Bi 621 – Problem Set 6

Our goal with this assignment is to generate some measures of accuracy for whole genome assemblies and then use these measures to assay our success in several Velvet assemblies.

Part 0 – Write unit tests

1. Read through Part 1 to be sure you understand the details of what your algorithm should do.
2. Create a file called `Unit_test.fa` that contains a text file that you know will yield a specific answer when analyzed by your code from Part 1.
3. Create a second file called `expected_results.txt` that contains the expected results from your test.
4. After finishing your code in Part 1, run your unit test to ensure your code behaves as expected.
5. OPTIONAL – trade unit tests and results with 1 or more people in the class. See if you can pass their unit test.

Part 1 – Contig length distributions

1. Parse the `contigs.fa` file that is output by `velvetg`. Extract the FASTA ID lines as you parse the file (remember: these strings will begin with the ">" character).
2. You can use the sample data `contigs.fa` from Talapas to test your code.
3. Using Python regular expressions, extract k-mer length of each contig (in **red** below). In addition, extract the k-mer coverage for the contig (in **blue**). Assume a k-mer length of 49.

   `>NODE_11_length_3717_cov_19.315845`
4. Adjust the k-mer length to represent the physical length. Calculate the number of contigs, the maximum contig length, the mean contig length, and the total length of the genome assembly across the contigs. Calculate the mean depth of coverage for the contigs.
5. Calculate the N50 value of your assembly.
6. Calculate the distribution of contig lengths and bucket the contig lengths into groups of 100bp. So, all contigs with lengths between 0 and 99 would be in the 0 bucket, those with lengths between 100 and 199 would be in the 100 bucket, etc.
7. Print out the distribution.

```
# Contig length   Number of contigs in this category
0     0
100   5324
200   3345
300   1130
...
```

Part 2 – Velvet

In this part, we'll be assembling fosmids from goldeye fish (Hiodon alosoides) from quality trimmed Illumina PE100 reads.

First, install Velvet:

```
% conda activate bgmp_py3
% conda install velvet -c bioconda
% velvetg
```

Note that the bioconda recipe for building velvet (found here: https://github.com/bioconda/bioconda-recipes/blob/master/recipes/velvet/build.sh) specifies several compile-time options that are critical:

- `MAXKMERLENGTH=50`
- `OPENMP=1`
- `LONGSEQUENCES=1`

1. All your work in this section should be completed using the queuing system on HPC. (See Nick's lecture notes on HPC and https://hpcrcf.atlassian.net/wiki/display/TCP/How-to+Submit+a+Job to remind yourself how the queuing system works)
2. You can find the data here:

   ```
   /projects/bgmp/shared/Bi621/800_3_PE5_interleaved.fq_1
   /projects/bgmp/shared/Bi621/800_3_PE5_interleaved.fq_2
   /projects/bgmp/shared/Bi621/800_3_PE5_interleaved.fq.unmatched
   ```

   Please do not copy the data, but rather refer to its location in your script (PRO TIP: Use variables or symlinks to make your script more readable). Remember to NOT write to the group project directory.
3. Run Velvet on the dataset. You can find the Velvet manual here: https://www.ebi.ac.uk/~zerbino/velvet/Manual.pdf
   a. Our fosmid library is comprised of 50 fosmids, each (approximately) 40 Kb long. How many total nt should be in this fosmid library?
   b. Coverage is a metric applicable to total fosmid library size just as it is applicable to genome (estimated) size. Think of all of these fosmids combined as the genome we're trying to assemble. Calculate the expected coverage.
      i. NOTE: Reads are varying lengths due to adapter clipping and quality trimming. You must calculate the total number of nt in the input FASTQ files in order to calculate the expected coverage.
   c. Given the calculated coverage from 3.b and total fosmid library size, calculate the k-mer coverage.
   d. Optional: Noting that our reads have been quality trimmed, now calculate the *correct* coverage and k-mer coverage, explaining your work.
   e. Run velveth/velvetg with k-mer sizes of 31, 41, and 49.
   f. Use your code from Part 1 to collect assembly statistics on each result.
4. With a k-mer size of 49, adjust the coverage cutoff to 20x, 60x, and 'auto'. Again, assay your results with your code.
5. Finally, adjust the minimum contig length to 500bp (with k-mer size of 49 and coverage cutoff 'auto') and again, assay your results.

<u>Part 3</u> – Questions

1. Describe how the assembly changes with different k-mer values using the assembly statistics you have collected. How does the contig length distribution change?
2. How does an increased coverage cutoff affect the assembly? What is happening to the de Bruijin graph when you change the value of this parameter? How does velvet calculate its value for 'auto'?
3. How does increasing minimum contig length affect your contig length distribution and N50?

To turn in your work for this assignment, do the following:

Be sure to turn in your unit tests and expected results, your code, your output (mean, max, N50, etc.) and plots, as well as the answers to the questions above to GitHub.