BCB 444 Fall 2015 Project 2

Typical FASTA file processing

Due Sep. 16 at 1 p.m.

Introduction

FASTA files are the most common sequence data files in bioinformatics. They got the name from the sequence alignment software FASTA that originally defined this file format as follows:

```
>seq1_name field1 field2 field3 …
AGTCATGGAGCCGTTAGTACC…
TGCCAGTTTGGGACCAGTGCA…
…
>seq2_name field1 field2 field3 …
AGTCATGGAGCCGTTAGTACC…
TGCCAGTTTGGGACCAGTGCA…
…
```

A FASTA file can contain multiple DNA, RNA or protein sequences (but it may contain zero or just one sequence, too; a good bioinformaticist should always prepare for these special cases). Each sequence starts with a sequence name header line indicated by the greater-than symbol >, followed by the sequence name. Multiple data fields describing the sequence can optionally follow the sequence name **on the same** header line. Fields are separated from each other and the sequence name by a *delimiter* character. The delimiter can be a vertical bar `|`, a space character, a TAB key, or something else. Although delimiters can vary among FASTA files, they are the same for all sequences within the same file. Sometimes you just have to look into the FASTA file to figure out what is the delimiter.

Following the header line, multiple (but could also be zero!) sequence lines define the sequence. There is no real limit on how many sequence lines can follow each header line, or how wide is each sequence line (generally ≤70 characters but some output lines can be thousands of character long!) For genomic sequences, sequence lines can run in the tens of thousands! Sequence lines will continue until either the end of the file (if it's the last sequence in the file), or the beginning of the next sequence header line. Other than these basic formatting rules, FASTA files are very flexible, and as such, it may be hard to read them into your Perl code correctly. In this project, we will practice typical FASTA file processing tasks. To learn more about FASTA file formats, Google it.

Starting with a sample Perl program

To help get you started, a sample Perl program has been provided. This program will read a FASTA file, combine its sequence lines, and output each sequence. Download the "project_2_begin.pl" program from Blackboard Learn and run it on the "lucy.seq" FASTA file also provided, to see what output it produces. Here is a brief explanation how this sample program works:

- It creates a variable $count to count input sequences, and a variable $seq to collect sequence lines
- It scans input file one line at a time using the diamond operator <>, which stores each input line into the default $_ variable (this is a system variable that you do NOT need to declare).
- The scanning ends when there is no more input line, and the program reports the total sequence count in the standard error output STDERR (this isn't an error message but it is customary to separate informative text messages from output data using the special STDERR output).
- Inside the program it decides if each input line is a header line (by matching to the pattern /^>/), or a normal sequence line (if not matching).
- For header lines, it will print out the header, increase the sequence counter, and reset the sequence variable $seq to an empty string. But before doing all these, it will output the $seq if it's not empty.
- For sequence lines, it will **chomp** away the trailing newline character on each line, and then concatenate the line (which is in the $_ variable) to the $seq variable.

Try to understand the sample code as written and described above. Once you feel that you understand the code well enough, proceed to work on the following project jobs.

Job 1: what does chomp do?

The **chomp** function removes the trailing newline character on each line that is stored in $_. If you don't use it, the newline characters will be preserved. Copy the "project_2_begin.pl" program to a new program file "project_2_job1.pl", comment out (by adding a # symbol) or remove the **chomp** function call, run your new program on "lucy.seq" and redirect your output to a file "job1_output.seq". You can then run the Linux **diff** command to compare the output from your modified job1 program to the original "lucy.seq" input file to see how they differ. Save the **diff** command output to another file "job1_output_lucy_diff.txt". Submit this file as your job1 result.

Job 2: what does $seq="" do?

Now copy "project_2_begin.pl" again to another program filename "project_2_job2.pl", comment out or remove the line that shows `$seq="";` and run the new program on “lucy.seq”. You may wish to send the output to a paging program such as **less**, so you can see what is wrong when you remove that sequence variable reset step. Copy and paste only the *last complete sequence* from your output and save that to another file “job2_output_long_sequence.txt”. Submit this file as your job2 result.

Job 3: What is the bug in “project_2_begin.pl”?

If you have been paying attention to the output we have seen so far, you should have noticed that “project_2_begin.pl” seems to be inconsistent. It reports 10 input sequences but only prints out 9 sequences; it seems that it only prints the header of the last sequence but misses out its sequence string!

Copy the “project_2_begin.pl” file to another program “project_2_job3.pl” and fix the bug there. If you can fix the problem, your output should now contain 10 complete sequences. Submit “project_2_job3.pl” as your job3 result.

Hint: it seems that “project_2_begin.pl” prints each sequence output *after* it sees the beginning of a new header line. This is reasonable since otherwise the Perl program does not know if all the sequence lines belonging to a sequence have been collected. However, another condition, the end of the input file, also ends the sequence collection but that condition was overlooked in the code.

Job 4: Break up a sequence into multiple lines

The output from “project_2_begin.pl” contains a single long line for each sequence. If you compare its output to the original “lucy.seq” file, you will notice that there is no line breaks in the output sequences. Is this better or worst? It depends. If your purpose is to process each sequence (e.g., to trim it or reverse-complement it; see Job 5 below), you will *need* the whole sequence to be in a single variable. However, when it comes to outputting FASTA sequences, it is recommended that long sequences be broken up into multiple sequence lines not longer than 70 characters each, as we have described in the **Introduction**.

The following subroutine is provided to do just that task. Copy your bug-fixed version “project_2_job3.pl” into “project_2_job4.pl”, modify your code by including and calling the following subroutine in the right place(s) in your code, test run your code on “lucy.seq” to make sure it works, and then submit “project_2_job4.pl” as your job4 result.

```
my $line_width = 70;
# this subroutine prints $seq out, broken into $line_width pieces
sub print_seq {
  for (my $i=0; $i<length($seq); $i+=$line_width) {
    print substr($seq, $i, $line_width), "\n"; # output FASTA lines
  }
}
```

Job 5: Sequence trimming

The “lucy.seq” file contains coordinates that define good quality region for each sequence on its header line. For example, the following header says that between bases 30 and 491 (starts from 0), the sequence quality is good:

```
>ATIEA01TF 0 0 0 30 491
```

We wish to trim away the bad quality regions on both ends of a sequence and retain its good quality region in the output. Copy your completed “project_2_job4.pl” program into “project_2_job5.pl”, and modify it so it now trims the sequences before it output them. Test your program to make sure it is trimming correctly, and then submit it as your job5 result. The following are hints how this can be done:

- When matching a header line, you can also match to the left and right boundary values and store them into $left and $right variables as follows:

  ```
  if (/^>\w+ 0 0 0 (\d+) (\d+)/) { # match a header line?
    ($left, $right) = ($1, $2); # yes, save the coordinates
  ```

- Subsequently, when you have collected a sequence for output, you can trim it using the **substr** function as we have seen in Job 4 for breaking up sequences.
- Note that the **substr** function takes three parameters, 1) the sequence to trim, 2) the start position, and 3) the *length* of the subsequence to take out. Therefore, you must convert $left and $right to $left and length in order to use **substr**. Could you figure out how to do that?

For questions about this project, please post them to Blackboard Learn -> Discussions -> Project Discussion area. Collect all required job results from this project to a folder, say “Project_2_jobs”, archive the directory into a zip file, and then upload the zip file to Blackboard Learn assignment submission. If your results are not exactly matching the expected results described above, include a README.TXT file in your upload to explain why your results differ from the project description (e.g., you failed on certain jobs, or you added more features, etc.)