Zihang Zhou
10/02/2019

Project Summary

My overall approach was to extract DNA data into a list of lists, where I can sort it so it would reduce run time for later searching. Then extracted target DNA position data into another list of lists, where I can add coverage later on. Then I started searching coverage and added them into list of target DNA positions. After searching was completed, I exported list of target DNA positions filled with coverage. Unit tests were implemented to ensure accuracy and integrity of the program.

Extracting DNA data:

When extracting DNA data, I decided to use a dictionary to store data first, then convert them into a list. This has two reasons. First, accessing items in a dictionary takes O(1) time while accessing elements in a list takes O(n) time. Second, many data were repeated, in order to reduce store size, combine them were important. This would require a search in the data storage structure, which consumes time. So dictionary would be ideal. This way, I can quickly import data into program. However, dictionary is unordered and takes up a lot of memory, this would cause a problem for later searching because it would be difficult to apply any algorithm to improve efficiency and burns out memory. Thus convert it to a list would be a wise choice. Also, I decided to store start and end position instead of start and length of DNA sequence because it would be easier for later comparison in searching function.

Extracting target DNA position data:

I used list here because the sample size is acceptable so list would be ideal.

Searching algorithm:

I used brute force algorithm to find coverage. Initially, I tried to use binary search to find coverage since I had a sorted list. The problem with it is target DNA position is in a range between start and end positions of a given DNA sequence, and the graph of start or end position is zigzag-like given any one of the other sorted. While performing brute force search, two areas were optimized to improve efficiency. First, the data size has been truncated significantly. Original DNA data has 1980584 records. After extracting data while pre-processing, the new DNA dataset has 143915 records. This is approximately 1/13 of the original dataset. Second, when searching for each target position, once the start position of a given DNA sequence is larger than the target position, the algorithm would stop this searching and start searching for next target position. This would cut off huge unnecessary searching that would otherwise perform when target position is significantly smaller than the majority of records. Another potential area to improve is to find out a way to cut off unnecessary searching for target positions where majority of records' end positions were significantly smaller than target position. The issue here is since the graph of dataset is zigzag-like, it would be difficult to find the right position to cut off without cutting records that may cover target DNA positions.

Exporting data:

After coverage were found, export them to the original target DNA positions file.

Unit testing:

This project had multiple methods in the program. Testing each individual method is important to help debugging when error occurs and ensure coverage of potential edge cases and detect unnoticed errors.

Run-time Analysis:

If use brutal force without optimization, the run-time would be 1980584 * 1000. After shrinking the database, the run-time became 143915 * 1000. After applying cutting on the start position, run-time became < 143915 * 1000. Additional cutting may be added on the end position if a proper way of finding the lowest matching DNA sequence can be performed effectively.