Series 3

Genomics and bioinformatics - Week 3

October 3, 2011

1 Learning objectives

The final goal of the programming exercises you receive each week is for you to be able to download some interesting data files from UCSC, format them, extract the relevant data from them and apply some statistics that you have seen in this course. At the end of the fourteenth week, you should be able to answer many different types of biological questions by fetching the data needed and using some existing software necessary for the analysis or writing the code necessary for the analysis yourself.

2 Today's exercise

We are going to create a contig from a bunch of reads. The reads have been specially created by the assistants for this problem and will be given to you on the Moodle website. Your goal is to generate the largest contig possible. We will consider a simplified ideal case where:

- The reads don't contain any errors.
- The reads have the same length.
- The reads are all from the same strand.
- No two reads are the same.
- The full chromosome of our imaginary species is small.

See the last section of this document to learn some useful Python tricks.

2.1 Hamiltonian path problem

1. Read the file `reads1.fastq` and load all the reads it contains inside a list. It may help you to take a look at `reads1.fastq` in your text editor first.
2. Write a function that takes two reads as input and returns first read extended with the second; if they don't overlap, return `False`. The function should also have a `min_overlap` parameter that specifies how many base pairs must match between both reads at a minimum for it to be considered an overlap.
   e.g. `overlaps(AATG,TGTT,2) -> AATGTT , overlaps(AATG,TGTT,3) -> False`
3. Using the the list of reads you just loaded and the function you just wrote, you can now generate a list of vertices and a list of edges of the overlap graph - i.e. the reads will act as the vertices and the overlaps as the edges. In our case, a edge between two reads exists if the first overlaps the second.

4. Copy the code from `findcycles.py` into your code (or import its functions).

5. Use the function `bruteforce(edges)` to get an Hamiltonian path from your graph. Deduce the largest contig possible, and display it on the screen.

6. Record the time it takes to compute: import the `time` library and use `time.time()` to get the current time.

7. The contig should be AATGTCGATT. If you get it right, you can try to build more reads and run it again.

8. Now try it on `reads2.fastq`.

2.2 Eulerian path problem (de Bruijn algorithm)

1. Write a function that for a given read returns a list of all the l-mers contained in the read, e.g. `subseqs(AATGCG,4) -> [AATG,ATGC,TGCG]`

2. Taking `l` = 3bp, and using the same reads library as before, generate the list of vertices and the list of edges as defined for the de Bruijn graph.

3. Spot the start and end vertices (the ones that appear only once in the edges), and add one vertex to close the graph.

4. Use the `hierholzer(nodes,edges)` function - that you copied or imported earlier - to find an Eulerian cycle in your graph. Deduce the largest contig possible, and display it on the screen. Record the time it takes to compute, and compare with the Hamiltonian case.

2.3 Useful Python features

1. Use list comprehensions:
   `[x+4 for x in [1,2,3]]`
   returns `[5,6,7]`
   `[(x,y[0]) for x in [1,2] for y in [[1,2],[3,4]] ]`
   returns `[(1, 1), (1, 3), (2, 1), (2, 3)]`

2. No need to add "`;`" at the end of your lines. You can do it though if you want to put several statements on the same line.

3. You can use `list(set(...))` to get only unique elements of a list. Unfortunately there exists no such built-in function in Python.

4. Use `dir(object)` to get all possible methods for this object. If you are using Ipyhon as advised, just type `object.` and hit the "tab" key.