Advanced Programming in Python

THERESSE JOY VILLARIEZ CALO (2158606)

July 2022

1 Overview

DNA sequencing is a method used to determine the order of nucleotide in a DNA. It is an important medical and biological tool that benefits numerous fields such as biotechnology, forensic science and virology.

A DNA sequence is a long chain of nucleic bases adenine, guanine, cytosine, and thymine. In order to construct one, a sequence undergo a process that involves segmentation. These smaller segments overlaps and aligns in order to produce the original sequence.

In this project, we were presented with several tasks that lead to finding a complete DNA sequence. In general, the codes that were used in the implementation of the codes lies heavily in the requirements specified in each task and limitations that comes with it. Some of the tasks were divided into smaller parts to allow the application of the specifications. In cases of the simpler tasks, a more straight forward techniques were applied.

2 Implementation Details

In the implementation of this project, the following factors were mainly considered: (1) Flow of the Code, (2) Creating Sub features, (3) Coding Style and (4) Standard Documentation. Additionally, for each sub-task, the following are enumerated: • approach chosen and possible alternatives, if any • reason chosen approach is the best one.

2.1 Flow of the Code

Instead of creating an entire class to produce a DNA sequence, the project is divided and arranged into separate functions relative to the subtasks assigned in the project. Before each function a comment of task description is included in order to ease possible improvements and changes. The inputs and outputs are clearly labeled so it is easy to recognize which kind of data is expected from and by the user.

2.2 Creating Sub features

In several occasions, some functions were broken down further into smaller function when the task is more intricate and specific applying a "Divide and Conquer" strategy. For example, in the event of cleaning the data with errors, assist functions were created to eliminate every possible error that can be expected from a raw data file. Meanwhile, in checking for the validity of a graph, assist functions were created to check for the adherence to a Euler's path conditions. Therefore, in an actual function itself, the procedures only involves summarizing the mini functions.

2.3 Coding style

Implementation of coding style is important in large projects such as this one. Otherwise, it can get very confusing and misleading when labels are topsy-turvy. One of the focus of this area is the proper naming conventions of the variables. Aside from following the the Python style guide in naming a variables, they are also formulated in a way that they are easily understandable and direct to the point. For example, the assist functions in cleaning the data functions are named after the kind of error they are expected to eliminate such as missing position and duplicate segments. Additionally, variables with more that one word are separated with and underscore and maintain in lowercase. Finally, iteration variables are kept simple and is usually assigned with a single letter variable.

2.4 Standard documentation

In both the project file and the project test file, description of sub-tasks were included as a guide during the initial coding process as well as revision and editing phases. The descriptions include the method to be processed and the type of input and output that will be dealt with, whether it is Boolean, string or graph or will it be returned or printed. Additionally, labels were also incorporated inside each function.

2.5 Subtasks Details

This section describes the functionalities applied to each subtask as well as the strategy use to achieve desired result.

2.5.1 Subtask 1: Read the csv file given its name.

The read_csv function takes a string input from the user which represents the csv file that contains different sequences of segments of the DNA. The file name takes the form of DNA_[x]_[k].csv where x is a number referring to which DNA this csv file belongs and k is the number needed to construct the de Bruijn graph with. The output of this function is a pandas data frame with column names 'segment', 'position', 'a', 'c', 'g' and 't'. This was implemented using the read _csv function of the pandas library.

2.5.2 Subtask 2: Clean the data of the data frame.

The function clean_data is divided into five (5) parts. The main part summarizes the four minor parts and at the same time implements them. The minor parts mediates the four possible errors that may occur in the data frame which are (1) missing position in the segment, (2) duplicated position in the segment, (3) wrong position and (4) duplicated segments. The clean_data function takes the data frame produced from subtask 1 as an input returns cleaned version of the data frame.

To mitigate the error missing position in the segment, an internal function _missing_position was created that drops an entire segment if the position is not complete. Through iteration, it records the positions of a specific segment as well as their assigned index. The maximum position is obtained from the position list. If the position list with unique position items matches the list of 1,..,maximum position, no action is takes and the data frame remain as is. However, if the two items do not match, the function drops the whole segment using the index recorded previously.

For the error duplicated position in a segment, the first step uses the existing drop_duplicates function that is set to delete a row if there is a duplicate in segment', 'position', 'a', 'c', 'g' and 't' but keep the first row. The second step is to delete any segment that has duplicated position but does not agree on the values of 'a', 'c', 'g' and 't'. This is done through iterating through each segment and recording the position and their assigned index in a list. If the position list matches the unique set of position list, then it means no more duplicates found and no action will be taken. On the other hand if the two don't match the whole segment will be dropped from the data frame because it cannot be identified which position is the correct one. This is done under _duplicated _position function

For the third error wrong position, an entire segment is dropped from the data frame if one position has more than one 1 value in 'a', 'c', 'g' and 't' because there is no way of knowing which is correct. This is done through the function _wrong _position and iterating each segment and recording their position and assigned index. Next step is to do a next level iteration in every position and totalling the values of 'a', 'c', 'g' and 't'. If the total is equal to one then it means that we have a correct position. However, if the total is not equal to 1m then the whole segment is dropped from the data frame using the previously recorded index.

Lastly, for the error duplicated segments, the function _duplicated _segments was created. The main idea of this function if to drop any duplicated segment if it has a similar 'position', 'a', 'c', 'g' and 't' values to other segment despite not having similar segment number. First, through iteration each 'position', 'a', 'c', 'g' and 't' of a segment are stored in a temporary list. A permanent list of unique segments is also created. If the current segment has no equal value in the unique segments list, the segment is added to the unique segments. Otherwise, the whole segment is dropped from the data frame using the index recorded for it.

2.5.3 Subtask 3: Generate JSON sequences from the dataframe.

The function generate _sequence takes in a previously cleaned data frame in subtask 2 as an input and produces a json object. For each segment on the list, a string composed of the letter 'a', 'c', 'g' and 't' is produced depending on each position they take. Each segment string is then consolidated into a list and the list gets dump into a json object to produce a json data.

2.5.4 Subtask 4: Construct de Bruijn graph.

The construct_graph function takes in a json data created in the previous subtask and a k value indicates the length of each k-mer taken from the input string of the user. To construct the graph, the first step is to list down all the k-mer of each json data entry. from for each k-mer, a left side and right side k-mer are extracted which are less than one element from the original. Each left and right k-mers is added to the graph as nodes and edges are also added from the left to the right. The function returns a a multidigraph.

2.5.5 Subtask 5: Plot the de Bruijn graph.

The function plot_graph takes in a graph created from the previous subtask as input and creates an image file on the local folder. The name of the image file is in the format of DNA_[x].png where x is the number referring to which DNA this graph belongs and is obtained from the string input of the user. The plot of the graph is constructed using the library matplotlib.pyplot.

2.5.6 Subtask 6: Check whether the de Bruijn graph can be sequenced.

To verify whether the de Bruijin graph can be sequenced, the function isvalid _takes a de Bruijin graph created in subtask 4 as an input an returns True is the graph adheres to the conditions of the Euler's path existence and False if not.

This function is divided into four parts. The main part returns the final Boolean value if and only if conditions 1 and 2 or a Euler's path are satisfied.

To check whether the condition 1 of Euler's path is satisfied, the function _condition _1 was created to verify if the given graph is connected. This function checks that for every node in the graph there is a path connecting it to all other nodes. If this is true for all nodes then it means that the graph is connected and the function returns a True value. Otherwise, if at least one node does not satisfy this condition then the graph is not connected and the function returns a False value.

There are three possible scenarios for condition 2. On the second condition stating that the number of nodes in the graph that have its InDegree different from its OutDegree is either 0 or 2, the function _condition _2 was created . (Scenario 1) In the case that no InDegree and Outdegree node is found in the graph then it means that the graph is Eulerian and the function returns a value of True. (Scenario 2) If there exist such nodes and the value is not 0 or 2 then the function returns False. (Scenario 3) If the difference is zero then the function returns True. (Scenario 4) However, if the difference is 2, the function calls out _condition _3 to verify further the validy of the graph.

For the third function, it verifies that if one of the two nodes has InDegree - OutDegree = 1, then the other should have OutDegree - InDegree = 1. Conversely if one of the two nodes has OutDegree - InDegree = 1, then the other has InDegree - OutDegree = 1. If this condition is satisfied then the function returns True otherwise, False.

2.5.7 Subtask 7: Construct DNA sequence.

To construct the DNA sequence, the function construct_dna_sequence takes in a de Bruijn graph as an input. First it validates if the DNA sequence of that graph can indeed be constructed using the function of subtask 6. If not, then an error message "DNA sequence can not be constructed." will be printed and returned. Otherwise, the complete string representation of the DNA sequence is constructed using the concept of Eurler's path.

2.5.8 Subtask 8: Save DNA sequence or write the error message.

The function save_output takes (1) the string representation of the DNA sequence produced in subtask 7 and (2) a string representation of an output file name as inputs. The output file name should follow the format DNA_[x].txt where x is the number referring to which DNA the sequence belongs to. It will contain either contain the DNA sequence of the graph if it satisfies the Euler's path condition or the error message "DNA sequence can not be processed." if it does not. The function simply creates a new txt file for writing with the given file name and writes the given string message.

3 Evaluation

This section documents if the task level has been completed on the following parameters (1) Sub-tasks, (2) Overl-all run, (3) Output in png and Output in txt., (4) Bonus Questions. Each parameter is rated as either *Correctly Implemented*, *Partially Implemented* or *Not implemented*.

3.1 Sub-tasks

Correctly Implemented

All eight (8) subtasks required in this project are completely implemented within the required conditions specified in the project information sheet. The correctness of each subtask can be validated through the test cases created along with the tasks. Although not all subtask has a parallel test case, they are interconnected thus their correctness affect the other.

3.2 Over-all run

Correctly Implemented

The program is able to take the input csv file with the file name format DNA_[x]_[k].csv and extract the relevant values of x which is a number referring to which DNA the csv file belongs and k which is the the number that you need to construct the de Bruijn graph with. When the program was executed two outputs was produced. (1) plot of the de Bruijn graph file name DNA_[x].png and a txt file with the name DNA_[x].txt contaning the DNA sequence of a de Bruijn graph which satisfies the conditions of having an Euler's path or the error message "DNA sequence can not be constructed.", otherwise. The program is runnable with the command python project.py DNA_[x]_[k].csv

3.3 Output

Correctly Implemented

In this project, two (2) program outputs are expected to be produced: (1) A plot of the de Bruijn graph. The name of the file is DNA_[x].png. (2) A txt file with the name DNA_[x].txt that contains the DNA sequence of the given data in case the de Bruijn graph satisfies the conditions of having an Euler's path or the message "DNA sequence can not be constructed." Variable x is a number referring to which DNA sequence the csv input file belongs. During testing process, it took approximately 1-2 minutes for the output to show up on the local folder.

3.4 Bonus Questions

Not implemented

4 Challenges

The primary challenge which I believe is also the most important in implementing such projects is really understanding the concept behind the assignment and getting a clear mental picture of what is it really about. I was able to over this challenge by (1) Reading the project information sheet over and over again. (2) Researching related literature and blogs to get more perspective about it. (3) Answering one subtask at time and correcting them again later once I have gathered more information and have better understanding about the task.

Another challenge that I encountered in this project which I believe is also inevitable for its length is how to stitch several subtask into a whole tapestry of a functioning DNA sequence generator. The key that I used was consistency constantly. I managed my time accordingly taking one step at a time and working on it constantly as not to lose momentum and fervor.

5 Tests Description

This section describes the functionalities of test cases employed in the project, how they are designed and how each test case is relevant to each subtask.

5.1 Subtask 2 Test: Clean the data of the data frame.

Subtask 2 was designed to clean the given data frame with the following errors (1) missing position in the segment (2) duplicated position in a segment, (3) wrong position and (4) duplicated segments. For testing, test cases for all the four conditions and all their possible scenarios were created to ensure that the function deliver the required.

5.2 Subtask 3 Test: Generate JSON sequences from the dataframe.

The test is designed to check how successfully the function converts a cleaned data frame from subtask 2 into a json object. The output of the function should be equal to an expected json object.

5.3 Subtask 4 Test: Construct de Bruijn graph.

This test aims to verify that the function is able to construct a correct de Bruijn graph given a json object an a k-mer number. As a requirement, the function should be able to breakdown the items of a json object into several k-mers and each k-mer to further left and right k-mers. The output of the function should be equal to the expected according to the given de Bruijn graph definition.

5.4 Subtask 6 Test: Check whether the de Bruijn graph can be sequenced.

Given the conditions of the Euler's path the function should be able to return True or False if the graph adheres to them. In the test scenarios for a valid and invalid graph were included to test whether the function returns the appropriate Boolean value.

5.5 Subtask 7 Test: Construct DNA sequence.

A test case is made for given a valid de Bruijn graph, the function returns a string representation of a DNA sequence. Another test is made for given a invalid graph, the function is able to return the error message "DNA sequence can not be constructed."