(a)

I downloaded the genome from *Ca d da Ca a R dd* from the NCBI database, formatted as a Fasta file. I decided to use this genome as my running example upon recommendation from our textbook. In chapter Four of the textbook, focused on genome assembly, the authors recommend attempting to solve the problems only with simulated data or with a small genome, to avoid problems related to repeats and subsequent bubbles in the de Bruijn or Eulerian graphs.

I did not manually modify the Fasta file.

Since the information that I needed to use was not an actual genome but rather a sequence of k-mers or of re

Evaluation: I was able to solve this problem and results were checked in Rosalind.

3

I created one program to solve this problem: eulerianCycle, which both read a txt file and produced a list of edges, or trajectory for the graph.

3

I reused readFile and adjList and created a program stringR in order to solve this problem.

input: list of k-mers (Pattern)
output: reconstructed sequence (string)

input: adjacency list in the form of a dictionary (prefix/suffix)
output: reconstructed sequence

The program consisted of two FOR loops. The first FOR loop went through the list of suffixes and made a list in which every element in the Dictionary values was of type string. The second FOR loop checked every prefix against the suffix list. The prefix that did not appear in suffix list

I incorporated kmerComp to this program.
input: string (text), integer k, integer c (number of copies I want to make of the k-mer composition list)
output: dictionary of reads, in which every read is of length k, some may be repeated, and some may be missing.
The program runs through a FOR loop ‘c’ number of times. [This replicates the amount of copies one would have of a given DNA sequence for the reads to be produced]. A second FOR loop is embedded, of the average length of the sequence divided between the length of k-mers (we want to replicate what would happen if for each of the copies ‘c’, the sequence was fragmented at random points – i.e., producing not consecutive k-mers with overlap – produced by a sliding window – but instead fragments or reads with the possibility of missing areas or repeats. This average number (no) is the amount of k-mers that are selected and appended to the dictionary every time(c times) that the program runs through the dictionary of k-mers.

The reads dictionary was longer than the k-mer dictionary.

Example from k-mer dictionary:

```
0: 'ATGAATAATATTTTTGCAAAAATAA',
1:  'TGAATAATATTTTTGCAAAAATAAC',
2:   'GAATAATATTTTTGCAAAAATAACT',
3:    'AATAATATTTTTGCAAAAATAACTG',
4:     'ATAATATTTTTGCAAAAATAACTGC',
```

Example from reads dictionary:

```
0: 'TTTTTTTTTTTAAAAAAAAAAATAT',
1: 'CTAATAGAAAAATAATTTTTTATTT',
2: 'GAACAAAATGATATAAAAAAAATTA',
3: 'TATGTGCTGGGACTTTTATTAATTC',
4: 'TTTAATTTAACAATGGAAAAACAAA',
```

,

Once the two dictionaries are produced, I pass them through the programs adjList and stringR, already explained. AdjList worked for both k-mer and reads dictionary, but stringR could only be applied to the k-mer composition. Here is a fragmentary example:

```
ADJACENCY LIST FROM K–MERS:

'TTTTGTGTTGGAAAATAATGATTT':
 'TTTGTGTTGGAAAATAATGATTTA,

'TTGCAGGAATAAATGCAGCTAGAA':
 'TGCAGGAATAAATGCAGCTAGAAA
```