Jacklyn Sun
December 18, 2020 Project 1: Genome Assembly using De Bruijn Graph

Running My Code
Command Line
    python script.sh
Ubuntu Docker: I followed this link
https://www.geeksforgeeks.org/how-to-run-a-python-script-using-docker/
    first build docker container:
    run docker: sudo docker run -it jackie script.sh
    run line: ./script.sh, need to add #!/usr/bin/python at header to script.sh header file
User Input:
    -commented in code but manually need to change X, Y, Z, and genome string
    -for my testing purposes I left X = 3, Y = 2, Z = 2, and genome as
"TAATGCCATGGGATGTT"

Problem:

The problem I tried to solve for this project was given a random human genome, how does the k-mer size, repeat size, and number of repeats affect our re-construction of that genome. From class, we learned how to first construct a De Bruijn graph given a genome and desired k-mer length by "chopping" the genome into k length substrings. Then we had to create a pair of nodes for each k-mer such that the prefix was the starting node with k-1 nucleotides and the suffix was the endinging node with k-1 nucleotides. The directed edge from the prefix to the suffix node represented the k-mer. Then, we connect all the pre nodes to suffix nodes and create the De Bruijn. We can reconstruct the genome by finding a Eulerian path in the De Bruijn graph, but of course we know, with shorter K-mer sizes and more repeats, our graph becomes more "tangled" and can have a more difficult time re-constructing the correct original genome.

Approach:

1. I randomly generated a genome of 1000 nucleotides using the function randGenome()
2. I created a class printGraph, which, given a string representing the genome, outputs the graph representation of the De Bruijn graph as a pdf file. I had to import and install Graphviz, a python library, in order to output this pdf file. All of my graph figures below are outputs of various genomes I tested from this function.
3. I created a class DeBruijnGraph which given the genome, X, Y, Z:

a. First we restructure the genome to insert Z number of repeats given length Y substrings by picking the first Y elements of the genome and inserting copies of that substring into my original genome in the functions repeats()

b. Next given the genome with Z number of repeats with length Y substrings, we will construct the graph by creating a DeBruinGraph object which in the constructor creates the graph

c. we iterate through the string, create the nodes and map the nodes such that the edges represent the path from one node to another by storing it in a dictionary g, and count the number of in edges and out edges for each node, so that later check for each node if it is balanced, semi-balanced, or neither

d. Then we call our function eulerianWalkOrCycle(self): it first checks that our graph is eulerian so that a path can be found using the information earlier to tally each node's in and out degrees and whether they are balanced, semi-balanced, or neither.

e. eulerianWalkOrCycle(self): then using the starting node, the first key of the dictionary, we randomly walk the graph until a euler path is found: how we do this is for each node that we are currently at as the key in the dictionary, we find the list of nodes our current node has a path to, randomly choose a node from this list, and pop it from our list and set it as our new current node and recursively call into the visit function with this new node

f. This way, as we traverse the dictionary, we delete the corresponding edges of the graph so that we do not traverse them again and violate the rules of Euler's path.

g. My function finally outputs the list nodes it travelled

Results:

I found that my randomGenome() function does indeed return random nucleotides of A, T, G, C everytime and generates the correct DeBruijn graph from the graphviz package (which I commented out but if properly installed works as well). Since my code randomizes the path it chooses when there are two possible paths to traverse in a node, it does not always output the same genome. However, it loops until it can construct a full Euler path. The average rate for my function to output the correct genome depends on the number of unique eulerian paths that can be generated. For example, if there are two possible euler paths that are different sequences and only one of them is correct, on average I found that the correct sequence was output 50% of the time if there is a node that has 2 diverging paths in the euler path.

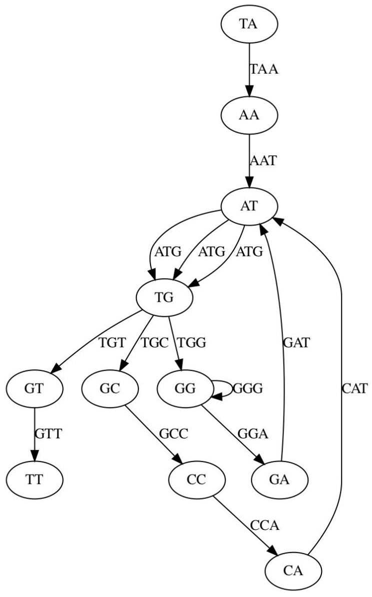
I measured the results of this project by first testing to see if I could find a eulerian path with the given parameters: X = 20, Y = 20, Z = 10 for a genome of 1,000 nucleotides. I was able to find a eulerian path after my functions looped through on average 800 times. When I varied X from 10 to 13 without repeats (Y = 0, Z = 0), I found that on average the number of random walks I needed went from 60 to 5. The larger the X value, the longer the K-mers and the less ambiguity in the correct eulerian path/ same genome to be found and the lower the average of times it takes more times on average to find a random walk.

I also varied the length of repeats Y and the number of repeats for Z. Repeats caused the graph to become more "tangled" and have much more possible paths to travel in the edge, lots of looping back. I found on average the number of random walks I had when I varied Y for (X = 20, Y from 15 to 20, Z = 10) my average went up from around 2 to 5. A demonstration of the effect of increasing Y on a small genome (**is shown in the diagram below**), we can see from g**raph A to graph B**, there are more possible paths that can be taken due to repeats. Similarly, when I varied Z, for (X = 20, Y = 20, Z from 10 to 15), I found that the average number of random walks I need to do went from from 5 to 20. This shows that the more repeats we have, the more times we have to run euler's path on the graph on average since there repeats cause ambiguity and more options for the traversal of nodes to stray from a euler path.

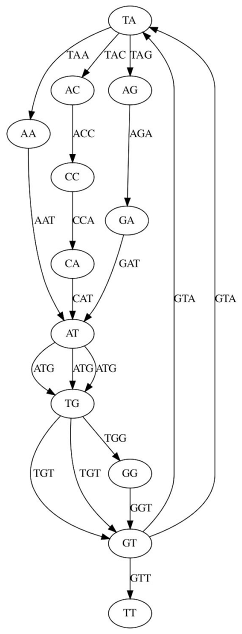
Also I noticed as I varied X, Y, Z and found when X < Y the average number of random walks increases by a lot since my repeats were longer than the reads and the genome mostly became repeats, causing my graph to have many loops.

Diagram A image (no repeats) for “TAATGCCATGGGATGTT”, X = 3, Y = 0, Z = 0
Diagram B image (with 2 repeats) for “TAATGCCATGGGATGTT”, X = 3, Y = 2, Z = 2 → “TAATGTACCATGGTAGATGTT”

Diagram A ➜ Diagram B



TAATGTTAACCATGGGATGTT



TATAATGCCATTAGGGATGTT

Issues:

I found that one in 50 times I run my code I get this error “RecursionError: maximum recursion depth exceeded in comparison” for when the recursion limit for checking is too high. It was fixed by just running the code again because it generates a different random genome or use “sys.setrecursionlimit(1500)” which set the recursion limit to a higher value since python is worried about stack overflow.

Also, since my god continues to loop until it finds a euler path, when I was testing I had to put a breakpoint to check how many times it took for it to loop to find that euler path. Keep in mind this is just outputting any correct euler path not the euler path that correctly matches the genome we are looking for.

Graphviz only worked on an IDE for me. I tried to install in a virtual environment in terminal and it would not work and commented it out my code. If you would like to run it I suggest pycharm.