ITE315 Module 3 Part D - Programming in Perl: Case Study - Perl and Bioinformatics

Athens State University

Contents

1 Perl and Bioinformatics

Bioinformatics

- **Bioinformatics**: the science of collecting and analyzing complex biological data such as genetic code
  - Biologists in the room don't scream at me... that's what is in Mr. Webster's dictionary
- Interdisciplinary field that focuses on the use of computer science and mathematics in understanding biological data

Sequence Analysis

- The part of bioinformatics most commonly associated in the popular press is sequence analysis
- At this point, we have the DNA sequences of thousands of organisms decoded and stored in databases
- Analysis of this data uses lots of CS concepts: string processing, pattern analysis, data mining, and machine learning
- Scripting languages such as Perl and Python have become the programming tools of choice for this work

2 Representing Sequence Data

Just What Exactly is DNA?

- From a chemical standpoint, DNA is a polymer composed for four molecules
  - These are usually called bases or nucleotides
  - The four bases: adenine (A), cytosine (C), guanine (G), and thymine (T)

Just What Exactly is DNA?

- In cells, DNA usually appears in a double-stranded form with the strands wrapped around each other in a double helix shape
- Stands of the double helix have matching bases, known as base pairs
  - An A on one strand is always opposite a T
  - A G is always paired with a C
- Strands also have orientation:
  - One end of the strand is noted as the 5'-end ("5-prime")
  - The other end is the 3'-end ("3-prime")
  - Nucleotides always connect the 5' end of a strand to a 3' end of another strand
  - DNA is transliterated to RNA in the 5' to 3' direction

Representing DNA in Strings

- The structure of DNA means that we can represent its structure as string of symbols
- Each symbol represents one of the bases
- The sequences is stored in the string from left to right with the 5'-end on the left

```
$DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';
```

3 Now Let's Do Some Stuff

Transcription of DNA to RNA

```
#!/usr/bin/perl
use v5.010;
use strict;
use warnings;
# Transcribing DNA into RNA
# The DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
# Print the DNA onto the screen
print "Here is the starting DNA:\n\n";
print "$DNA\n\n";
# Transcribe the DNA to RNA by substituting
#  all T's with U's.
$RNA = $DNA;
$RNA =~ s/T/U/g;
# Print the RNA onto the screen
print "Here is the result of transcribing the DNA to
RNA:\n\n";
print "$RNA\n"; p
```

Here's where we start to understand why regular expressions are so important. In this case, we can do transcription simply by using a regex to swap out all of *T* bases with *U* bases in the string!

Reverse Complement

```
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$revcom = reverse $DNA;
$revcom =~ s/A/T/g;
$revcom =~ s/T/A/g;
$revcom =~ s/G/C/g;
$revcom =~ s/C/G/g;
print $revcom\n";
```

So: here's a question one for the biologists in the room. Is this a correct algorithm for doing reverse transcription?

Think substitution!

Reverse Complement

```
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$revcom = reverse $DNA;
$revcom =~ tr/ACGTacgt/TGCAtgca/;
print $revcom\n";
```

The first attempt to calculate the reverse complement failed. Each base in the string was translated as a whole, using four substitutions in a global fashion. Another way is needed. You could march though the DNA left to right, look at each base one at a time, make the change to the complement, and then look at the next base in the DNA, marching on to the end of the string. Then just reverse the string, and you're done.

Searching Sequences For Motifs

```
print "Please type the filename of the protein sequence data: ";
$proteinfilename = <STDIN>;

chomp $proteinfilename;

unless ( open(PROTEINFILE, $proteinfilename) ) {
  say "Cannot open file \"$proteinfilename\" ";
 exit;
}
@protein = <PROTEINFILE>;

close PROTEINFILE;
```

Searching Sequences For Motifs

```
$protein = join( '', @protein);
$protein =~ s/\s//g;

do {
 print "Enter a motif to search for: ";
 $motif = <STDIN>;

 chomp $motif;

 if ( $protein =~ /$motif/ ) {
    print "I found it!\n\n";
```

```
 } else {
    print "I couldn\'t find it.\n\n";
 }
} until ( $motif =~ /^\s*$/ );
```

Counting Bases

```
print "Please type the filename of the DNA sequence data: ";
$dna_filename = <STDIN>;

chomp $dna_filename;

unless ( open(DNAFILE, $dna_filename) ) {
 print "Cannot open file \"$dna_filename\"\n\n";
 exit;
}
```

Counting Bases

```

# Read the DNA sequence data from the file, and store it
# into the array variable @DNA
@DNA = <DNAFILE>;

close DNAFILE;
# From the lines of the DNA file,
# put the DNA sequence data into a single string.
$DNA = join( '', @DNA);
# Remove whitespace
$DNA =~ s/\s//g;
```

Counting Bases

```
# Now explode the DNA into an array where each letter
# of the original string is now an element in the array.
# This will make it easy to look at each position.
# Notice that we're reusing the variable @DNA for this
# purpose.
@DNA = split( '', $DNA );
# Initialize the counts.
$count_of_A = 0;
$count_of_C = 0;
$count_of_G = 0;
$count_of_T = 0;
$errors = 0;
```

Counting Bases

```
# In a loop, look at each base in turn, determine which of the
# four types of nucleotides it is, and increment the
# appropriate count.
foreach $base (@DNA) {
```

```
 if ( $base eq 'A' ) {
    ++$count_of_A;
 } elsif ( $base eq 'C' ) {
    ++$count_of_C;
 } elsif ( $base eq 'G' ) {
    ++$count_of_G;
 } elsif ( $base eq 'T' ) {
    ++$count_of_T;
 } else {
    print "Error - I don\'t recognize this base: $base\n";
    ++$errors;
 }
}
# print the results
print "A = $count_of_A\n";
print "C = $count_of_C\n";
print "G = $count_of_G\n";
print "T = $count_of_T\n";
print "errors = $errors\n";
```

4 Working With Downloaded Data

Addressing Downloaded Data

- Sequences are typically found in databases such as the GenBank database
- One uses tools like BLAST to find sequences and then download files in the GenBank format for further processing
- We can use Perl (or other scripting tools) to process this data

Loading a GenBank File

- Known in CS terms as "parsing the data file"
- Know the format, have to read the data, dealing with errors
- Remember... regular expressions are your friend

Loading a GenBank File

```
# declare and initialize variables
my @annotation = ( );
my $sequence = '';
my $filename = 'record.gb';
parse1(\@annotation, \$sequence, $filename);
# Print the annotation, and then
# print the DNA in new format just to check if we got it
okay.
print @annotation;
print_sequence($sequence, 50);
exit;
```

Loading a GenBank File

```
sub parse1 {
  my($annotation, $dna, $filename) = @_;
  my $in_sequence = 0;
  my @GenBankFile = ( );
  # Get the GenBank data into an array from a file
  @GenBankFile = get_file_data($filename);
  # Extract all the sequence lines
  foreach my $line (@GenBankFile) {
   if ( $line =~ /^\/\/\n/ )   {
                          # If $line is end-ofrecord
       last;              #break out of the foreach loop.
   } elsif ( $in_sequence) {
       $$dna .= $line;    # add the current line to $$dna.
   } elsif ( $line =~ /^ORIGIN/ ) {
       $in_sequence = 1;  # set the $in_sequence flag.
   } else {
                          # Otherwise
       # add the current line to @annotation.
       push( @$annotation, $line);
   }
  }
  # remove whitespace and line numbers from DNA sequence
  $$dna =~ s/[\s0-9]//g;
}
```

Loading a GenBank File

```
sub get_file_data {
  my($filename) = @_;
  use strict;
  use warnings;
  # Initialize variables
  my @filedata = ( );
  unless( open(GET_FILE_DATA, $filename) ) {
      print STDERR "Cannot open file \"$filename\"\n\n";
      exit;
  }
  @filedata = <GET_FILE_DATA>;

  close GET_FILE_DATA;
  return @filedata;
}
```

5 Key Points

Key Points

- Use of scripting as enabling technology for bioinformatics
- String processing and pattern matching