Bio-informatics Assignment Submission 02

Student:
Tajkia Nuri Ananna
Reg. No: 2016331092
Email: tazkiaaltaaf@gmail.com

Instructor:
Moqsadur Rahman
Assistant Professor
Email: moqsadsust@gmail.com

Date of Submission: December 20, 2020

Department of Computer Science and Engineering

Contents

1 Chapter 3

1.1 Questions for One Marks

1.1.1 Q1: What is the length of human body genome?

Roughly 3 billion.

1.1.2 Q2: What is a Hamiltonian path?

A path in a graph visiting every node once is called a Hamiltonian path.

1.1.3 Q3: How do we know if any binary string is k-universal?

If the string contains every binary k-mer exactly once.

1.1.4 Q4: What is the main difference between Hamilton path and Eulerian path?

Hamilton path is a path visiting every node in graph exactly once. Eularian path is a path visiting every edge exactly once.

1.1.5 Q5: Write down the Euler's theorem.

Every balanced, strongly connected directed graph is Eulerian.

1.2 Questions for Five Marks

1.2.1 Q2: Draw overlap graph from the following reads. AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

The overlap graph drawn from the reads are shown in Fig. 1.
The rule followed while drawing the graph is : connect two 3-mers with an arrow every time the suffix of one is equal to the prefix of the other.

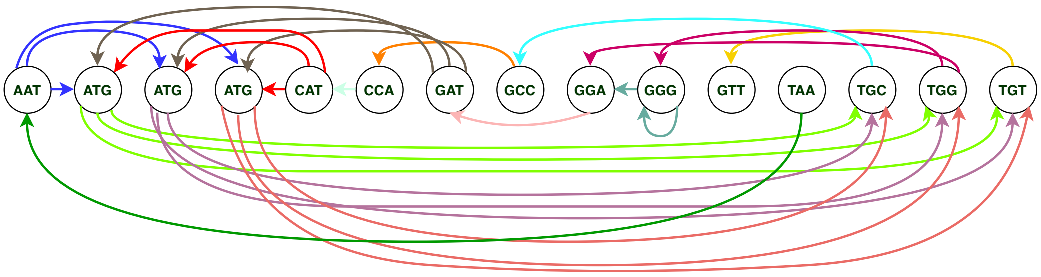


Figure 1: Overlap graph

1.2.2 Q3: Find out Hamilton path from the of the overlap graph drawn in Fig. 1.

A path in a graph visiting every node once is called a Hamiltonian path. We have highlighted the path of one Hamiltonian path found from the overlap graph. We have shown the output in Fig 2. The path we got is TAATGGGATGCCATGTT.

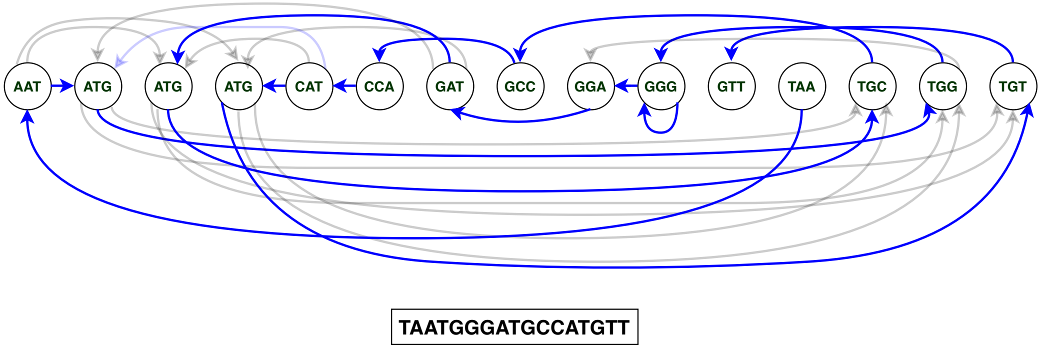


Figure 2: Hamilton graph

1.2.3 Q3: Write down Euler's theorem. Take a graph by yourself and prove that it satisfies Euler's theorem.

Euler's Theorem: Every balanced, strongly connected directed graph is Eulerian.
Proof: We have taken a figure which is shown in Fig. 3. From the Euler's theorem we know that for a graph to be eularian, it has to satisfy two rules:

1. **Balanced:** A balanced graph means for every node the outgoing and incoming edge should be equal.

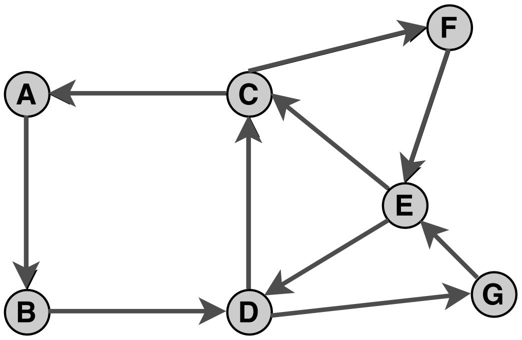


Figure 3: Eular Graph

2. **Strongly connected:** From every node, there should be a way to visit every other node.

If we can prove that Fig. 3 satisfies these two rules, we can say that it is an Eularian graph.

1. **Balanced:**

   (a) Node A: Incoming edge (A) = 1 = Outgoing edge (A)

   (b) Node B: Incoming edge (B) = 1 = Outgoing edge (B)

   (c) Node C: Incoming edge (C) = 2 = Outgoing edge (C)

   (d) Node D: Incoming edge (D) = 2 = Outgoing edge (D)

   (e) Node E: Incoming edge (E) = 2 = Outgoing edge (E)

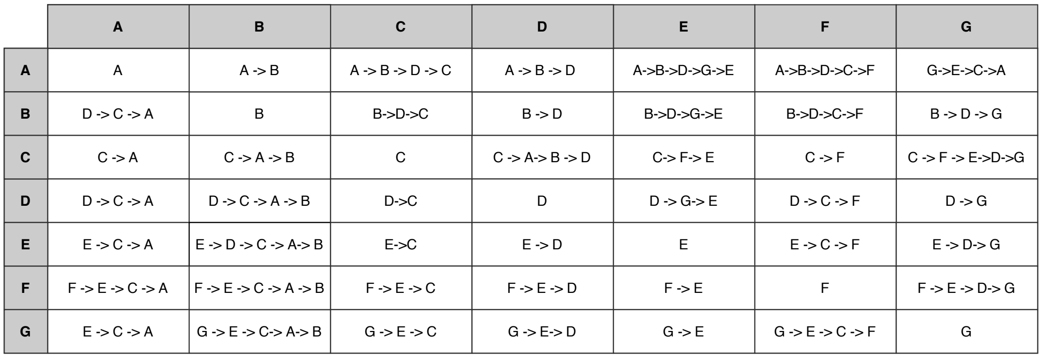
   (f) Node F: Incoming edge (F) = 1 = Outgoing edge (F)

   (g) Node G: Incoming edge (G) = 1 = Outgoing edge (G)

   Conclusion: This is a balanced graph.

2. **Strongly connected:** In the Fig. 4, we have enlisted all the way from each node. We can observe that, it is possible to visit every other node from each of the nodes.

Thus we can say, the graph shown in Fig. 3 is an Eularian graph.



	A	B	C	D	E	F	G
A	A	A -> B	A -> B -> D -> C	A -> B -> D	A->B->D->G->E	A->B->D->C->F	G->E->C->A
B	D -> C -> A	B	B->D->C	B -> D	B->D->G->E	B->D->C->F	B -> D -> G
C	C -> A	C -> A -> B	C	C -> A-> B -> D	C-> F-> E	C -> F	C -> F -> E->D->G
D	D -> C -> A	D -> C -> A -> B	D->C	D	D -> G-> E	D -> C -> F	D -> G
E	E -> C -> A	E -> D -> C -> A-> B	E->C	E -> D	E	E -> C -> F	E -> D-> G
F	F -> E -> C -> A	F -> E -> C -> A -> B	F -> E -> C	F -> E -> D	F -> E	F	F -> E -> D-> G
G	E -> C -> A	G -> E -> C-> A-> B	G -> E -> C	G -> E-> D	G -> E	G -> E -> C -> F	G

Figure 4: Path from every node

1.3 Questions for Ten Marks

1.3.1 Q: Construct DEBRUIJN$_3$(TAATGCCATGGGATGTT) showing COMPOSITIONGRAPH$_3$(TAATGCCATGGGATGTT) and PATHGRAPH$_3$(TAATGCC

1. **Composition Graph:**

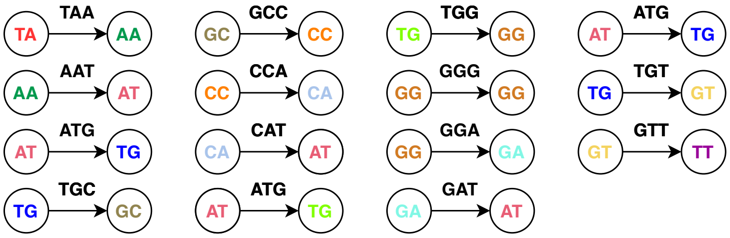


Figure 5: COMPOSITIONGRAPH$_3$(TAATGCCATGGGATGTT)

2. **Path graph:**

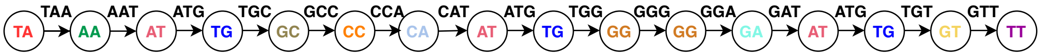


Figure 6: PATHGRAPH$_3$(TAATGCCATGGGATGTT)

3. **de Bruijn Graph:**

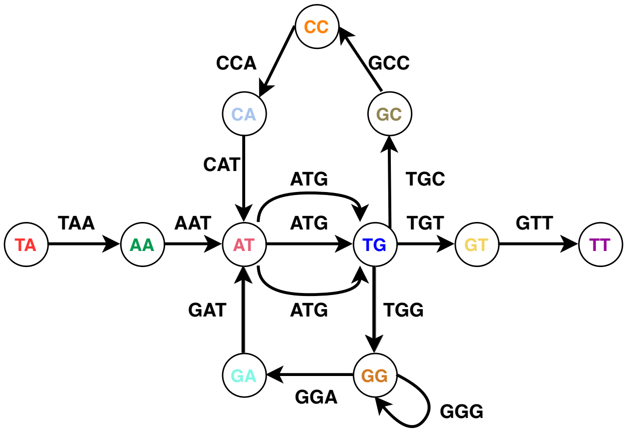


Figure 7: DEBRUIJN$_3$(TAATGCCATGGGATGTT)

2 Chapter 4

2.1 Questions for One Marks

2.1.1 Q1: What are antibiotics?

A substance that kills bactaria.

2.1.2 Q4: What is the length of amino acids in Tyrocidine B1?

10 amino acid seequence

2.1.3 Q1: What are non-ribosomal peptides?

The peptide that do not follow the rule of central dogma and are synthesized by a protein named NRP synthetase.

2.1.4 Q3: What is a mass spectrometer?

A molecular scale that shatters the moleules into pieces and then weighs the resulting fragments.

2.1.5 Q4: What is a theoretical spectrum?

The theoretical spectrum of a cyclic peptide Peptide is the collection of all of the masses of its subpeptides, in addition to the mass 0 and the mass of the entire peptide, with masses ordered from smallest to largest.

2.2 Questions for Five Marks

2.2.1 Q1:Why brute force cyclopeptide sequencing is considered infeasible?

In brute force approach of cyclopeptide sequencing we first take a peptide sequence from our known masses list, then we check whether the mass of the sequence is equal to the PARENTMASS(Spectrum). If so, we generate CYCLOSPECTRUM(Peptide) then compare it with our given spectrum. If CYCLOSPECTRUM(Peptide) has the same Cyclic spectrum as given spectrum, we got our required sequence. Otherwise we will take a new sequence and do the same approach.

There are 2 problems in this brute force approach:

1. One is, we have to check it for all possible peptide combinations as long as the MASS(current sequence) is equal to PARENTMASS of given spectrum. which is less efficient, more time consuming.

2. Another one is, we may think we will get only one sequence which will have the same MASS as PARENTMASS, but using dynamic programming, scienctists have resolve that Tyrocidine B1 has trillions (10^{14}) of sequences that has same mass as PARENTMASS of Tyrocidine B1 (which is 1322). For which we will again have to generate cyclic spectrum!!

This two problem has made brute force solution impractical to implement.

2.2.2 Q2: Write down the steps of Leaderboard algorithm for Cyclopeptide sequencing?

the input we are given is a spectrum and a integer value N. Our target is to find out the amino acid sequence from the input. The steps of Leaderboard cycloppetide algorithm is written below:

1. In the first step, Two sets are taken into consideration, Leaderboard and Leaderpeptide (top item in the Leaderboard). Initially both of them are empty.

2. EXPAND each peptide in the Leaderboard by each of the 18 different amino acid masses: *Leaderboard* ← EXPAND(*Leaderboard*)

3. Cut peptide with low score from he leaderboard: *Leaderboard* ← *TRIM(Leaderboard, Spectrum, N)*

4. Update Leaderboard if there is a higher scoring peptide in Leaderboard with *MASS(peptide)* = *PARENTSMASS(Spectrum)* and eliminate all the peptides with *MASS(peptide)* >*PARENTSMASS(Spectrum)*.

5. Repeat 2-4 steps until Leaderboard in empty.

6. Return Leaderpeptide.

2.2.3 Q: Explain are the limitations of Leaderboard cyclopeptide sequencing algorithm?

The main two main limitations of this algorithm. These are explained below:

1. We use this method because, the experimental spectrum may contain errors such as missing/false masses. This algorithm works well in case of 10% missing/false masses. But as the error increases the so does the likelihood that this algorithm will return an incorrect peptide. It has been proved that at 25% missing/false masses this algorithm gives an incorrect value.

2. Until now we have assumed that 20 amino acids are the building block of protein (named proteinogenic amino acid). But NRp contains 22 proteinogenic acids and more proteinogenic acids. This expands the building block of antibiotic from 20 to 100. This enlargement in amino acids causes problem in the existing method. the corrrect peptide now must compete with many more incorrect peptides for a place in the Leaderboard. Thus increases the chance that the correct peptide will be cut along the way. In this case, the Leaderboard cyclopeptide gives incorrect answer even if it is just 10% false/missing masses.

2.3 Questions for Ten Marks

2.3.1 Q: Build the convolution spectrum of the following integer masses and find out the masses and explain the reason to choose them: 0, 99, 113, 114, 128, 227, 257, 299. Given parent mass for this spectrum is 484

The convolution for given data is shown in Table 1.

Table 1: Convolution Spectrum for given masses

	0	**99**	**113**	**114**	**128**	**227**	**257**	**299**
0								
99	99							
113	113	14						
114	114	15	1					
128	128	29	15	14				
227	227	128	114	113	99			
257	257	158	144	143	129	30		
299	299	200	186	185	171	72	42	
484	484	385	371	370	356	257	227	185

By looking at the table, I will choose 99, 113, 114, 128, 185. Because these masses are between 57 to 200 and occurred highest 2 times.

3 Chapter 9

3.1 Questions for One Marks

3.1.1 Q1: What is a reference genome?

Reference genome is a database genome used for comparison.

3.1.2 Q2: What is the complexity of a brute force pattern matching?

$\mathcal{O}(|Text| \cdot |Patterns|)$

3.1.3 Q3: What is read mapping?

Read mapping is the process that determines where each read has high similarity to the reference genome.

3.1.4 Q4: What is a trie?

Trie is a directed acyclic graph.

3.1.5 Q5: What is a suffix trie?

A suffix trie, denoted SUFFIXTRIE(Text), is the trie formed from all suffixes of Text.

3.2 Questions for Five Marks

3.2.1 Q1: Why do we use reference genome for human genome rather than using genome assembly?

The reasons for not using genome assembly rather using reference genome is written below:

1. Genome sequencing/assembly is always computationally intensive and not perfect.

2. Genome assembly requires a lot of memory. Human genome in approximately 3 biilion nucleotide long. So using assembly will take a huge memory space.

3. Genome assembly often generate error prone contigs.

That is why, it is easier to use reference genome rather using genome assembly for human genome.

3.2.2 Q2: Why bruteforce is considered inefficient for pattern matching?

The brute force pattern matching algorithm slides each Pattern along Text, checking whether the substring starting at each position of Text matches Patterns.

This algorithm has a high runtime. The reason why the runtime of this algorithm is so high is that each string in Patterns must traverse all of Text independently. If we thing a text as a long road,brute force algorithm is analogous to loading each pattern into its own car when driving down text. This makes the complexity of the algorithm $\mathcal{O}(|Text| \cdot |Patterns|)$. This is an inefficient approach which will make the process very slow.

3.2.3 Q3: Build the suffix array for the string "panamabananas$"

List of suffixes:
panamabananas$, anamabananas$, namabananas$, amabananas$, mabananas$, abananas$, bananas$, ananas$, nanas$, anas$, nas$, as$, s$, $
Sorted suffix:
$, abananas$, amabananas$, anamabananas$, ananas$, anas$, as$, bananas$, mabananas$, namabananas$, nanas$, nas$, panamabananas$, s$

Table 2: Suffix array positions

Sorted suffixes	Starting positions
$	13
abananas$	5
amabananas$	3
anamabananas$	1
ananas$	7
anas$	9
as$	11
bananas$	6
mabananas$	4
namabananas$	2
nanas$	8
nas$	10
panamabananas$	0
s$	12

From the table, SUFFIXARRAY("panamabananas$") =$[13, 5, 3, 1, 7, 9, 11, 6, 4, 2, 8, 10, 0, 12]$

3.3 Questions for Ten Marks

3.3.1 Q: Construct trie for the following patterns. 'happy' 'happier' 'happiest' 'theme' 'themes' 'thus' 'bad'

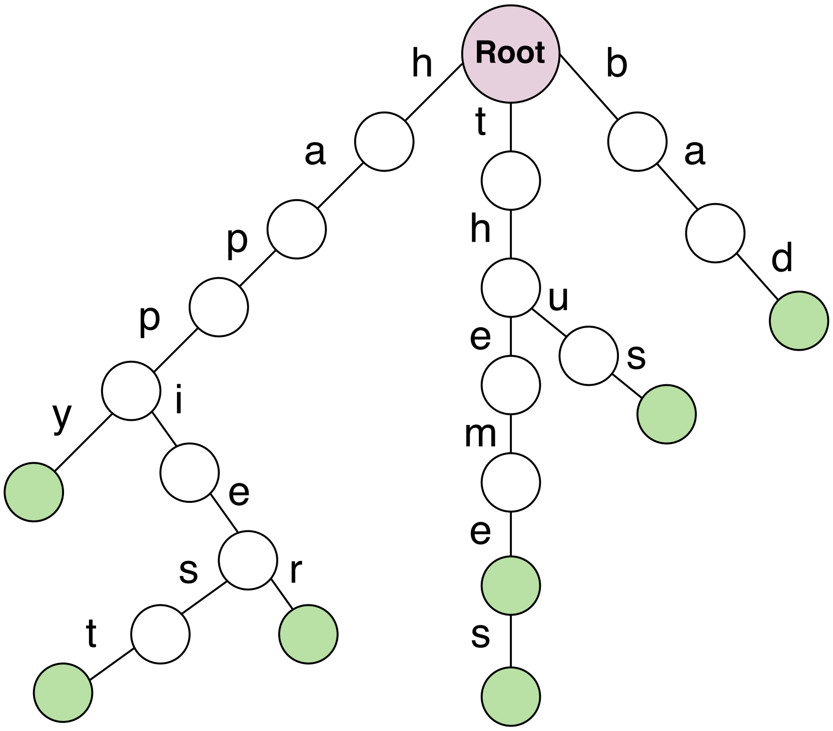


Figure 8: The trie for the following collection of strings Patterns: 'happy' 'happier' 'happiest' 'theme' 'themes' 'thus' 'bad'