Series 3

Genomics and bioinformatics - Week 4 - October 3, 2012

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.

- The reads are unique (no replicates).

- The full chromosome of our imaginary species is small.

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. Try to find the contig by aligning the reads by hand. The reads are AATGT, ATGTC, GTCGA, and CGATT.

2. Write a function that takes two reads as input and returns the overlap sequence. 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)` → TG , `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, an edge between two reads exists if the first overlaps the second.

4. Copy or import the `hamiltonian(nodes,edges)` function from the file `hamiltonian.py` to find an Hamiltonian path in the graph, and deduce the longest contig. It will also print the time it takes to find the path.

2.2   Eulerian path problem (de Bruijn graph)

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` = 3 bp, 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 (in our case, the ones that appear only once in the edges), and add one edge from end to start to close the graph.

4. Copy or import the `hierholzer(nodes,edges)` function from the file `hierholzer.py` to find an Eulerian cycle in your graph. Deduce the largest contig possible. It will also print the time it takes to find the path.

5. If you have time, it is interesting to try with more reads defining a more complex network. Then compare the time taken by both algorithms. The second should be much faster. *Note:* some pathological cases *could* cause the 'hierholzer' hand-made script to fail.

2.3   Useful Python features

1. 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.

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

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

4. You can use "list comprehensions":
   `[x+4 for x in [1,2,3]]`
   returns [5,6,7]
   `[x+4 for x in [1,2,3] if x>1]`
   returns [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)]
   `[x for y in [(1,4),(2,3),(3,5)] for x in y]` # note the inversion
   returns [1, 4, 2, 3, 3, 5] # there are simpler methods, of course