Parsing assignment
Due Friday, 2013 Oct 11

UCSC BME 205 Fall 2013
Bioinformatics: models and algorithms

(Last Update: 14:11 PST 17 November 2013)

Purpose

The purpose of this assignment is for you to build re-usable parsers for reading some of the most common sequence formats, so that you will have them available for future assignments.

Two concepts are important here: creating reusable modules and what the popular sequence formats are.

The program for this assignment should be built starting from the template created in the first assignment. Note that it need not be the template that *you* created for the first assignment, but if you start with someone else's template, they need to be credited in writing—their name should still be in the source code!

The formats: FASTA and FASTQ

The two most popular formats for sequences are FASTA (in many variants) and FASTQ (with two variants). You will need to do input and output of FASTA and FASTQ files for this assignment. As always with bioinformatics programs, you should be tolerant of minor variations in the format on input, but very strict on output.

There are four pieces of data associated with a sequence in a file:

1. The name of the sequence, which usually is a string that contains neither spaces nor commas, but may have other special characters in it (NCBI loves using the "|" character in sequence names). For the purposes of this assignment, the name should be considered atomic, and not broken into smaller pieces.
2. Extra information about the sequence. This is a string which may have any information in it, but is usually prohibited from having newlines. Again, some programs structure this extra information (in various incompatible ways), but for this assignment, it is an uninterpreted string.
3. The sequence itself, consisting of either nucleotides or amino acids. Various wild cards are allowed, the special symbol "*" for stop codons and the special symbol "-" for gaps (which are interpreted differently in different programs).
4. Sequence of quality information (Q-values), generally integers that are rounded approximations to $-10 \log_{10}$ P(character is wrong)

FASTA format

FASTA format has ID lines starting with ">" in the first column and sequence lines. The ID extends from immediately after the ">" on the id line to the first white space or comma. The extra information is everything after the id and separator on the id line. Sequence lines contain the sequence, and are not allowed to contain ">" characters, but may have white space that is ignored.

Rather than giving you a precise definition of the FASTA format, I point you to the OK description on Wikipedia's FASTA format page. Go read it now. On that page, they say

> *Some databases and bioinformatics applications do not recognize these comments and follow the NCBI FASTA specification.*

You should follow the NCBI FASTA specification for output of FASTA, since most programs don't understand the ';' comments. But one of NCBI's suggestions is both good and bad advice. They recommend

> *It is recommended that all lines of text be shorter than 80 characters in length.*

Limiting lines in a file to a maximum length helps avoid problems with buffer overflow in programs where a fixed-length array is allocated to hold an input line. Unfortunately, it is not possible to keep all lines to below 80 characters. Since the extra information on the ID line may be arbitrarily long, and only one ID line is permitted per sequence, it is not possible to keep the ID line to 80 characters.

Some FASTA-derived formats, such as UCSC's A2M format extend the alphabet (adding "." in A2M), and some programs do not accept parts of the standard format (such as "*"). Your first FASTA parser should accept any FASTA file that follows the NCBI conventions, and should properly ignore white space within sequences. It should also ignore any extraneous characters (like the "." of A2M format). You can add options, if you want, to modify what alphabet is considered acceptable.

I have occasionally encountered a badly written program that can't handle having white space or newlines breaking up a sequence, in which case you need to put out the entire sequence on one line. There should probably be a command-line option in any program that outputs FASTA to choose between line-length limitations and sequence-on-a-single-line.

Note that the FASTA format provides name, extra data, and sequence information, but does not provide quality information. When quality information is needed with FASTA files, two files are used. One is the straight FASTA file with names, extra data, and sequences, and the other is a file with ">" lines having the same ids in the same order, but having white-space-separated integers instead of bases or amino acids. Note that the number of integers is expected to be identical to the number of bases or amino acids for the corresponding sequence in the sequence file.

FASTQ format

The FASTQ format is used only with nucleotide alphabets, unlike FASTA, which is used with both nucleotide and amino-acid alphabets. Note that the nucleotide alphabet may include bases other that A, C, G, and T (like "H", which means "anything except G").

The Wikipedia article on FASTQ format is a pretty good description of the format.

Some important points to note:

- The FASTQ format is poorly designed, since the punctuation characters "+" and "@" are also legal quality values.
- The FASTQ format consists of records, each of which has the data for one sequence and the corresponding quality information. If the line that introduces the quality data has an identifier, it should match the identifier on the id line for the nucleotide sequence. Your program does not need to check this, but you could add it as an extra feature, once everything else is working correctly.
- On input, you need to be able to handle nucleotide and quality sequences that are on multiple lines,

even quality sequences that have "@" or "+" characters at the beginnings of the line. Note that spaces, tabs, and new lines are all not legal characters for nucleotides or quality values, and so should be ignored. The key to getting this right is that a quality sequence must be exactly the same length as the preceding nucleotide sequence, and the "+" and "@" characters can't appear in nucleotide sequences, so a "+" character encountered while reading a nucleotide sequence must signal a switch to the quality sequence (after reading the optional id after the "+").

- On output, FASTQ sequences should *not* have line breaks in either the nucleotide sequence or the quality sequence, so that other programs with less careful parsers won't break. I've seen some break anyway, since the first quality character can be a "+" or "@", and some parsers are so fragile that they can't handle that case even when the nucleotide and quality sequences are single lines.

- There are two different ways to convert Phred quality scores to FASTQ quality letters: Phred+33 and Phred+64. Defaults for input and output should be Phred+33 format, but Phred+64 should be available as an option for either. Both formats convert a small integer that represents the quality of a base to a single ASCII character. They can be thought of in Python as `chr(qual+33)` and `chr(qual+64)`. The conversion in the other direction is `ord(c)-33` or `ord(c)-64`.

  The usual interpretation of the Phred quality number is as $-\log_{10}(\text{Prob(error)})$, which really only makes sense for quality values bigger than 1, since if we know nothing about a base, we have about 3/4 chance of being wrong about it. Despite this, most programs expect 0 as the quality for unknown bases (though Illumina files from versions 1.5 through 1.7 used 2, encoded as 'B' in Phred+64, for unknown bases).

  Solexa files (Illumina version 1.0) encoded $-\log_{10}(\text{Prob(error)}/(1-\text{Prob(error)}))$, which actually makes more sense than the standard interpretation, but standards prevail over sense, so Illumina version 1.3 onward uses $-\log_{10}(\text{Prob(error)})$, as do other sequencing platforms.

  Although you can use a dict look-up table to translate FASTQ quality values to integers, it is probably better to use the ord() function and to use chr() to covert quality values to characters.

- It is possible to guess from a FastQ file which of several encoding schemes is used, by looking at the distribution of characters, but that sort of cleverness is not expected in this assignment—you can assume that the user has given you the correct choice.

The task

Create three generators in a module that you can include from other Python programs:

- read_fasta, for yielding sequences from a fasta file passed in as a file-like object.
- read_fastq, for yielding sequences with Phred-score quality values from a fastq file passed in as a file-like object.
- read_fasta_with_quality, for yielding sequences with Phred-score quality values from a pair of files (one fasta, one similarly formatted but with white-space-separated numbers instead of amino acid or nucleic acid sequences. It would probably be best if this parser used the read_fasta parser directly, so that bug fixes there would automatically propagate.

  The module should be a separate file whose name ends with ".py", so that it can be included in your main Python program.

  Your generators should accept arguments that affect how they parse (for example, which characters are kept in sequences, whether other characters are ignored or treated as errors, whether case is

preserved, and so forth). You will probably want to extend and modify these options for different assignments over the quarter, so it is more important now just to get the parsers working than to dream up lots of complications.

You may choose any convenient internal format for yielding the sequences, but you should be sure to include the sequence id, any comments that are on the id line, and the sequence itself. Three obvious choices are tuples: (id, comment, sequence) or (id, comment, sequence, quality sequence), classes with members id, comment, sequence, quality, or dicts: {'id': id, 'comment':comment, 'sequence':sequence, 'quality':quality}.

Using classes is probably the most Pythonic choice, but tuples are easier to implement and may be easier to unpack in the for-loops that the generators will be called from. I generally use tuples for these parsers, because of the easy of unpacking them.

Write a main program "fastq2fasta" that can accept a FASTQ file and convert it either to FASTA or to a pair of files FASTA and quality.

Write another main program "fasta2fastq" that can accept a FASTA plus quality file and convert the pair to FASTQ.

I will test your programs with the following commands:

```
fastq2fasta < foo.fastq  > foo.fasta
fastq2fasta --33 --in foo.fastq --out foo.fasta --qual foo.qual
fastq2fasta --64 --in foo.fastq --out foo.fasta --qual foo.qual

fasta2fastq --64 --in foo.fasta --qual foo.qual --out foo.fastq
fasta2fastq --33 --in foo.fasta --qual foo.qual --out foo.fastq
```

If fastq2fasta does not include a --qual specifier (as in the first example above), then the quality information is just discarded. This is a fairly common use case, when converting data for programs that can't use quality information.

The --33 and --64 options always refer to how quality information is encoded in the fastq files. The .qual files paired with .fasta files are always encoded as decimal numbers, not with the single-character encodings of the fastq format.

You may find it useful to do other variants (merging datafiles, automatically generating extensions, detecting errors in the input, ...), but the core of this assignment is producing reusable sequence parsers and sequence output in a couple of the most common formats. Get the core functionality working completely before you try adding bells and whistles. Core functionality and readable, simple code is more important than error checking or reporting. Note that empty files and empty sequences are legal (even common) use cases, and should not cause crashes.

Because the specification here is deliberately vague about the precise output format to use, I will be checking output by using your program to convert from one format to another, then my program to covert back. I will expect the results of that conversion to be identical to doing both conversions with my programs. If your programs produce legal files with the correct data in them, I should be able to read them and get identical results, even if the formatting choices are different from the choices I make.

There are a few fastq test files for your use in http://users.soe.ucsc.edu/~dbernick/bme205/ Note: I believe that 7_1.fastq is deliberately bad. I've also provided tiny.fasta, tiny.qual, and tiny.fastq files

for quicker testing (tiny.fastq is a Phred+33 file and has the same data as tiny.fasta and tiny.qual, but may stress some fastq parsers).

Note: the following text is borrowed from David Bernick's assignment in Fall 2011, with some modifications:

Think carefully about boundary conditions: for example, an empty file, an empty string as the id, or an id line with a zero-length sequence (all of which are legal inputs). Think also about illegal inputs, like a mis-ordered FASTQ file (ID lines of sequence and quality lines are both specified but not equal), or a FASTA file that contains extraneous characters (like numbers in nucleotide sequence). Your programs must be able to handle any legal input, but its behavior may be undefined on illegal inputs.

The program should also report to stderr (not stdout—we don't want to contaminate the output files) any anomalies that are found, and the assumptions that are made about the input file. There are no specific requirements for what you must report, but some things to consider include the following: Is it a PHRED+33 or PHRED+64 file? How many records were found? How many bad records were found (missing IDs, duplicate IDs, wrong number of quality scores, sequence without quality or quality section without sequence, ...)?

Start with no reporting of extra information or errors, and gradually add information and error checks as you have time or see the need. It is possible to get tangled up in doing too much error checking initially, when your focus should be on clean, readable code.

Any error messages to stderr should be fairly terse. You probably want to report only once errors that are likely to be in many sequences (like quality score out of range for PHRED+33 or illegal character in sequence). I would like to see short informative messages like

```
ERROR: '@' not in first column for 'HWI-ST611_0189:1:1101:1237:2140#0/1'
ERROR: not FASTA format, sequence line before first id line.
WARNING: input has no sequences.
WARNING: input has 3 empty sequences.
```

Note: it would probably better to give the names of the three empty sequences. Line numbers can also be useful in error messages, though they require a bit more forethought in designing your parser —look up the Python "enumerate" function.

I generally reserve "ERROR" for serious problems that should shutdown a pipeline and prevent further processing until the problem has been fixed manually. I use "WARNING" for things that may be legal, but are rare and ought to be checked, as they may result from a file having been contaminated or truncated.

things to change next year

Half the students still managed to choose a bad internal representation for quality that was not easily manipulated and that was different for fasta+qual and for fastq inputs. I should change the task so that students have to interpret the quality scores. Perhaps a variant of end-trimming would be best: reading in a fastq file in Phred33 or Phred64 or fasta+qual pair of files and outputting a fastq file in Phred33 with the longest subsequence having no more than n bases with qual score less than k. (Also have them doctor the comment to start with newlength/oldlength.)

I need to add checks for handling of upper- and lower-case characters, as fasta parsers that succeed

with current tests may still fail for Markov model assignment, where there are mixed-case sequences.