IITM-CS6024 : Algorithmic Approaches to Computational Biology — Release Date: Sep 19

Assignment 1 — Due Date: Sep 29, 23:59

Roll No: BS17B011 — Name: G PRASHANT

- Use LaTeX to write-up your solutions, and submit the resulting single pdf file at GradeScope (Note: Within GradeScope, indicate the page number where your solution to each question starts, else we won't be able to grade it!).
- Collaboration is encouraged, but all write-ups must be done individually and independently, and mention your collaborator(s) if any. Same rules apply for codes uploaded to HackerRank (i.e., write your own code; we will run plagiarism checks on codes).
- If you have referred a book or any other online material for obtaining a solution, please cite the source. Again don't copy the source *as is* - you may use the source to understand the solution, but write-up the solution in your own words.

---

1. (6 points) [Choose your DS!]:

   We saw the algorithm below to construct an Eulerian cycle in a graph (with n nodes and m edges). What data structures would you choose to implement Lines 4,5,6 in the algorithm below, so that the overall algorithm has $O(m)$ running time complexity? Please assume that Graph is a balanced and strongly connected directed graph provided as an adjacency list.

```
1 EULERIAN_CYCLE(Graph)
2  form a cycle Cycle by randomly walking in Graph(don't revisit same edge
3   while there are unexplored edges in Graph
4     select a node newStart in Cycle with still unexplored edges
5     form a cycle Cycle' (starting at newStart) & then randomly walking
6     Cycle <- merge(Cycle, Cycle')
7   return Cycle
```

SOLUTION

One of the appropriate and efficient data structures that can be used to store `Cycle` in this algorithm is **Doubly Linked List (DLL)**. The Graph can be stored in an adjacency list, as mentioned in the question. Details on how the lines $4, 5$ and 6 can be implemented, are mentioned below:

Line 4: At the initial iteration, we traverse through the DLL to find a node that has at least one outgoing edge and assign that to `newStart`. In subsequent iterations, we make use of the stack `outGoing` obtained from Line 5 to choose `newStart`. This line runs in constant time ($\mathcal{O}(1)$).

Line 5: This step involves generating a new cycle `Cycle'` starting from `newStart`. Given a node, determining one of its outgoing edge happens in constant time because of the Graph is stored in an adjacency list. We have to keep removing the edges from the graph as we traverse through them, and deleting an edge from an adjacency list also happens in constant time. Additionally, inserting a node in a DLL at the end happens in constant time. At the end of this line, we also generate a linear stack `outGoing` that stores the nodes having at least one outgoing edge in the updated Graph. Hence, the time complexity scales with the number of edges in this smaller cycle m_i, making it $\mathcal{O}(m_i)$

Line 6: This step involves merging `Cycle` and `Cycle'`. Both `Cycle` and `Cycle'` are DLLs, and we already know at which position we have to merge (`newStart` in `Cycle`). We have to associate `newStart` in `Cycle` with the beginning of `Cycle'` (after deleting the first node of `Cycle'`). Further, we have to associate the last node of `Cycle'` with the node that was previously the next node of `newStart` in `Cycle`. All these individual operations happen in constant time, making the time complexity of this step $\mathcal{O}(1)$.

Hence, the overall time complexity of this algorithm becomes

$$\sum_{i=1}^{J} \mathcal{O}(m_i) = \mathcal{O}(m)$$

where J is the number of smaller cycles traversed. Therefore, the time complexity of this algorithm scales linearly with the number of edges.

Other approaches to solving this problem are Hierholzer's Algorithm [2] and Fleury's Algorithm [1].

2. (6 points) [Counting ambiguity]:

   (a) (3 points) Find a simple DNA sequence whose k-mer composition agrees with that of exactly 5 other DNA sequences. Justify. (Hint: What would its de Bruijn graph look like? Optional: Do "articulation points" help identify contigs in this graph, or do you need "maximal non-branching paths" as mentioned in the book?)

SOLUTION

Let us consider a simple case where $k = 2$ and where the DNA sequence of interest consists of only two nucleotides (say A and T). For the 2-mer composition of the sequence to agree with exactly 5 other DNA sequences, the de Bruijn graph, with nodes "A" and "T", should have one node (say A) with 4 self loops. In addition to this, there should be one outgoing edge from each node to the other. By traversing through all possible Eulerian circuits,

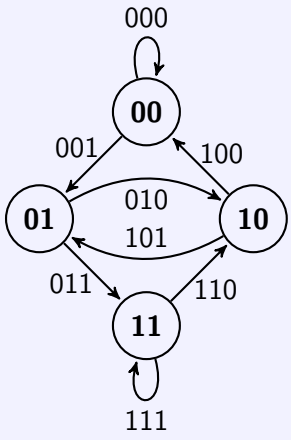
we can identify six distinct DNA sequences with the same k-mer composition.

One possible DNA sequence of length 7 is **S** =**AAAAATA**. The 2-mer composition of **S** is {AA,AA,AA,AA,AT,TA}. Five other DNA sequences with the same 2-mer composition are **AAAATAA**, **AAATAAA**, **AATAAAA**, **ATAAAAA**, **TAAAAAT**.

Optional Question: The contigs of this de Bruijn graph can be identified by the maximal non-branching paths. In this example, the maximal non-brancing paths that represent contigs are **ATA, AA, AA, AA, AA**.

(b) (3 points) How many 3-universal circular binary strings are there? Justify. (Optional: How would you count the number of k-universal circular strings?)

SOLUTION



The balanced and strongly connected graph on the left represents a De Bruijn graph of 3-universal strings.

Starting from node 00, we can follow the cycle:
000 → 001 → 011 → 111 → 110 → 101 → 010 → 100 → 000
This cycle gives the universal binary string **00011101**.

Alternatively, starting from 111,
111 → 110 → 100 → 000 → 001 → 010 → 101 → 011 → 111
This cycle gives the universal binary string **11100010**.

We can notice that they are complementary to each other. If we consider all the rotational variants of these two 3-universal circular binary strings, there are **16 3-universal circular binary strings** (8 rotational variants of each of the two) in total.

Optional Question: We know that there exists k-universal binary circular strings, as the corresponding De Bruin graph will be balanced and strongly connected, with each node having in-degree = out-degree = 2, indicating the possibility of multiple Eulerian circuits. The number of k-universal circular binary strings is given by the formula

$$N(k) = 2^{2^{k-1}}$$

The derivation of this formula is not straightforward and cannot be easily derived using combinatorics.

3. (11 points) [Coding warmup]:

   (a) (8 points) Implement the clump finding algorithm in the HackerRank contest at `https://www.hackerrank.com/aacb-assignment-1`. (Note: Create your `https://www.hackerrank.com` account using your smail id, and ROLL NUMBER as your user-name. Languages supported are C, C++, Java, Python, and R (use only one)).

   (b) (3 points) In addition, write here in this document the key data structure you used to implement the code, and associated time and space complexity of your implementation? Does you algorithm work for any value of L, t and k?

SOLUTION

The key data structure used to implement the Python code is **dictionary** (`freq_dict`), which uses a **hash table** to store Key-Value pairs. In this scenario, the Keys represent the immutable k-mer strings and the Values represent the number of occurrences of that k-mer in a particular region of interest (of length L) of the Genome (of length N).

We iterate over all the $N - L + 1$ regions of length L in the Genome. In the first iteration, we scan through the region containing the first L nucleotides of the Genome, to identify the k-mers and update the counts/occurrences in the dictionary. We make use of the `freq_dict.get(kmer,0)` method to handle cases when the k-mer is not already a key in the dictionary. If the number of occurrences of a k-mer is at least t, we store it in a list named `freq_patterns`. For subsequent iterations across the Genome, rather than recomputing the frequency dictionary, we only reduce the frequency of the first k-mer in the previous region by 1 and increase the frequency of the last k-mer in the current region by 1. If the last k-mer's frequency is at least t, we append it to `freq_patterns` if it is not already included. Finally, we sort `freq_patterns` in ascending order, whose time complexity is ignored as in practical cases, the number of clumps is far lesser than the length of the Genome.

The worst case time complexity of this algorithm is $\mathcal{O}(\mathbf{N})$. This algorithm took $\approx$ 12.25 ms to find the clumps of the Sample Input given in HackerRank, with Genome size 8969, $L = 511$, $k = 10$ and $t = 20$. The worst case space complexity is $\mathcal{O}(\mathbf{4^k})$, when the dictionary has all 4^k k-mers as its keys at any point in the algorithm.

As the time complexity of the algorithm is linear in the size of the Genome, it scales very well and can efficiently work even for very large values of Genome size. The algorithm will work for any value of L, t and k, provided the appropriate constraints are satisfied, i.e., $N \geq L$, $L \geq k$, and $L - k + 1 \geq t$.

4. (11 points) [Assembling SARS-CoV-2/COVID-19 genome] A tiny virius SARS-CoV-2 (~$100nm$ diameter per virion) has been causing such havoc in the way our world operates and communicates, and you are naturally curious to know what genome sequence it holds. Thankfully, we've algorithms to assemble the viral genome from a set of whole genome sequence (WGS) reads obtained using an Illumina sequencer (see attached **SRR12638317.sra_[12].fastq** files). You can use Velvet, which is one of a number of *de novo* assemblers that uses short read sets as input and constructs de Bruijn graphs for genome assembly, inside the Galaxy framework.

   You can use any of these Galaxy Servers that host Velvet Assembler.

   - `https://usegalaxy.org.au/`
   - `http://bf2i-galaxy.insa-lyon.fr:8080/`
   - `https://usegalaxy.eu/`

   Helpful Link: `https://galaxyproject.github.io/training-material/topics/assembly/tutorials/general-introduction/tutorial.html` for genome assembly workflow of a different data.

   (a) (2 points) What is the average number of wrong base calls in the last 5 positions of the 1^{st} read in SRR12638317.sra_2.fastq file? (Hint: Use quality scores provided by the Illumina 1.9 Next Generation Sequencer (NGS).)

SOLUTION

The equations used to calculate the probability (p) of a base being incorrect are as follows:

$$(Q_{Phred}) = \text{ASCII Value(Base Quality)} - 33$$

$$p = 10^{-\frac{(Q_{Phred})}{10}}$$

The last 5 positions of the 1^{st} read in SRR12638317.sra_2.fastq, along with the base quality and Phred Score are tabulated below:

Base	Base Quality	ASCII Value	Phred Score (Q_{Phred})	p
A	H	72	39	1.2589e-4
C	H	72	39	1.2589e-4
C	H	72	39	1.2589e-4
C	H	72	39	1.2589e-4
A	H	72	39	1.2589e-4

As the base qualities of all the bases in the last 5 positions are same (H), the probability of each of them being incorrect is also same ($p = 1.2589e-4$). Hence, the average number of incorrect bases in the last 5 positions is:

$$I_{n-5:n-1} = \sum_{i=n-5}^{n-1} p_i = 5 \times 1.2589 \times 10^{-4} = \mathbf{6.2945 \times 10^{-4}}$$

where n is the read length of the first read, and i is indexed from 0.

(b) (2 points) The first step in any sequence analysis is to run FastQC tool. Using its output, report the length of each read, and calculate the average coverage of the genome by all reads (given a guess that the length of the coronavirus is 31.7 kilo bases; note that average genome coverage is the average number of reads that span a nucleotide in the genome)?

SOLUTION

The two FASTQ files represent paired-end sequence reads. Each file contains 483691 reads. Hence the total number of sequence reads is $2 \times 483691 = 967382$. The read length of each sequence ranges from 35 to 151. The average read lengths of the two FASTQ files are 116.414 and 116.568, respectively. Given that the length of the SARS-Cov-2 genome is 31.7 kilo bases, the average sequence coverage is given by

$$\begin{aligned} \frac{\text{Tot. No. of Sequenced bases}}{\text{Genome length}} &= \frac{\sum_{i=1}^{2}(\text{no. of reads})_i \times (\text{Average Read Length})_i}{\text{Genome length}} \\ &= \frac{483691 \times (116.414 + 116.568)}{31700} \\ &\approx \textbf{3500X Coverage} \end{aligned} \tag{1}$$

(c) (3 points) Run velvet with a k-mer size of 29, and report how many contigs have been built, and what the mean, min and max length of the contigs are?

SOLUTION

A total of 2,657 number of contigs were built, with a k value of 29.
Mean length of contigs = 43.1376
Minimum length of contigs = 29
Maximum length of contigs = 353

(d) (4 points) Rerun velvet with the following k-mer sizes: 23, 57, 100, and report for each case the above contig metrics in table format. Comment on the trade-off between small and large k-mer values, and which k value looks optimal to you.

SOLUTION

k	No. of Contigs	Mean length	Minimum Length	Maximum Length
23	3,855	34.99	23	320
29	2,657	43.1376	29	353
57	412	78.885	57	411
100	66	185.074	101	798

We can observe that as the value of k increases, the total number of Contigs

decreases and the mean length of the Contigs increases. A higher value of k is more advantageous as it builds lesser number of contigs of longer length, making *de novo* Genome Assembly relatively easier. Moreover, a higher value of k also results in a less tangled de Bruin graph. However, it also results in lesser coverage of the genome, which would cause the de Bruijn graph to have missing edges. Hence, the optimal value of k should be chosen keeping this trade-off in mind. In this example, $k = 57$ appears to be optimal.

5. (6 points) [Research warmup]: Provide properly-formatted references for papers in this solution.

   (a) (3 points) Browse through the latest issue of top-ranked bioinformatics or systems biology journals (e.g., Bioinformatics, PLoS Computational Biology, Cell Systems, Molecular Systems Biology, etc.). Which one article caught your attention/interest the most based on only the titles and/or abstracts, and why?

SOLUTION

The research article that caught my attention the most is **Promoter analysis and prediction in the human genome using sequence-based deep learning models**, which was published in the Bioinformatics Journal in 2019 [4]. The reasons that motivated me to read this paper are as follows:

- Sequencing the human genome is a less laborious task and there is an explosion in the amount of sequence data generated nowadays.
- However, not all the regions of the genome have been identified and characterized due to inherent computational challenges.
- Promoter regions play an important role in the initiation and regulation of gene transcription. Identifying these regions can provide insights into potential treatments for a variety of diseases.
- I was highly intrigued by the approach of using cutting-edge Deep Learning techniques like Convolutional Neural Networks (CNNs) to solve this complex biological problem.

   (b) (3 points) What is the latest research publication you could find on *de novo* genome assembly (i.e., genome assembly as seen in class without the knowledge of a reference genome), and what key algorithm did it use?

SOLUTION

One of the latest research publication on *de novo* Genome Assembly is **GRASShopPER - An algorithm for de novo assembly based on GPU alignments**[3], which was published in PLOS ONE Journal in 2018. This

greedy hyper-heuristic algorithm uses an **Overlap Layout Consensus (OLC)** approach and leverages advanced hardware technologies like **GPUs** for parallelizability, thereby greatly reducing time and memory-consumption.

References

[1] M Fleury. Deux problemes de geometrie de situation. *Journal de mathematiques elementaires*, 2(2):257–261, 1883.

[2] Carl Hierholzer and Chr Wiener. Über die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[3] Aleksandra Swiercz, Wojciech Frohmberg, Michal Kierzynka, Pawel Wojciechowski, Piotr Zurkowski, Jan Badura, Artur Laskowski, Marta Kasprzak, and Jacek Blazewicz. Grasshopper—an algorithm for de novo assembly based on gpu alignments. *PloS one*, 13(8):e0202355, 2018.

[4] Ramzan Umarov, Hiroyuki Kuwahara, Yu Li, Xin Gao, and Victor Solovyev. Promoter analysis and prediction in the human genome using sequence-based deep learning models. *Bioinformatics*, 35(16):2730–2737, 2019.