Team Contract:

Team Members:

1. Aryan Arora, aarora14
2. Aditya Prerepa, prerepa2,
3. Shashwat Mundra,, mundra3
4. Nikhil Dhomse, dhomse2

Communication:

To collaborate and discuss the progress of the project the team will meet twice each week, either in person at a pre-decided location or online via Zoom. The minutes of these meetings will be recorded by Aryan on a shared google doc.

For all other instant questions and doubts that team members may have for each other would be discussed in a group already created on Discord. The platform “Discord” was mutually regarded as the most convenient platform by all team members.

To respect each team member’s ideas and opinions, each team member agrees to listen to the ideas other team members might have and not disregard it.

Furthermore, each team member agrees to be transparent with the team and to intimate in advance in the following but not limited to these cases: unable to attend weekly meetings, unable to complete assigned work or other commitments that might interfere.

Collaboration:

To promote an equitable work environment, each team member agrees to take on particular aspects of the final project. All team members are interested in working on the different deliverables for the project and would be contributing to each deliverable, however each team member’s preferences are highlighted below :

- Aryan Arora:
- Shashwat Mundra:
- Aditya Prerepa:
- Nikhil Dhomse:

To manage time commitments, all team members will try finding mutually convenient times to meet and the team members will have general flexibility in working around their schedule. If there are certain unforeseen circumstances that arise, all team members agree to be flexible and help find a solution or navigate the situation.

In case of conflicts, the team members worry that an intellectual disagreement has a high likelihood of arising. In the case it does, team members agree to discuss the different philosophies or concepts and decide as a group, the best way to move through, however, the team would also ask the mentor for advice to make sure that the team is still on track.

In case of other conflicts such as team members slacking, or incompletion of work, the issue shall be brought up during the weekly meeting and in case the team is unable to find a possible solution, the team would seek help from course staff.

Final Project Proposal:

We will be referring to the following paper to reproduce some of their methods:
Integration of string and de Bruijn graphs for genome assembly
(https://academic.oup.com/bioinformatics/article/32/9/1301/1744507)

Leading Question

Can the sequence of the genome for Staphylococcus aureus be successfully reproduced using De Bruijn Graphs with improvements made using assembly approaches with FM index to build an assembly graph from the corrected data?

Major Deliverables:

Our Major Deliverables include the construction of a **De Bruijn Graph** to represent the overlaps between sequences of DNA/RNA and to find the 'contigs'. Then **finding the regions of high error using FM Index** with **assembly approaches** and building an **assembly graph** from the corrected data which will help determine the genome.

To condense it:

1. **Construction of the De Bruijn Graph**
    a. The De Bruijn Graph is a directed graph with nodes representing k-mers and edges representing the overlap between adjacent k-mers. Each edge is labeled with the corresponding (k-1)-mer that overlaps between the two k-mers.
    b. The graph is constructed from the dataset and each node represents the nodes represent k-mers and edges connect to each k-mer
2. **FM Index Approach**
    a. This step follows the construction of the De Bruijn Graph. A De Bruijn graph represents the overlapping relationships between k-mers in the sequencing data, but it does not provide an efficient way to search for specific sequences or patterns within the data.
    b. We will use a Suffix Array to store the data here as it can be accessed in O(1) time.
    c. The FM Index is aimed at finding feasible {A, T, C, G} - extensions from the first end to the other end by updating the suffix array (SA) intervals. The source and destination of each search are maintained using a search tree which includes storing the source index and the destination index.
3. **Stride Assembler Techniques:**
    a. This includes processing the produced Suffix Array, in terms of read decomposition and overlap computation with SA pruning
    b. Read Decomposition involves going through the sub-reads and removing dead ends that are not required.

c. Overlap Computation with SA Pruning involves reducing the number of false edges formed due to the decomposition of subreads in b.

4. **Graph Layout:**
    a. This is another major part of the project where the work done involves using few techniques to output the unique paths of the graph that display high accuracy and resemblance.
    b. The first part is Simple Path Statistic: It measures the length distribution of simple paths in the assembly graph without repeating edges.
    c. This involves the removal of edges with low overlap ratios.

5. Overlap Layout Consensus/Construction of the Simple-path statistic
6. *(In case the above one doesn't work) Making an algorithm to match the output with the genome sequence, measure of accuracy)*

Dataset Acquisition and Processing:

Sequence Read Archive from the NIH (https://trace.ncbi.nlm.nih.gov/Traces/?view=run_browser&acc=SRR23718782&display=metadata)

Most of the data used will be from the FASTA/FASTQ files from the NIH Database, more specifically referring to the S. Aureus Genomic Dataset for the SEMAPHORE project.

The data includes the DNA information for the S. Aureus bacteria formatted using FASTA format, more information here FASTA File Format Documentation : (https://zhanggroup.org/FASTA/)

The data has mostly been pre-processed and cleaned since it has been used for a study already. We will however, be extracting information from each line and taking into account the length and the DNA string, but ignoring the other information.

We will be using an adjacency list to store the de bruijn graph each node of the graph represents a k-mer and each edge represents the corresponding node

Algorithms

For academic reference, we will be referring to the following paper titled "Integration of string and de Bruijn graphs for genome assembly" https://academic.oup.com/bioinformatics/article/32/9/1301/1744507) mentioned earlier as well.

This includes both string and de Bruijn graphs, and most of the project is modeled on this research paper.

We take the data obtained from the NIH and convert that to a De Brujin Graph and then process it using FM Index which refers to Full-text index in Minute space which can efficiently find the number of occurrences of a pattern within the compressed text, as well as locate the position of each occurrence. The de bruijn graph is a directed graph representing overlaps between sequences of symbols, in our case it refers to the repetead DNA sequences.

The project involves four stages, first being covered in the paragraph above which involves the construction of the De Bruijn Graph. The nodes and edges for the same have been defined in the major deliverables. This graph is then used to work on a FM Index to find paths from the node to the end. This outputs a suffix array and a suffix tree.

While this data seems conclusive, we need to take into account a few more things used by the StriDe Assembler. This involves following the read decomposition and the overlap computation done by setting breakpoints to test segments.

Followed by this we have the Graph Layout Algorithms which are required to identify unique paths.

Let's take a look at what each algorithm used here does:

1. Construction of the De Bruijn Graph:
    a. Choose a value of k, the length of the k-mers that will be used to construct the graph.
    b. Construct a set of k-mers by breaking each DNA sequence into overlapping substrings of length k.
    c. Create a node in the graph for each unique k-mer in the set.
    d. For each k-mer, create a directed edge from the node representing the prefix k-1-mer to the node representing the suffix k-1-mer of the k-mer.
    e. If multiple DNA sequences are available, connect the nodes corresponding to the overlapping k-mers from adjacent sequences with additional directed edges.
2. FM Indexing algorithm
    a. Construct the Burrows-Wheeler Transform (BWT) of the de Bruijn graph.
    b. Construct the inverse BWT, which is the original string from which the BWT was constructed.
    c. Precompute the rank array, which is an array that stores the number of occurrences of each character in the BWT up to a given position.
    d. To search for a specific sequence or substring in the de Bruijn graph, first construct its BWT, and then use the BWT and rank array to perform a binary search in the suffix array. The resulting range of indices in the suffix array corresponds to the positions of all occurrences of the query sequence in the original string.
    e. Do the FM-index for each k-mer. The suffix array is a sorted list of all the suffixes of the string.

        i. The FM-index can be used to quickly find the suffix array of a substring of the string. To construct the FM-index for each k-mer, we use the FM-index construction algorithm.
    f. Store the k-mers that are associated with each edge in the FM-index of the k-mer that is the suffix of the edge. For each edge (u, v) in the de Bruijn graph, store the k-mers that are associated with the edge in the FM-index of the k-mer that is the suffix of v. This can be done by traversing the edge from u to v and adding each k-mer that is associated with the edge to the FM-index of the k-mer that is the suffix of v.
3. Stride Assembler Techniques:
    a. First we do Read Decomposition which involves the following steps
        i. Identifying regions in the sequencing reads that are likely to contain errors or polymorphisms, such as regions with high coverage, low quality scores, or frequent mismatches with a reference genome.
        ii. Divide the reads into k-mers of fixed length, with a size that depends on the expected error rate and the size of the error-prone regions.
        iii. Cluster the k-mers based on their sequence similarity and create consensus sequences for each cluster using a voting or alignment-based approach.
        iv. Reconstruct the original reads from the consensus sequences and the remaining k-mers, using a greedy approach, to obtain a more accurate representation of the original sequences in the error-prone regions.
    b. This is then followed by overlap computation with SA-interval pruning.
        i. Construct a suffix array (SA) and its corresponding LCP (longest common prefix) array for the set of reads.
        ii. Compute the SA interval for each read, which is the range of suffixes in the SA that correspond to the read.
        iii. Prune the SA intervals for each read by considering its LCP with adjacent reads.
        iv. Compute the overlap length between each pair of reads whose pruned SA intervals overlap, and use the resulting overlap graph or table to assemble the reads into longer contiguous sequences.
4. Graph Layout Algorithm
    a. This involves computing simple path statistics for the graph - in which the statistic for a node is defined as the sum of the products of the weights of all paths from the source node to that node, where the weight of a path is the product of the weights of all edges in the path.
    b. This is then followed by the removal of edges with low overlap ratios
    c. Followed by removal of Removal of edges with large overlap differences, which can be measured at the time.

For our output we will produce genomic data, which will be compared to the genome of the bacteria S. Aureus. which is publicly available. Our project aims to try to find a somewhat similar match to that.

Theoretically, FM index takes O(m) to be constructed where m refers to the pattern being searched, however, we will be constructing the graphs as well, which will take up a much higher time complexity.

Our testing strategy will be to test our program in parts, this includes testing the de bruijn graph construction first followed by testing of the FM index algorithm, and the assembly algorithm used to make sure all overlaps are properly detected and indexed, followed by testing of the alignment before finally testing the output genome produced by comparing it to the publicly available genome.

Timeline

Our Timeline for this project includes us doing weekly meetings with some time alloted each week to work on the project.

- Week 1: Data Processing and start work on Graph construction
- Week 2: Analyse the De brujin graph to find overlaps, start work on analysis
- Week 3: Finding the regions of high error using the FM index
- Week 4: Build the assembly graph and output the genome.
- Week 5: Testing overall process, making sure all functions are complete.