Programming Project 1: Sequence Alignment

CS 181, Fall 2023

Out: Sep. 29
Due: Oct. 12, 11:59 PM

1 Task

You must implement the following three alignment algorithms you've learned in class:

1. Global Alignment
2. Local Alignment
3. Global Alignment with Affine Gap Penalties

Each algorithm should take in two strings u and v, a scoring matrix, and a gap penalty (or penalties in the case of affine gap). The scoring matrix will be a table that gives scores for the match or mismatch of two letters. More details on the precise form of the inputs will be given in the following section.

After implementing these algorithms, you will use them to perform alignments using real sequence data.

Note: There is an application section described below. Make sure to start early so you are not crunched for time closer to the deadline!

2 Program Specifications

2.1 Setup

To grab the support code, download from the website. We've provided example shell scripts and a few test cases and matrices.

2.2 Programming Language

- You may use any of the following programming languages to write your code for this class: Python 3, Java, R, and Julia. Our Gradescope autograder is configured to accept solutions written in these.
- Your solutions generally should not require the installation of any packages that do not come in the standard installations of your chosen programming language. However, if you are using Python, you will also have access to numpy and pandas.
- To facilitate anonymized and automated grading, each of your solutions must be accompanied by a shell script. Make sure each problem is able to output the correct result using the shell script provided. If you are using Python 3, your shell script must run your program using the command "python3" rather than "python", as "python" will run your code with Python 2.

- Your shell scripts should print exactly what is shown in the examples given for each problem. If you print any extra text, you will fail our autograder and lose points. This means that if you are coding in R, you may need to print text using “cat()” instead of “print()”.
- Be sure to print any terminal output to stdout, which is the channel that the standard print functions write to in most programming languages. If you print to stderr, your solution will be interpreted as an error message and fail the autograder.

2.3 Alignment Characters

Use ‘-’ to represent gap characters. Your project should also be case-insensitive, so ‘a’ and ‘A’ should be equivalent.

2.4 Shell Scripts And Examples

Your handin should provide three shell scripts: `global.sh`, `local.sh`, and `affine.sh`. Each script should take three arguments: (1) a path (relative or absolute) to a file with the two sequences to align, (2) a path to a file with the scoring matrix, and (3) a gap penalty (or penalties, in the case of `affine.sh`). Your script should then print the first string aligned and then the second string aligned, followed by the alignment score, as an integer. Examples of the input files are shown below.

The sequences file just has two sequences on two separate lines. For example, here are the contents of `test_cases/01.txt`. Note that input files may or may not have newlines at the end of line 2.

```
AAAGAATTCA
AAATCA
```

The scoring matrix file has rows on separate lines and columns delimited by a single space. (The top left cell will be a simple ‘X’, to make parsing the matrix a little simpler.) An example scoring matrix can be found at `matrices/standard.m`. All test case matrices will follow the same format.

```
X C T A G
C 1 -1 -1 -1
T -1 1 -1 -1
A -1 -1 1 -1
G -1 -1 -1 1
```

Here are a few examples illustrating how we expect to be able to call your shell scripts and what we expect as output.

```
> sh global.sh test_cases/01.txt matrices/standard.m -1
AAAGAATTCA
AAA---T-CA
2
```

```
> sh local.sh test_cases/01.txt matrices/standard.m -1
AATTCA
AAT-CA
4
```

```
> sh affine.sh test_cases/01.txt matrices/standard.m -1 0
AAAGAATTCA
AAA----TCA
5
```

Note that `affine.sh` requires a gap opening penalty and a gap extension penalty **(in that order!)**.

2.5 README

All projects must include a README file. Include the following infomation:

- A description of any known bugs.
- Anything you want the TAs to know about your project.

3 Patient Sample Collection and Consent

In April 2020, the UNESCO International Bioethics Committee (IBC) and the UNESCO World Commission on the Ethics of Scientific Knowledge and Technology (COMEST) released a statement addressing ethical concerns during the COVID-19 pandemic, one of them being the importance of responsible research practices.

Patient-collected samples play a crucial role in COVID-19 research- these samples enable the construction of COVID-19 reference genomes, one of which you will be working with in the next portion of this assignment.

When dealing with individual patients' samples, researchers must receive consent before moving forward. Read this paper about navigating consent in the context of the COVID-19 pandemic.

1. What are possible accessibility issues that may impact consent?
2. Would you use samples from each of the following patients for research purposes? What concerns might you have and how could they be addressed? Describe your thought process.
   (a) A patient does not speak English so their next of kin relays their verbal consent, although it is not clear what the patient is saying.
   (b) A patient of specific interest is unaware of their surroundings and physically unable to consent. They have no legal representative or next of kin.
   (c) A patient is unable to consent and a distant relative is the only person accompanying them. Their personal relationship is not clear.

(d) A patient's sample is collected as part of routine treatment. Their nurse mentions as an aside that the sample might also be used for research. They are unable to elaborate further, so the patient reluctantly says yes.

(e) A patient is shown a 10-minute long video giving an overview of how their sample might be used, although the exact research pipeline is not yet clear. They can choose to stop the video at any point to sign a consent.

4 Application: ExploreInformatics Part 1

Please go to this link to access the application material. Please keep in mind that this section should take around 3 hours to complete. You will have to share your findings/results in a document titled **application.pdf**. You **must** include this in your submission for feedback.

5 Handin

- To hand in this assignment, upload your shell scripts along with all files needed to run your code on Gradescope. If you have any subdirectories or folders that you would like to preserve in your handin, you will need to compress your submission into a zip file and upload the zip file to Gradescope. However, to ensure that the autograder runs your handin properly, make sure that all shell scripts are present at the root of your handin; in other words, do not place your shell scripts inside any folders.
- When you hand in your solution, our autograder will immediately run on your submission. We have made the test cases provided in the project handout immediately visible to you so that you can ensure your solution runs properly with the autograder. If you fail the autograder and cannot determine why, notify a TA and we will help you to diagnose the problem. If your final submission is not compatible with our autograder, it will be very difficult for us to give you credit for this assignment.
- **Application Handin:** Make sure to submit the following: `align_reads.sh`, `student_chipseq.r`, `application.pdf`, and the counts table you generated from the miniset of data.

6 Grading

We will grade your handin using pre-generated test cases, with a certain number of points allocated per test case. Your code should not throw exceptions for any valid inputs. Test edge cases extensively! We will be checking for common gotchas, including but not limited to the following:

1. Global alignment of two very short strings, e.g. '`C`' and '`G`', using various scoring schemes.
2. Local alignment of two very short strings, e.g. '`C`' and '`G`', using various scoring schemes.

You will be graded solely on the correctness of your implementation. You will not be graded on code performance, as long as it is within reasonable bounds. For this project, we will give a lenient limit of one minute for aligning two 500bp sequences. (If implemented wisely, these alignments should take less than half a second to perform.)

7 Tips

- We do not provide you with pseudocode for the algorithms in this handout. Instead, you should refer to the lecture notes/slides.
- If you are having trouble with the shell scripts or with parsing the input files, try temporarily hard-coding some test cases and scoring schemes and returning to parsing later or come to office hours. We want you to focus on the implementation of the algorithm, not the technical details of I/O.
- In the past, students have had more problems with backtracking than with filling out the dynamic programming table. Think carefully about what you need to do to backtrack. Again, feel free to come to office hours if you need help checking your logic.