

The expected output from the provided "q1.py" script is as follows:

```
(biol7200) ~/biol7200/ex9$ ./q1.py
>Pseudomonas_aeruginosa_PAO1_NC_002516.2:635141-635853 Pseudomonas aeruginosa PAO1,
complete genome
TCTCGGCGACGGTCAGCTCGACCTCGCTTTCCAGGGCCGCCAGCTTCTGCTGGTTGCGCAGGATGTCGTCGCGCAGGCGCTCG
ATGGCCTCGGCGTACTTCGGCTTGCTCTTCAGGACGCTGTCGACCCACTTCTCGTCGGTCTCGTGGTTCGGGAACAGGCGCAG
GAAGTCGGCACGCGGCATGCGCGCGTCACGCACGCAGAGCTGCATGATGGCGCGTTCCTGGGCGCGCACGCCTTCCAGGGCGG
AGCGCACGCGGGCGACCAGGGCGTCGAACTGCTTGGGCACCAGCTTGATCGGCATGAACAGCTCGGCCAGGCCGGTGAGTTCG
GCGGTGGCCTGCTTGCTGCCGCGACCGTGCTTCTTCAGGGCCTTCTTGGCCTTGTCGAGCTGCTCGGAGACCGCGGTGAAACG
CAGGCGGGCTTCTTCCGGATCCGGACCGCCGTCGCCTTCGTCGTCGCTGTCGCTGCTGTCGTCGCTTTCTTCTTCCTCGTCGT
CCTTCTCTTTCGAGTCGGCGGAATCGTCCTTCAGGTTGACCGGCTCCACCTCTTCGGCGGGCAGGCTGCCGTCATCGGGATCG
ATATAGCCGCTGAGGACGTCGGAGAGGCGACCGCCTTCGGCGACGATGCGATTGTAGTCGGCCAGGATGCTGTCCACCGTGCC
CGGGAACTGGGCGATGGCGCTCATCACTTCGCGGATGCCTTCCTCGATG
```

I have provided you with an example solution to last week's assignment. If you would prefer, you can use that as the foundation for this week's assignment instead of the code you wrote yourself last week.

2. Implementing the Needleman-Wunsch global sequence alignment algorithm (50 points)

Instructions

It is rare in bioinformatics to be interested in DNA sequences just for the sake of the sequence itself. Instead, the function of DNA sequences is normally the area of study and the sequence itself is just a way of getting to the information that actually relates to the research question. For example, you might care about the gene that the DNA encodes and what function it might have. Perhaps the most common way of studying DNA sequences (in order to better understand either the sequence itself or whatever gene or function the sequence is associated with) is to compare your sequence of interest to other sequences. Perhaps you have a sequence of a gene of interest from a tumor and want to understand the role of your gene in the disease. You might find that comparing the gene sequence in a healthy tissue to the sequence in the tumor would reveal differences that may be biologically significant.

Comparing two sequences to one another is conceptually straightforward. Because the order of bases in DNA sequence generally matters for its function (i.e., a gene's function isn't a product of how many As vs Gs there are, it's what order the bases are in the sequence), comparing two sequences requires that you figure out which position in the first sequence corresponds to which position in the second. Once you have corresponded the positions in each sequence you can then check which positions differ. If your two sequences are identical or a small number of bases differ between them, then it would be easy to line up the positions that correspond between the sequences and spot the differences. However, in cases where the sequences differ, it may not be easy to tell which positions correspond.

Except for when dealing with very short sequences, manually aligning sequences is infeasible. Instead, computers are used to automatically align sequences. There are a few different algorithms that have been

developed to tackle the challenge of sequence alignments. In general, these algorithms don't try to use biological information to correspond positions in the sequences. It would be enormously difficult (impossible?) to generate data to experimentally determine which bases correspond and once you had those data you wouldn't need a computer program to align the sequences. Instead, alignment algorithms are designed to produce a hypothesis of which positions in some set of sequences might correspond to one another. They generally take an approach maximizing the number of identical bases that line up in the sequences while minimizing gaps and mismatches. We covered an algorithm in class that takes this sort of approach: Needleman-Wunsch.

For this question you must write a Python implementation of that algorithm and add it to your "magnumopus" module. Your Needleman-Wunsch implementation should be called using a function named needleman_wunsch() with the following definition line:

```
def needleman_wunsch(seq_a: str, seq_b: str, match: int, mismatch: int, gap: int) -
> tuple[tuple[str, str], int]:
```

Where the return value is a tuple containing a tuple of your aligned sequences, and the score of the alignment. i.e., your return line will look something like

```
return (aligned_1, aligned_2), score
```

Your package will be tested by running the included "q2.py" script to import your "magnumopus" package and call your function. Note that you do not need to write a function that identifies all optimal alignments. You need only write a function that finds one of the optimal alignments.

If you would like to, you can write a function that finds all optimal alignments, but for the purposes of this exercise it should only return one of them. We will discuss a function that finds all alignments when we cover recursion.

Expected output

The expected output from the provided "q2.py" script is something like as follows:

```
(biol7200) ~/biol7200/ex9$ ./q2.py
CTTCTCGT-CGGTCTCGTGGTTCGGGAAC
CTT-TCATCCACT-TCGTTGCCCGGGAAC

True
11
```

If you look at the code of the q2.py script, you will see a range of possible alignments. The alignment produced by your function will be printed and, beneath that alignment, a boolean will be printed indicating whether your alignment is one of the possible alignments I expected to see. If you do not get the exact alignment shown above, but see the word "True" printed, then your output is correct. If you see False, that means your alignment is not one of the expected alignments. Any of the expected alignments will get the points for this question.

3. Putting it together (30 points)

Now you have code to amplify sequences of interest from assemblies and, separately, code to align two sequences. Those two processes are often linked in studies of DNA sequences. As described above, a common approach to understand the function of a DNA sequence is to compare it to other sequences and identify differences and similarities. However, before you can compare sequences you have to know what they are. If, for example, you were a lab biologist who wanted to compare the sequence of a gene in a tumor to that gene in healthy tissue and what you had in front of you was two test tubes containing tissue samples, then your first task would be getting the gene sequence out of the two tissue samples. One approach could be to try to directly extract only the sequence you are interested in from each tissue directly. Perhaps you would try to fragment the genome using sonication or bead beating and then pull down the target sequence using affinity chromatography. At the end you would have your target sequence in various fragments as well as some random other fragments of DNA which would constitute a sort of background noise (you always get some off-target stuff when trying to purify things in the lab). That approach would be quite laborious and expensive and you would still end up with noisy data. Instead, what people generally do is design primers to amplify the sequence of interest using PCR. At the end of a PCR reaction you end up with an enormous number of copies of your sequence of interest. You still have the input DNA as well, but the amplicons are so abundant that they essentially dilute the input DNA to the point that it is undetectable.

By combining our isPCR and Needleman-Wunsch processes, we will end up with software that performs an analysis that has been performed by enumerable scientists over the years. Instead of starting from tissue samples in a tube, we are starting with assemblies, but the process is essentially the same.

For this question, you must unite the two components of your package (isPCR and Needleman-Wunsch) into a single script, named amplicon_align.py. This script should have the following characteristics:

1. It should import and use your "magnumopus" package to perform isPCR and Needleman-Wunsch alignment. You may also define functions within the script
2. It should accept as command line input two assembly files, a primer file, a maximum amplicon size, and a match, mismatch, and gap score
3. It should have a command line interface that includes named arguments, not simply positional arguments. See the example help message below.
4. It should perform isPCR on both provided assembly files with the provided primer file
5. The amplicons from each assembly file should be aligned. There should only be one amplicon from each file so you don't have to worry about dealing with multiple
6. It should check which orientation the two sequences align best in and only return the best alignment (i.e., reverse complement one sequence to check)
7. It should print the alignment **followed by** the alignment score to the terminal. It should not print any other information such as logging.

An example of the help message printed by the example script I have written is shown below. You should use the same command line options to make it feasible for the TAs to test your code. i.e., the usage of your script must match that in the expected output example below. However, you are free to add long options and better help messages if you wish. They just won't be used or considered when grading your assignment. They'll just be extra flair.

Note that when providing negative numbers as command line inputs, argparse is prone to interpreting them as options. You can get around that in two ways: first, quote the number with a leading space (e.g., `--gap '`

`-1'`). Second, you can associate an option and its value using and "=" symbol (e.g., `--gap=-1`).

```
(biol7200) ~/biol7200/ex9$ ./amplicon_align.py -h
usage: amplicon_align.py [-h] -1 ASSEMBLY1 -2 ASSEMBLY2 -p PRIMERS -m
MAX_AMPLICON_SIZE --match MATCH --mismatch MISMATCH --gap GAP

Perform in-silico PCR on two assemblies and align the amplicons

options:
  -h, --help            show this help message and exit
  -1 ASSEMBLY1          Path to the first assembly file
  -2 ASSEMBLY2          Path to the second assembly file
  -p PRIMERS            Path to the primer file
  -m MAX_AMPLICON_SIZE  maximum amplicon size for isPCR
  --match MATCH         match score to use in alignment
  --mismatch MISMATCH   mismatch penalty to use in alignment
  --gap GAP             gap penalty to use in alignment
```

You should upload this script along with your other package/module files. The autograder will not move any file called "amplicon_align.py" when it is organizing your package files.

Expected output

```
(biol7200) ~/biol7200/ex9$ ./amplicon_align.py -1
data/Pseudomonas_aeruginosa_PAO1.fna -2 data/Pseudomonas_protegens_CHA0.fna -p
data/rpoD.fna -m 2000 --match 1 --mismatch=-1 --gap=-1
TCTCGGCGA-CGGTC-AG-CTCGACCTCG-CTTTCCAGGGCCGCCAGCTTCTGCTGGTTGCGCAGGATGT-CGTCGC-
GCAGGCGCTCGATGGCCTCGGCGTACTT-CGGCTTG-CTCTT---CAGGACGCTGTCGACCCA-CTTCTCGT-
CGGTCTCGTGGTTCGGGAACAGGCGCAGGAAGTCGGCACGCGGCATGCGCGCGTCACGCACGCAGAGCTGCATGATGGCGCGT
TCCTG-
GGCGCGCACGCCTTCCAGGGCGGAGCGCACGCGGGCGACCAGGGCGTCGAACTGCTTGGGCACCAGCTTGATCGGCATGAACA
GCTCGGCCA-GGCCGGTGAGTTCGGCGGTGGCCTGCTTGCTGC-CGCGACCGTGCTTCTTCAGGGCCTT-CTTG-
GCCTTGTCGAGCTGCTCGGAGACCGCGGTGAAA-CGCAG-GCGG-
GCTTCTTCCGGATCCGGACCGCCGTCGCCTTCGTCGTCG-CTGTCGCT-GCTGTCGTCGC-T-TTCTTCTTC-
CTCGTCGTCCTTCTCTTTCG-AGTCGGCGGAATCGTCCTTCAGGTTGACCGGCTCCACCTCTTCGGCGGGCAGGC----
TGCCGTCAT-
CGGGATCGATATAGCCGCTGAGGACGTCGGAGAGGCGACCGCCTTCGGCGACGATGCGATTGTAGTCGGCCAGGATGCTG-
TCCACCGTGCCCGGGAACTGGGCGATGGCGCTCATCACTTCGCGGATGCCTTCCTCGATG
TCTCGGCGATC-TTCAAGCCT-G-TTTCGAC-TTCAAGAGCGGTCAGCTTCTGCTGGCAACGGATGATGTCCG--
GCTGCAGGCGGGCGATGGCTTCGGCGTACTTGC-TCTTGCCT-TTGGCCAGG--GC-GTCGGTCCAGCTT-TCATCCACT-
TCGTTGCCCGGGAACTGGCGCAGGAAGTCGGCACGTGGCATGCGGGCATCACGCACGCAGAGCTGCATGATGGCGCGTTCCTG
CTG-
GCGCAGGCGATCCAGGGCACTGCGTACACGTTCAACCAGGCCTTCGAATTGCTTCGGAACCAGTTTGATCGGCATGAACAGTT
CAGCCAGGGCCAG-CATTTCGGCGATGGCTTGCTTG-TTCTCGCGGCCGTGTTTCTTCAGCGCCTTGCGGGTGAC-T-
TCCATCTGGTCGGCGACGGC-GCCAAAGCGCTGTGCGGCG-AT-GACCGGATCCGGACCGCTTTCGGCTTCTTCTTCGTCT-
TCGGTCGCT-TC--CGCTTCTTCGTCTTCGCT-GTCGTCATCCGCTTTCGCGGTC-TTGG--T-GT-CGACA-G--G--
CGGCGGCACTTCGGCGGCAGGC-GGCGTAATGCCGTCATCCGGG-
TCGATGTACCCGCTGAGAACGTCGGACAGGCGTCCACCTTCGGTGGTGACGCGAGTGTACTCGGAGAGGATG-
TGATCAACCGTGCCAGGGAAGTGCGCGATTGCGCCCATCACTTCACGGATACCCTCTTCGATA
368
```