ADVANCED METHODS IN BIOINFORMATICS

MASTER'S IN APPLIED COMPUTER SCIENCE AND ENGINEERING VRIJE UNIVERSITEIT BRUSSEL

ASSIGNMENT-2 (VUB 4018221FNR / ULB INFO-F439)

Read Mapper

Submitted By	LECTURER:
Aneruth Mohanasundaram (VUB)	Matthieu Defrance
0573520	

Index

I. Objective:

The main objective for this assignment is to align a large set of reads to a reference genome based on a **read mapping software** algorithm.

II. Datasets:

- **Source:**

To make our sequence alignment, our given data is **ChIP-seq** data that is technically a procedure of how proteins interact with DNA determining the regulation of gene expression. ChIP-seq helps us to identify the locations of the genome leaped by proteins. In our case this technique has been applied to Drosophila melanogaster "***Oregon-R S2***" strain.

- **Technology:**

Illumina Technology (using single-end sequencing)

- **Data files:**

Data file	Description
dm6_chr2L.fa.gz	only chromosome chr2L extracted from the **dm6.fa.gz** genome.
10k_reads.fastq.gz	only 10,000 reads extracted from **all_reads.fastq.gz.**

III. Mapping software algorithm:

- **Workflow:**

Step	Description	Code	Explanation

1	**Packages**	`import matplotlib.pyplot as plt`
		`import matplotlib.ticker as mticker`
		`from collections import defaultdict`
		`%matplotlib inline`
			These packages were used for the visualization process.

2	**Reading the data files.**	`genome_info = '/content/chr2L.fa'`
		`sequence_info='/content/10k_reads.fastq'`
			To limit computing power, I have used the reduced datasets. However, it could be easily applied to all reads and reference genome files.

3	**Reading the Genome File.**	`def readGenome(aFile):`
		`    f = open(aFile)`
		`    aval = ''.join([i.rstrip() for i in f.readlines() if not i [0] == '>'])`
		`    return aval.upper()`
			This Function is to read the genome file.
			
			- the focus here is get the targeted sequence after neglecting the followed distinctive character.
			- Since our characters consisted of mixed letters. I used. upper() to change the characters cases.

4	**Count the number of bases present.**	`def countFrequency(listToPass):`
		`    base_count = {'A':0,'G':0,'C':0,'T':0} #`
		`    for i in listToPass:`
		`        if i in base_count.keys():`
		`            base_count[i] += 1`
		`    return base_count`
			This Function counts the frequency of each base present in our genome.

5	**Reading sequences.**	`def readSequence(seq):`
		`    f = open(seq,'r') # Reading the file`
		`    sequence,quality = [], []`
		`    while True:`
		`        f.readline()`
		`        seq = f.readline().rstrip() # Assigning our sequence to a variable`
		`        f.readline()`
			Since we know the format of a reference genome. This function extracts all the sequences considering the Quality score from the 10k_reads. fastq dataset.
			- First, I assigned our sequence to a variable.

		qual = f.readline().rstrip() # Fetching our quality if len(seq) == 0: break # Exit the while loop sequence.append(seq) # Appending our sequence to our newly created empty list quality.append(qual) # Appending our quality score to newly created empty list return sequence,quality # Returns the values in tuple format	- **I fetched our corresponding quality score.** - **After testing if I do have a sequence, I appended it to an empty list.** - **the readSequence function will return the sequence along with its corresponding score in a tuple format.**

6	**Quality Score**	# Functions to calculate the quality score
		def qulaityScore(astring):
		return ord(astring) - 33 # Ord function gives the ascii values
		
		def fetchQualityScore(alist):
		blist = [qulaityScore(j) for i in list(alist) for j in i] # Consider each character and calculates its quality score
		return blist
			Quality scores are encoded in ASCII code, and they represent the probability of the base. Here we made two functions:
			- **The** qualityScore **function takes the quality score encoded in ASCII and subtracts the first element of the ASCII table from it.**
			- **The** fetchQualityScore **loops on all the sequence in order to calculate all the quality scores.**

7	**The mapping algorithm: Suffix Array** [1]	def suffixArray(s):
		suffixes = [(s[i:], i) for i in range(len(s))]
		suffixes.sort(key=lambda x: x [0])
		return {s [0] for s in suffixes}
			I chose to use to suffix array algorithm to sort all the suffixes of the given string S. In our case it will take the targeted sequence and sort it to match it later.

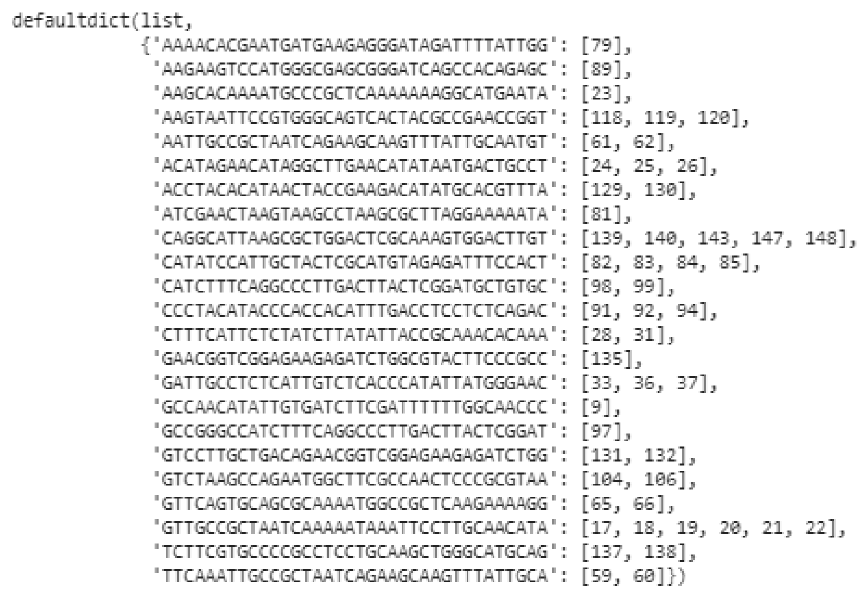
8	**Linear Scanning**	def linearScan(aString,sequenceCheck):
		count_dict = {} # Creating a dictioanry to strore number of occurence per read
		for i in sequenceCheck:
		if i in aString:
		count_dict[i] = aString.index(i)
		return count_dict
			I chose this algorithm to count all our alignments in a dictionary. The choice of this linear model algorithm was because of the reduced time complexity O(n).
			This function will take as an input the sequence list and after scanning it will return a dictionary with the number of alignments detected.

9	**Application**	genome = readGenome(genome_info)
		test_val = genome[:10000] # To read the first 10000 genome
		sequence_list = readSequence(sequence_info)[0]
		# quality_list = readSequence(sequence_info)[1]
		# qulaity_score = fetchQualityScore(quality_list)
		
		print(f'The first 5 sequence list is {sequence_list[:5]}')
		print(f'\nNumber of bases present is {countFrequency(genome)}')
			Here, I am applying the pre-defined functions to do the read mapping.
			- **Reading the genome from the chr2L.fa dataset.**
			- **Selecting the first 10000 reads from the genome file.**
			- **Printing the first 5 sequences along with the frequency of each base in our genome.**

10	**Merging all suffxarray in one String.**	val = ''. join(suffixArray(test_val))	**Here we are using the suffix array pre-defined function to add our reads in a sorted way to the variable Val.**
11	**Allignments.**	linearScan(sequence_list, val)	**With LinearScan; we are scanning all the sequence list from the 10k_reads file along with our sorted reads from the suffix array.**

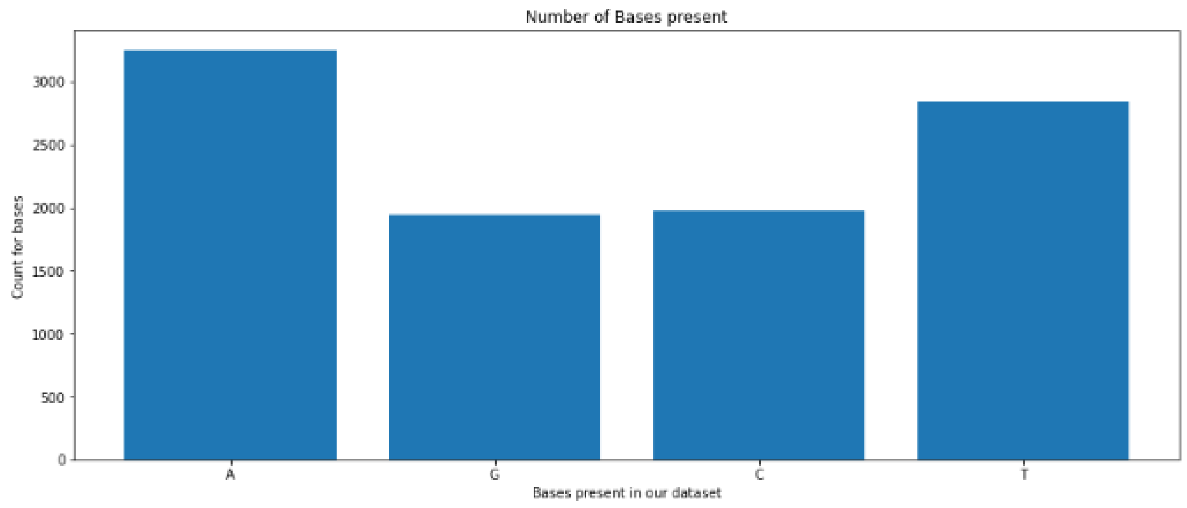
[1]*https://stackoverflow.com/questions/54642311/function-for-suffix-array-python*

IV. Visualization: Stats and plots



```
defaultdict(list,
            {'AAAACACGAATGATGAAGAGGGATAGATTTTATTGG': [79],
             'AAGAAGTCCATGGGCGAGCGGGATCAGCCACAGAGC': [89],
             'AAGCACAAAATGCCCGCTCAAAAAAAGGCATGAATA': [23],
             'AAGTAATTCCGTGGGCAGTCACTACGCCGAACCGGT': [118, 119, 120],
             'AATTGCCGCTAATCAGAAGCAAGTTTATTGCAATGT': [61, 62],
             'ACATAGAACATAGGCTTGAACATATAATGACTGCCT': [24, 25, 26],
             'ACCTACACATAACTACCGAAGACATATGCACGTTTA': [129, 130],
             'ATCGAACTAAGTAAGCCTAAGCGCTTAGGAAAAATA': [81],
             'CAGGCATTAAGCGCTGGACTCGCAAAGTGGACTTGT': [139, 140, 143, 147, 148],
             'CATATCCATTGCTACTCGCATGTAGAGATTTCCACT': [82, 83, 84, 85],
             'CATCTTTCAGGCCCTTGACTTACTCGGATGCTGTGC': [98, 99],
             'CCCTACATACCCACCACATTTGACCTCCTCTCAGAC': [91, 92, 94],
             'CTTTCATTCTCTATCTTATATTACCGCAAACACAAA': [28, 31],
             'GAACGGTCGGAGAAGAGATCTGGCGTACTTCCCGCC': [135],
             'GATTGCCTCTCATTGTCTCACCCATATTATGGGAAC': [33, 36, 37],
             'GCCAACATATTGTGATCTTCGATTTTTTGGCAACCC': [9],
             'GCCGGGCCATCTTTCAGGCCCTTGACTTACTCGGAT': [97],
             'GTCCTTGCTGACAGAACGGTCGGAGAAGAGATCTGG': [131, 132],
             'GTCTAAGCCAGAATGGCTTCGCCAACTCCCGCGTAA': [104, 106],
             'GTTCAGTGCAGCGCAAAATGGCCGCTCAAGAAAAGG': [65, 66],
             'GTTGCCGCTAATCAAAAATAAATTCCTTGCAACATA': [17, 18, 19, 20, 21, 22],
             'TCTTCGTGCCCCGCCTCCTGCAAGCTGGGCATGCAG': [137, 138],
             'TTCAAATTGCCGCTAATCAGAAGCAAGTTTATTGCA': [59, 60]})
```

This is the output after mapping our genome reads to the sequence file keys refers to the reads and value represents the index of that reads. This shows the output for dataset of 10000 dataset.



Here, we are plotting the count of bases present in our dataset: A; G; C; T.

Note: *To minimize the computing time I have converted all the suffix array list to string where it reduces the computation time. This is done in order to reduce the usage of double for loops (Brute force method). Full code can be found at https://github.com/Aneruth/Advance-Bioinformatics/tree/main/Assignment%202.*