Programming Assignment 2 - Megabase-scale alignment

BIOINFO M260

Part A Due: Monday, January 30th, 2017, 11:59 pm
Part B Due: Wednesday, February 8th, 2017: 11:59 pm

This programming assignment is designed to expand your understanding of sequencing and the difficulty of mapping insertions and deletions.

Overview

In the first two programming assignments for this class, you will solve the computational problem of re-sequencing, which is the process of inferring a donor genome based on reads and a reference.

You are given a reference genome in FASTA format, and paired-end reads.

The first line of each file indicates which project that the data relates to. In the reference file, the genome is written in order, 80 bases (A's, C's, G's, and T's) per line.

The paired end reads are generated from the unknown donor sequence, and 10 percent of the reads are generated randomly to mimic contamination with another genetic source. These reads are formatted as two 50 bp-long ends, which are separated by a 90-110 bp-long separator.

Grading

SNP Score	No Credit	Full Credit
Undergrad	55	75
Grad	70	90

Indel Score	No Credit	Full Credit
Undergrad	3	13
Grad	15	25

Your total score will be the average of (Your Score - No Credit Score)/(Full Credit Score - No Credit Score) for both SNPs and Indels.

Project 2A

By January 30, you need to have run your code and have submitted it with a score that will yield some credit on the project.

Starter Code

Starter code for the project is available at `https://github.com/michaelbilow/BIOINFO_M260B`. It is strongly recommended that you use git for these programming assignments, and to set up your own account on github.com. The tutorial at `https://try.github.io/levels/1/challenges/1` is short and excellent.

Create a branch for you to work on PA1 with, for example with `git branch pa1 && git checkout pa1`. This will allow you to save your work and still be able to update your code from my master branch.

Go into the PA1 folder and run `python basic_aligner.py` If you are on the current master branch, this should produce output, however it takes about a half hour to run on my machine.

Read the content of PA1, and see if you can understand what it is doing. You should also look to see where your input/output is going to go. This will generate an alignment file in the data folder from which it was executed. Read over the alignment file and see how it works.

Run `python basic_pileup.py`. This will generate three files; a consensus sequence, a text file correctly-formatted for submission on the herokuapp site, and a zipped version of that file.

Download the data from CCLE, and edit the functions in `basic_aligner.py` and `basic_pileup.py`. The warmup sets include a donor genome (i.e. the genome from which the reads were generated), and an answer set that you can use You can submit your results as many times as you want to achieve a passing score.

I/O Details

`https://cm124.herokuapp.com/ans_file_doc` should handle most of your questions on reading and writing output.

Sequence Alignment

The trivial sequence alignment algorithm is to ”slide” the read along the reference genome. This is written simply as a for loop:

```
... get reads as input ...

for read in reads:
    for i in range(len(genome) - len(read)):
    mismatches = [0 if genome[i + j] == read[j] else 1 for j in range(len(read))]
    n_mismatches = sum(mismatches)
    if n_mismatches < 3:
    ...
    add this to the output
    ...
    break

.... construct consensus sequence ...
```

The statement above has some useful Python tricks; instead of iterating only over indices, Python allows you to iterate over objects. If you store your reads in an array called `reads`, you can get each one using the word `read`, rather than using many different indices.

The statement `mismatches = [0 if genome[i + j] == read[j] else 1 for j in range(len(read))` is called a list comprehension, which borrows from functional programming. Essentially, this code is the same as:

```
output = []
for j in range(len(read)):
if genome[i + j] == read[j]:
output.append(0)
```

```
else:
output.append(1)
```

However, the syntax is much cleaner, and if you can read it, you'll be a much better programmer.

Questions to consider

The genome from which the reads are generated has not only SNPs, but insertions, deletions, and repeated sequences that are not present in the reference.

What will these non-SNP mutations look like when you try to align them to the genome? Try writing it out on a piece of paper.

More generally, what is the "signature" of a SNP mismatch in the consensus sequence? What is the "signature" of an insertion or deletion?

Grading

Remember to submit your solutions to `https://cm124.herokuapp.com/upload` as a `.zip` file. You can submit as many times as you want without penalty.

You will be graded on your performance on the test set, which is under week 2 in CCLE. You also have two sets of data with solutions (also under week 2 in ccle), that should be helpful for building and debugging your algorithms. You can also submit your solutions for those datasets to `https://cm124.herokuapp.com/upload` to see how your solution is performing.

Undergrads will get full credit with a score of 45 on SNPs, and no credit for a score of 25 or below. Grad students will get full credit with a score of 60 on SNPs, and no credit for a score of 40 or below.