Note: This assignment has two due dates. An initial progress report is due by the beginning of class on **Monday February 22** and the final project is due by the beginning of class on **Wednesday March 2**. If you choose, you may work with a partner on this assignment. You must submit your code and any other requested files as a zip file to Moodle.

1 Project Overview

In this project we will explore the process of genome assembly. Genome assembly of real DNA sequencing data is still a challenging area where many researchers are currently working. Thus, this assignment will focus on a simplified version of the problem using simulated data. As an overview, your job will be to simulate the process of sampling reads from a longer piece of DNA (similar to the output of a DNA sequencing experiment) and to assemble those fragments to get the best possible representation of the original sequence. You will also create a detailed write-up that describes and analyzes your program. In this project, we will consider some aspects of sequence assembly that make this problem difficult in practice, but will ignore others that would normally be considered when dealing with real data (e.g. double-stranded DNA). **Please read this whole document before beginning.**

This assignment is intentionally open-ended in a number of ways. You will have the opportunity to discuss the decisions you made (and why you made them in your writeup). In fact, while this is listed as a programming assignment, you should be aware that your final write-up is more heavily weighted than the code you turn in. So, please take the time to make your write-up clear and complete. In short, don't leave the write-up to the last minute.

Since this is an open ended assignment, don't be surprised if you try something, realize it doesn't work and have to scrap it and start fresh. If that happens, write about what you learned from it in your writeup. You will be given some specifications about how to construct your assembler, but many choices will be left up to you to decide. That said, a strong emphasis will be placed on your arguments for why you made the decisions that you did. **In short, you're main goal for this assignment is to clearly articulate what decisions you made, why you made them.**

Any data files mentioned below can be downloaded from here: http://www.cs.carleton.edu/faculty/loesper/courses/cs362_w16/prog3/. This directory also contains examples of any file formats mentioned below.

2 Project Specifications

Important: Make sure your code works on the lab machines, even if you developed your code on your own computer. Note: I do NOT expect you to be able to assemble "full size" genomes as that would require more RAM than is available on a lab machine. You should be able to handle data created for the `sample.fasta` file provided.

2.1 Shell Scripts

You will need to provide two shell scripts: `simulate.sh` and `assemble.sh`.

2.1.1 `simulate.sh`

`simulate.sh` will take in the following parameters (in this order):

- FASTA sequence file (string) - File containing a DNA sequence.
- Coverage (integer) - The coverage of the simulated sequencing experiment.
- Read length (integer) - The length of reads to simulate.
- Error rate (float) - The sequencing error rate (between 0 and 1).

`simulate.sh` will simulate n reads from the DNA sequence in the supplied FASTA sequence file where n is the number of reads indicated by Lander-Waterman statistics given the supplied read length, size of the genome and coverage. Reads should start at random indices in the genome (that is, they are uniformly distributed across the genome) and will all have the same orientation (that is, we are ignoring the existence of the reverse complement in DNA). **Handle edge effects in a manner of your choosing.** An error (incorrect nucleotide) should be introduced at sequenced every position in every read with probability equal to the specified error rate. Your program will output each simulated read as a separate line in a file in the current directory called `reads.txt`. For example, running:

```
$ ./simulate.sh example.fa 30 50 0.01
```

will create a file called `reads.txt` that may look like the following:

```
TAGCACCACTTCTGCGACCCAAATGCACCCTTTCCACGAACACAGGGTTG
TCCGATCCTATATTACGACTTCGGGAAGGGGTTCGCAAGTCCCACCCTAA
ACGATGTTGAAGGCTCAGGTTACACAGGCACAAGTACTATATATACGTGT
```

2.1.2 `assemble.sh`

`assemble.sh` will take in the following parameters (in this order):

- Reads file (string) - A file of reads, as output by `simulate.sh`.
- k (int) - the size of k-mer to use when building a De Bruijn graph.

`assemble.sh` will assemble the reads contained in the supplied `reads.txt` file using a de Bruijn graph approach with k-mers of the specified size. The program will output information about the assembled contigs in the following ways:

- The program will create a file in the current directory called `contigs.txt`. Each line of the file will contain the sequence of an assembled contig. This file may look something like the following (see the data directory for an example file) which shows 3 contigs.

  ```
  TAGCACCACTTCTGCGACCCAAGTTG
  TCCGATCCTATATTACGACTTCGGGAAGGGGTTCGCAAGTCCCACCCTAAACGATGTTGAAGGCTCAGG
  TTACACAGGCACAAGTACTATATATACGTGT
  ```

- The program will create a DOT file (see previous programming assignments for description of DOT files) of the de Bruijn graph. (Note: for large and complicated assemblies this file may not be very interpretable, but it should be useful for debugging with smaller examples. You may make a design decision to only output this graph for small graphs, but if you do so, you must discuss that decision in your writeup).

3 Simulation Program

Your program is only as good as the data you use to test it. For many of the questions in computationally biology that are interesting to study, there is no straightforward way to validate results obtained via computational approaches. Thus, it is common to "validate" through the use of simulated data where the "real answer" is known. Your final project will need to include/address the following:

- An overview of your simulator and further details about any design decisions that you made (e.g How do you handle edge effects? In what ways does your simulator create data that is similar to or different from real data?).
- You must submit a dataset you created that you consider **easy** to assemble (NOTE: easy does not mean trivial!). You must submit the FASTA file for the original sequence, the `reads.txt` file (you might want to rename it something meaningful) produced by your program and a list of the parameters used to create the dataset.
- You must submit a dataset you created that you consider **hard** in difficulty to assemble. You must submit the FASTA file for the original sequence, the `reads.txt` file produced by your program (you might want to rename it something meaningful) and a list of the parameters used to create the dataset.
- Please include descriptions of how you created the above datasets. For example: Where did you get the original FASTA file (there are many online tools for creating random DNA sequences or downloading real DNA sequences)? How did you pick the read length? (Don't just say you chose randomly!) How did you decide on an appropriate error rate? Do your datasets reflect real sequencing data - why/why not? Why is this dataset easy/hard to assemble? Etc.

4 Assembly Program

You will write an assembly program that includes the following high level steps:

1. Read in the supplied set of reads and construct a De Bruijn graph using the specified size of k-mers.
2. Simplify the graph (for example collapse strings of vertices connected by a unique path, or remove likely errors, etc.).
3. Identify the best set of contigs using an Eulerian-type approach such that every path that represents a unique read is represented in some contig).

How you choose to address the above requirements is totally up to you. But, your final write-up should include a detailed description of the decisions you made and how your program does (or doesn't) address the above requirements. You should include some analysis of how well your program accomplishes these tasks and how good of an assembler it is. This might include graphs or charts, or detailed explanations of the cases it does well on and when it fails.

You may look for inspiration on how to design your assembler from other places including real assemblers. For instance, on such "real" assembler is described in this paper paper. Other sources of information are also fair game - just make sure to cite them in your write-up.

5 Progress Report Due Monday 2/22

This is a chance for you to report progress on your simulator and assembler and ask for feedback. This will be worth **10 points** (2 for each of the following 5 deliverables) and you will likely receive all points if you turn in the following as single zip file to Moodle:

- A detailed plan for how you plan to design and build your simulator and assembler. (Note: much of this text may be useful for your final report).
- Functional code for your simulator (it doesn't have to be perfect, but you should be able to create at least some datasets). This should include a README with any necessary information for compiling and running your code.
- A first draft of your easy dataset.
- A summary of the implementation status of your assembler. I'd say you are in a good place if at least some portion is functional (perhaps reading in the reads file and partial creation the De Bruijn graph).
- A list of questions or problems that you are currently working on addressing.

6 Final Project Due Wednesday 3/2

Please hand in to Moodle a zip file containing all of the following:

1. All your source code.
2. The scripts to run your code as described above.
3. A README that explains the following:
   - How to compile your code (if necessary).
   - A description of any known bugs.
4. Your easy and hard datasets (FASTA file and reads file).
5. A PDF write-up that includes the following:
   (a) A detailed description of your simulator and answers to any questions posed earlier in this document about the simulator.
   (b) Descriptions of your easy and hard datasets as explained in a previous section of this document. Answers to previous questions about how your created your simulated data.
   (c) A detailed description of your assembler and answers to any questions posed earlier in this document about the assembler (and the decisions your made when designing it).
   (d) Using the FASTA file called `sample.fasta` in the data directory for this assignment, create a set of reads using the following parameters: coverage = 12, read length = 50, error rate = 0.01. Use your assembler to create a set of contigs for these reads. Turn in your reads file and the corresponding contigs file created by your assembler. Report what parameters you used to create the assembly and some analysis about how good of an assembly you think your contigs are. Can you get better results by changing your simulation parameters? Here is a good chance to convince your audience that your assembler is making good decisions!

7 Grading

The progress report will be worth 10 points (grading for that is discussed in that section of this assignment) and the final project will be worth 40 points and will be assigned according to the following categories.

- **Simulator program (5 points)** - We will asses whether your simulator functions according to the program specifications.
- **Assembler program (10 points)** - We will asses whether your assembler functions according to the program specifications. So please make sure to thoroughly test your program and ensure that you consider any edge cases.
- **Simulator description (3 points)** - Do you include complete, logical and well reasoned description of your simulator and any design decisions made?
- **Simulated datasets (5 points)** - Do you include the required datasets along with complete, logical and well reasoned descriptions of how they were created?
- **Assembler description (12 points)** - Do you include complete, logical and well reasoned description of your assembler and any design decisions made?
- **Data analysis (5 points)** - Do you include the required data analysis? Do you build a concrete argument about the quality of the resulting assembly?

8 Optional

There are a number of additional features that you may add to your simulation and assembly programs. You may receive up to 10 extra points for implementing some of them. If you choose to implement extra features, please make sure to discuss them in your write-up. Here are a few ideas of extra features you could implement:

- Improving the visualizations created by your assembler can often times be useful. In the DOT file of your De Bruijn graph highlight (with color) the paths in the graph that correspond to your assembled contigs. You could also make your program output a different type of visualization called a contig map where reads are "aligned" to their corresponding place in each contig. For instance, the output of such a file for a single contig `GTGGACCTAA` and 4 reads might look like the following. (This type of output really useful for debugging).

```
G T G G A C C T A A
G T G G
  T G G A C
      G A C C T A
        C C T A A
```

- Add functionality to your assembler to better handle repetitive regions of the genome.
- Make your simulator and assembler handle double stranded genomes (so, reads may come from either strand).
- Modify your simulator and assembler to create and handle errors where incorrect nucleotides have been inserted or deleted in a sequenced read.
- Anything else you think of that would be a good improvement.