Algorithms and Compression Techniques For Genome Sequencing

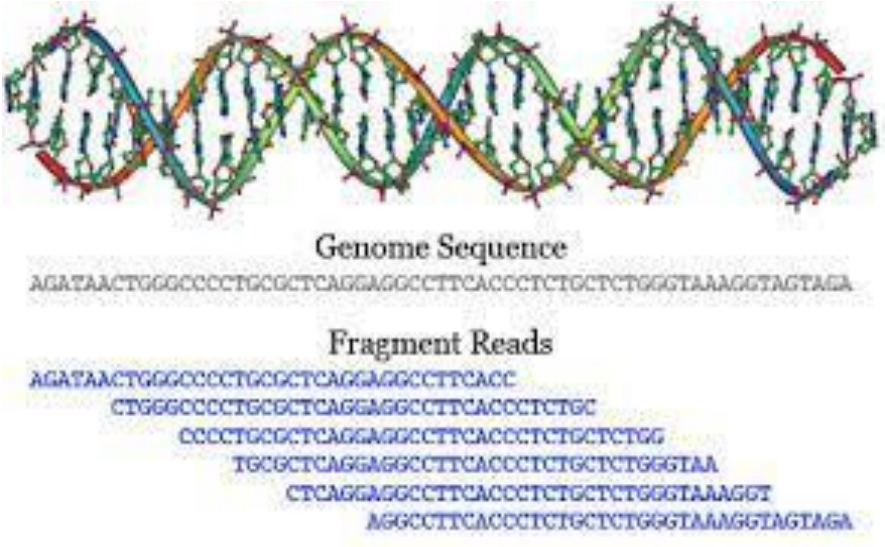
Department of Electrical Engineering, IIT Hyderabad

under the guidance of

Dr. Shashank Vatedka
Assistant Professor
Department of Electrical Engineering
Indian Institute of Technology, Hyderabad
Kandi, Sangareddy, Telengana 502285

Contents

1. Introduction
2. Motivation
3. Literature Survey
4. Abstract
5. Sequencing DNA
6. FASTQ read
7. Read Alignment Problem
8. Naive Exact Matching
9. Boyer-Moore Algorithm
10. Preprocessing
11. Indexing and k-Mer
12. Conclusion



Introduction

- DNA Sequencing has become so inexpensive and so good at delivering a lot of data very quickly that sequencers are being used all over the field of life sciences.
- Scientists are applying and refining new ways of sequencing to study people's genomes.
- These study answer some of the most fundamental questions about evolutions,patterns of migration, ancient civilizations etc.

Ancient DNA from the Indus Valley Civilization was found in this individual buried at India's Rakhigarhi archaeological site. VASANT SHINDE

Genome of nearly 5000-year-old woman links modern Indians to ancient civilization

By Michael Price | Sep. 5, 2019 , 2:00 PM

Genome sequencing central to COVID-19 response

25 February 2021

Motivation

- Since the introduction of the Sanger sequencing technology in 1977 by Sanger , we got explosion of sequence data. The cost of storage, processing, and analyzing the data is getting excessively high. As a result, it is extremely important that we develop efficient data compression and data reduction techniques.
- Understanding computational genomic algorithms is key to understanding where they will succeed and where they will fail. For example the de Novo shotgun assembly problem.
- This is a very active area of research where scientists from life sciences and computer science field work together.

Literature Survey

[1] Subrata Saha and Sanguthevar Rajasekaran, "**Efficient algorithms for the compression of FASTQ files**," *2014 International Conference on Bioinformatics and Biomedicine*

[2] Shengyu Lu, Hanping Chen, Lifa Peng,Beizhan Wang,Hongji Wang and Xuize Zhou,"**A compression algorithm of fastq file based on distribution characteristics analysis**", *The 13th International Conference on Computer Science & Education (ICCSE 2018),*Columbo, Sri Lanka

[3] Algorithms on Strings,Trees and Sequences, *Computer Science and Computational Biology*, by Dan Gusfield

[4] Langmead,B., Trapnell,C., Pop,M., and Salzberg,S.L. (2009), "**Ultrafast and memory-efficient alignment of short DNA sequences to the human genome".** Genome Biol, 10, R25.

Abstract

Various algorithms for analyzing ,encoding and compressing real DNA sequencing data are studied and implemented in this project. Some of the insights about designing the algorithm were studied from the research paper,“Efficient algorithms for the compression of FASTQ files” and few others are listed under the Literature Survey..

The three main algorithms discussed in these papers are:

RFRC-reads are clustered based on a hashing scheme. Followed by this clustering, a representative string is chosen from each cluster of reads. Compression is independently done for each cluster. In particular, the representative string in any cluster is used as a reference to compress other reads in this cluster.

RFQSC- It compresses quality score sequences by forming clusters of similar sequences.

Here clusters are formed without the employment of hashing. Each sequence in a particular cluster is then compressed with respect to its representative sequence. The representative sequence can be a brand new sequence or one of the quality sequences residing in the cluster.

MDC-Metadata is compressed by detecting the redundant information.

In this paper presentation, we will briefly explain the proposed novel algorithms for encoding and compressing FASTQ files.

Simulation results show that offline algorithms perform better than online algorithms.

How DNA Sequencers Sequence Genome?

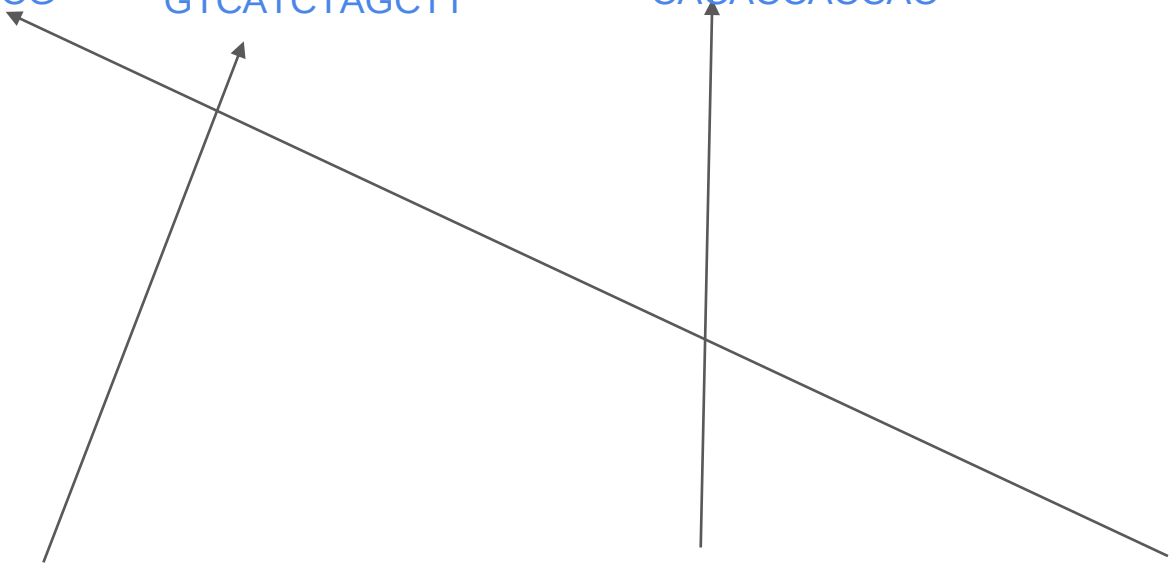
- Dna sequencers are not good at reading long stretches of DNA.
- They are good at reading short stretches of DNA but lots and lots of them at at time.by repeatedly reading randomly selected substrings from the input DNA.
- Snippets are called sequencing reads.Their length are very small compared to input DNA.

Input DNA
CCATAGTATATCTCGGCTCTAGGCCCTCATTTTTT
CCATAGTATATCTCGGCTCTAGGCCCTCATTTTTT
CCATAGTATATCTCGGCTCTAGGCCCTCATTTTTT

Cut into snippets
CCATAGTA TATCTCGG CTCTAGGCCCTC ATTTTTT
CCA TAGTATA TATCTCGGCTC TAGGCCC TCATTTTTT
CCATAGT ATATCTCGGC TCTA GGCCC TCATTTTTT

Genome Sequencing READS

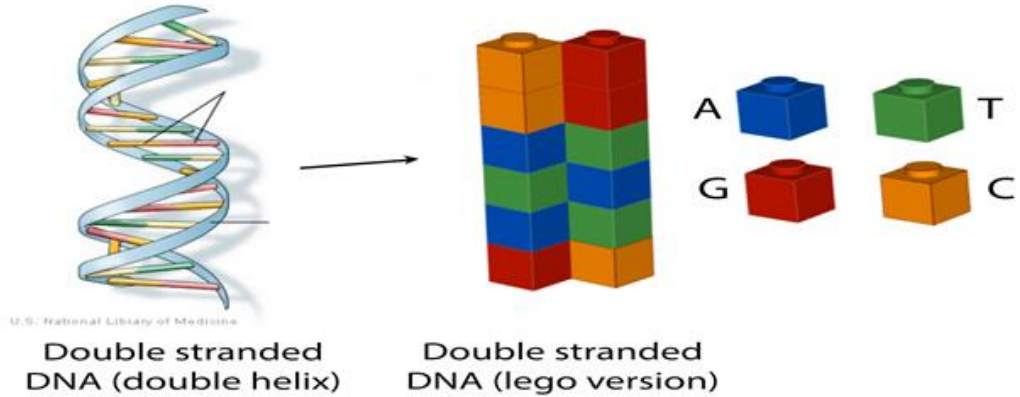
Reads: GTCTGCTCG GTCATCTAGCTT CACACCACCAC



Genome: CGTCGGTCATCTAGCTTGGTGTGACCACACACCACACCACCACCACTCTGTGTCTGCTCG

Sequencing DNA

DNA is the kind of molecule that encodes our genome.Genome is the set of chromosomes and it is written in a language that consists of 4 letters - {A,C,G,T}



FASTQ FILE READ

```
NAME-@ERR266411.1 HS18_09233:8:1307:10911:3848#168/1',
 SEQUENCE-
'TAAACAAGCAGTAGTAATTCCTGCTTTATCAAGATAATTTTTCGACTCATCAGAAATATCCGAAAGTGTTAACTTCTGCGTCATGGAAGC
GATAAAACTC',
IGNORE-'+',
 BASE_QUALITY-
'B@DFEFFFGEGGGHEHGHGHGGGGHIFGFIFHICFGHGHGJGHFGHGIHEHGGHJGFEFHGHEGGHHGHIFGFGDIFGGFGGGFHGGGHG
GGAGIFGGCG',
```

- Sequencing read is encoded in FASTQ format(lines of information)
  - **Name :** Metadata information
  - **Sequence** : Raw sequence
  - **Base quality :** Corresponding base quality of characters to sequence.

Meta data compression

NAME-@ERR266411.1 HS18_09233:8:1307:10911:3848#168/1',

The redundant information is utilised for meta data compression.

Base Quality Read In FASTQ Format

Example of base quality read:
Characters in base quality match up with corresponding characters in sequence line

Base quality is calculated using this formula
Q = -10 log10(p)
Where ,
Q is base quality
p is probability that the base call is incorrect

Example:
Q = 10 implies 1 in 10 chance call is incorrect
Q = 20 implies 1 in 100 chance call is incorrect

BASE QUALITIES:

```
#Convert  quality scores to ASCII symbols
def QTophred33(Q):
    return chr(Q+33)
```

```
[4] QTophred33(10)
```

```
'+'
```

```
#Convert ASCII symbols to quality scores
def phred33ToQ(qual):
    return ord(qual) - 33
```

```
[ ] phred33ToQ('#')
```

```
2
```

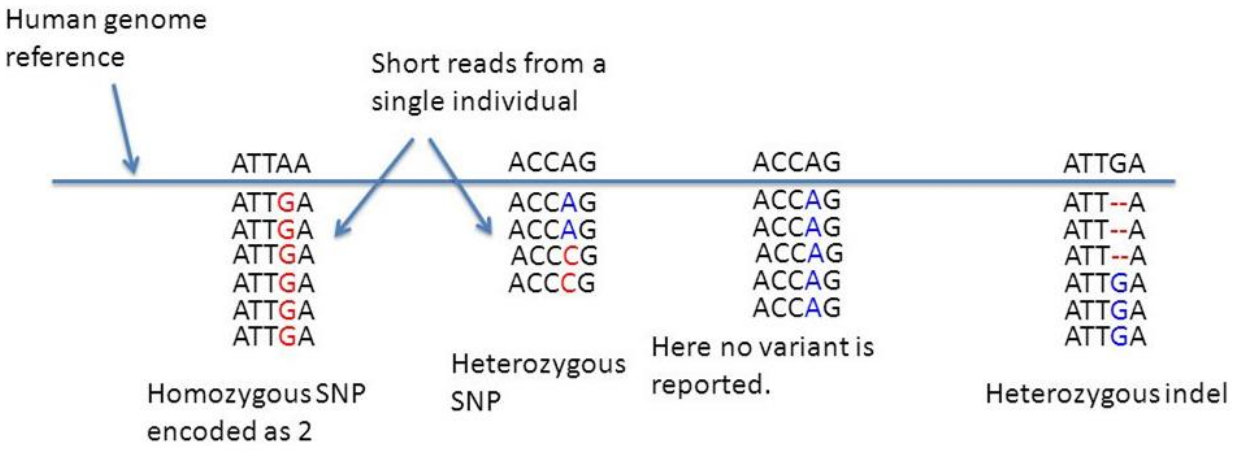
Usual ASCII encoding is phred+33

Read Alignment Problem

- Reading the very long sequence of genome is very challenging.
- Even unrelated humans have genome similarity upto 99 percent.
- The size of sequencing read is very less compared to overall genome sequencing length.
- The read alignment of DNA sequences can be done using 2 methods-
  - With reference
  - Without reference

Algorithms for Sequencing Genome Data

- Genome sequence can be encoded as a string containing characters from the alphabet {A,G,T,C}.
- The ability to write a genome as a string of characters helps in analyzing it using computational methods.
- So,sequencing using string is done -data structure algorithms are used.
  - Naive Exact matching
  - Boyer Moore algorithm



Naive Exact Matching

- Exact Matching is a computational method of matching a pattern P to the text T exactly.

Naive Exact Matching

- Exact Matching is a computational method of matching a pattern P to the text T exactly.
- For example consider the pattern, text pair given below.

  P : word

  T : There would have been a time for such a word.

  The most straightforward way is to align the pattern P with text T in every possible way and check if in any of these alignments all the characters of P match with corresponding character of T.

Naive Exact Matching

P : word

T : There would have been a time for such a word.

word word word word word word word word word

word word word word word word word word

word word word word word word word word

word word word word word word word word

word word word word word word word word

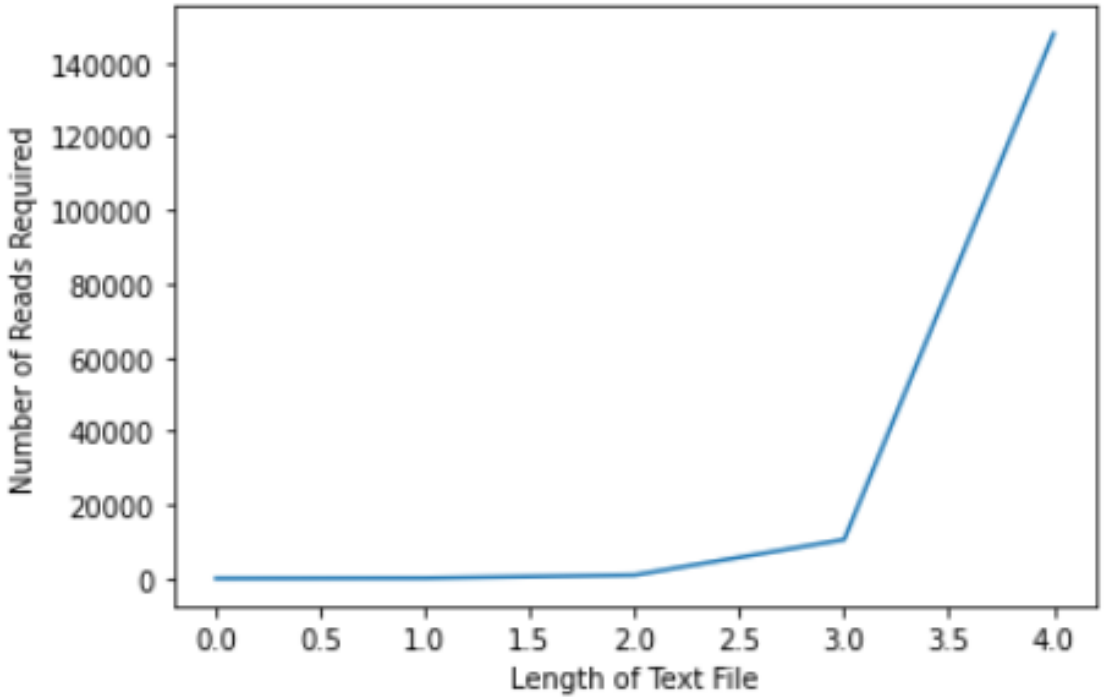
Naive algorithm practical

```
#Naive Exact Matching
def naive(p,t):
  #p:read sequence
  #t:genome sequence
  occurances=[]
  nmreads=0 #initializing number of reads to 0
  for i in range(len(t)-len(p)+1):
    match=True
    for j in range(len(p)):
      if not t[i+j]==p[j]:
        match=False
        break
    if match:
      occurances.append(i)
    nmreads+=1
  return occurances,nmreads
```

```
t = "AGCTTAGATAGC"
p = "AG"
naive(p,t)
```

```
([0, 5, 9], 11)
```

Naive Algorithm Performance



Boyer-Moore Algorithm

- Exact matching algorithm : better than naive algorithm, extended version of naive algorithm
  a. Alignments are done from left to right
  b. Characters comparison are done from right to left
- For example consider the pattern, text pair given below.
  P: ANNU
  T: ANKU ANNU STUDY TOGETHER IN SAME CLASS.

  In this example,
  T: ANKU ANNU STUDY TOGETHER IN SAME CLASS.
  P: ANNU
  → Alignments
  ← Character comparison
- Boyer-Moore algorithm does some preprocessing on the pattern P to create lookup tables which are used to implement Bad Character rule, Good Suffix Rule.

Boyer-Moore : Bad Character Rule

- Rules are followed as explained in previous slide
  - When mismatch between characters of text T and pattern P occurs ,skip the alignments till mismatch becomes the match
  - When there is no possibility for the match for any alignments,let the pattern P to pass over the the mismatched character in text.

Example
case 1:
T: GCTTCTGT CTCGTCACT
P: CCTTTTGT
Result: CCTTTTGT
C will go to T position, skip 3 alignments!

case2:
T: GCTTCTGTAC
P: CCTTTTGC
Result: CCTTTTGC
P moves past mismatched character

Boyer-Moore : Good Suffix Rule

- Rules are followed as explained in previous slide
  - When match between characters of text T and pattern P occurs ,skip the alignments till there are no mismatches between P and T
  - When there is no possibility for the match for any alignments,let the pattern P to pass over the the mismatched character in text.

Example
case 1:
T: GCTTCTGT CTCGTCACT
P: TGTTTTGT
Result: TGTTTTGT
C will go to T position, skip 3 alignments!

case2:
T: GTCTCTTACTTACTTCATTAC
P: CCTTACTTAC
Result: CCTTACTTAC
P moves past matched character T

Preprocessing-lookup tables

1.Bad character rule
2.Good suffix rule

Example-bad character rule
T=AATCCTC
P=TCGC, look at row T and column G ,offset of 1 is calculated and it will skip 1 alignment.

	T	C	G	C
A	0	1	2	3
C	0	None	0	None
G	0	1	None	0
T	None	0	1	2

Boyer-Moore : Code Snippet

```
def boyer_moore(p, p_bm, t):
    i = 0
    occurrences = []
    nmreads=0
    while i < len(t) - len(p) + 1:
        shift = 1
        mismatched = False
        for j in range(len(p)-1, -1, -1):
            if p[j] != t[i+j]:
                skip_bc = p_bm.bad_character_rule(j, t[i+j])
                skip_gs = p_bm.good_suffix_rule(j)
                shift = max(shift, skip_bc, skip_gs)
                mismatched = True
                break
        if not mismatched:
            occurrences.append(i)
            skip_gs = p_bm.match_skip()
            shift = max(shift, skip_gs)
        i += shift
        nmreads+=1
    return occurrences,nmreads
```

Reference-free Reads Compression

- This algorithm clusters the reads based on overlaps and similarity scores. For each cluster a consensus sequence is created which act as reference string to all other reads in cluster.
- All the reads in the cluster are compressed using the consensus as the reference.
- Two reads are said to be neighbors of each other if they have a large overlap (with a small Hamming distance in the overlapping region).
- We find the neighbors of each read in steps 1 and 2 and use this neighborhood information to cluster the reads and perform compression.
- In step 1 we find the potential neighbors of each read and in step 2 we find the neighbors of each read.

Reference-free Reads Compression

```
#Python hash data structure
t='GTCCTGCTG'
hash_table={'GTC':[0,1,2],'TCC':[3],'CCT':[5],'CTG':[4,6],'TGC':[7],'GCT':[9],'CTG':[8,10]}
hash_table['GTC']
```

```
[0, 1, 2]
```

```
[3] hash_table['CTG']
```

```
[8, 10]
```

Quality Scores Compression

- This score can be thought of as the probability that this base is correct.
- At first we count the frequency of each character in the quality score sequences and identify the top m characters based on frequencies (where the value of m is chosen to get the best compression).
- Hamming distance is calculated for each quality score sequence s ,if distance is least let's say between si and and s ,then si will serve as reference for all other s's in sequence s.
- S-quality score sequence-{s1,s2,s3……...sn}

Preprocessing

- How was Boyer-Moore algorithm different from Naive algorithm?

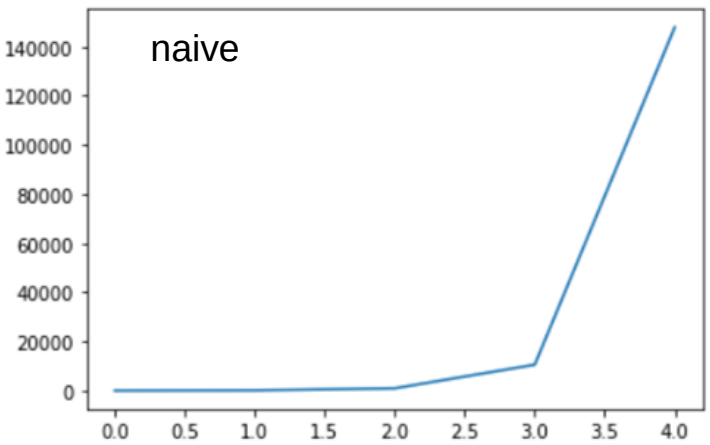


Fig1 : Number of reads versus text size performance for naive

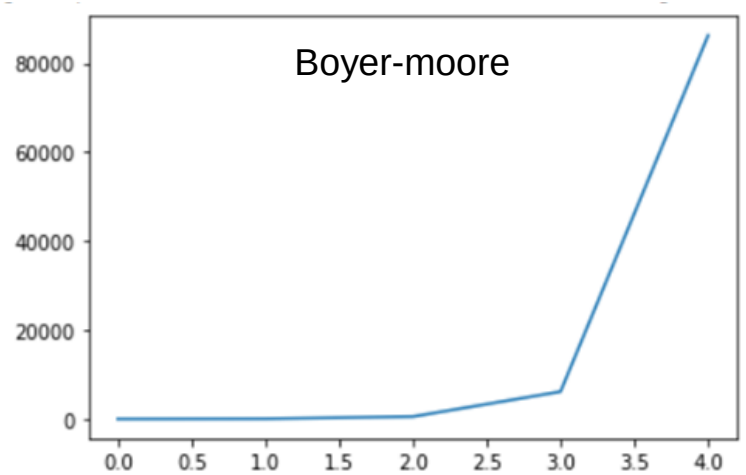


Fig2 :Number of reads versus text size performance for naive

Preprocessing

- How was Boyer-Moore algorithm different from Naive algorithm?

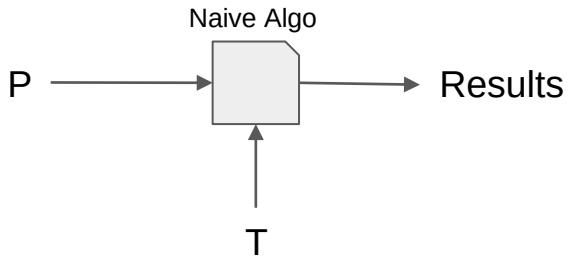
  The Boyer-Moore algorithm **preprocesses** the pattern P using a lookup table in order to use the the **bad-character and good-suffix rule.**

Preprocessing

- How was Boyer-Moore algorithm different from Naive algorithm?

  The Boyer-Moore algorithm **preprocesses** the pattern P using a lookup table in order to use the the **bad-character and good-suffix rule.**

- **Preprocessing : Naive algorithm**



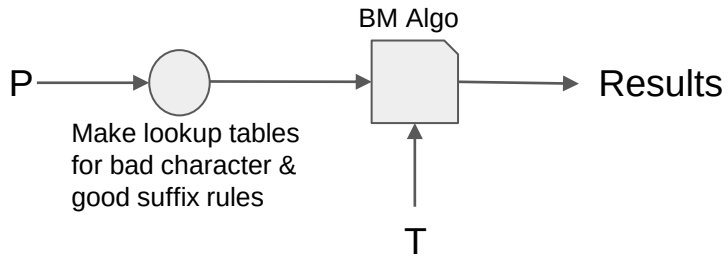
No preprocessing is done in Naive algorithm.

Preprocessing

- How was Boyer-Moore algorithm different from Naive algorithm?

  The Boyer-Moore algorithm **preprocesses** the pattern P using a lookup table in order to use the the **bad-character and good-suffix rule.**

- **Preprocessing : Boyer-Moore algorithm**



Preprocessing in Boyer-Moore algorithm.

Preprocessing

- What is the **cost** of preprocessing?

Preprocessing

- What is the **cost** of preprocessing?
  Preprocessing the data can be costly to do once but when the same data is executed multiple times it can be helpful.
- Which is *better* ? Preprocessing the text or the pattern?

Preprocessing

- What is the **cost** of preprocessing?
  Preprocessing the data can be costly to do once but when the same data is executed multiple times it can be helpful.
- Which is *better* ? Preprocessing the text or the pattern?
  At first glance preprocessing the text might not seem a good idea because text files are generally big (very big) when compared to pattern P but this can be really useful when we are in a situation where we have to match the pattern to the text multiple times keeping the text file same. Various encoding techniques (compression techniques) can be used to encode the text file before-hand.

- Algorithm that preprocesses ‘Text T’ is known as offline algorithm. Otherwise, algorithm is online.
  Examples:
  1. Naive Algorithm
  2. Boyer-Moore Algorithm
  3. Web Search Engine
  4. Read Alignment Problem for Genome Sequence

Preprocessing

- What is the **cost** of preprocessing?
  Preprocessing the data can be costly to do once but when the same data is executed multiple times it can be helpful.
- Which is *better* ? Preprocessing the text or the pattern?
  At first glance preprocessing the text might not seem a good idea because text files are generally big (very big) when compared to pattern P but this can be really useful when we are in a situation where we have to match the pattern to the text multiple times keeping the text file same. Various encoding techniques (compression techniques) can be used to encode the text file before-hand.

- Algorithm that preprocesses ‘Text T’ is known as offline algorithm. Otherwise, algorithm is online.
  Examples:
  1. Naive Algorithm **: Online**
  2. Boyer-Moore Algorithm
  3. Web Search Engine
  4. Read Alignment Problem for Genome Sequence

Preprocessing

- What is the **cost** of preprocessing?
  Preprocessing the data can be costly to do once but when the same data is executed multiple times it can be helpful.
- Which is *better* ? Preprocessing the text or the pattern?
  At first glance preprocessing the text might not seem a good idea because text files are generally big (very big) when compared to pattern P but this can be really useful when we are in a situation where we have to match the pattern to the text multiple times keeping the text file same. Various encoding techniques (compression techniques) can be used to encode the text file before-hand.

- Algorithm that preprocesses ‘Text T’ is known as offline algorithm. Otherwise, algorithm is online.
  Examples:
  1. Naive Algorithm **: Online**
  2. Boyer-Moore Algorithm : **Online**
  3. Web Search Engine
  4. Read Alignment Problem for Genome Sequence

Preprocessing

- What is the **cost** of preprocessing?
  Preprocessing the data can be costly to do once but when the same data is executed multiple times it can be helpful.
- Which is *better* ? Preprocessing the text or the pattern?
  At first glance preprocessing the text might not seem a good idea because text files are generally big (very big) when compared to pattern P but this can be really useful when we are in a situation where we have to match the pattern to the text multiple times keeping the text file same. Various encoding techniques (compression techniques) can be used to encode the text file before-hand.

- Algorithm that preprocesses ‘Text T’ is known as offline algorithm. Otherwise, algorithm is online.
  Examples:
  1. Naive Algorithm **: Online**
  2. Boyer-Moore Algorithm : **Online**
  3. Web Search Engine : **Offline**
  4. Read Alignment Problem for Genome Sequence

Preprocessing

- What is the **cost** of preprocessing?
  Preprocessing the data can be costly to do once but when the same data is executed multiple times it can be helpful.
- Which is *better* ? Preprocessing the text or the pattern?
  At first glance preprocessing the text might not seem a good idea because text files are generally big (very big) when compared to pattern P but this can be really useful when we are in a situation where we have to match the pattern to the text multiple times keeping the text file same. Various encoding techniques (compression techniques) can be used to encode the text file before-hand.

- Algorithm that preprocesses ‘Text T’ is known as offline algorithm. Otherwise, algorithm is online.
  Examples:
  1. Naive Algorithm **: Online**
  2. Boyer-Moore Algorithm : **Online**
  3. Web Search Engine : **Offline**
  4. Read Alignment Problem for Genome Sequence : **Offline** (text T does not change)

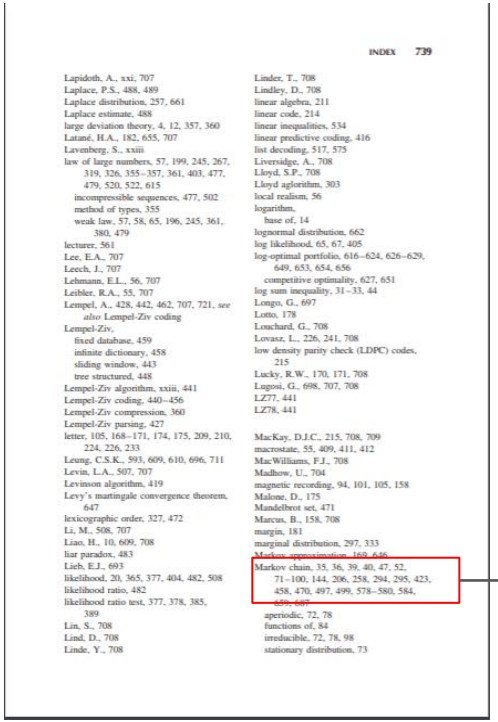
Indexing and k-mer indexes

- **Target Problem (Read Alignment Problem):** Large collection of read collections are to be matched with the genome sequence (text ‘T’) .

Indexing and k-mer indexes

- **Target Problem (Read Alignment Problem):** Large collection of read collections are to be matched with the genome sequence (text ‘T’) .
  We know that genome sequence is similar across the individuals of same species and hence it might be good to preprocess the genome sequence and come up with compression algorithm which can help the matching of read collections to the genome sequence.

Indexing (Textbook Analogy)



INDEX 739

Lapidoth, A., xxi, 707
Laplace, P.S., 488, 489
Laplace distribution, 257, 661
Laplace estimate, 488
large deviation theory, 4, 12, 357, 360
Latané, H.A., 182, 655, 707
Lavenberg, S., xxiii
law of large numbers, 57, 199, 245, 267, 319, 326, 355–357, 361, 403, 477, 479, 520, 522, 615
  incompressible sequences, 477, 502
  method of types, 355
  weak law, 57, 58, 65, 196, 245, 361, 380, 479
lecturer, 561
Lee, E.A., 707
Leech, J., 707
Lehmann, E.L., 56, 707
Leibler, R.A., 55, 707
Lempel, A., 428, 442, 462, 707, 721, *see also* Lempel-Ziv coding
Lempel-Ziv,
  fixed database, 459
  infinite dictionary, 458
  sliding window, 443
  tree structured, 448
Lempel-Ziv algorithm, xxiii, 441
Lempel-Ziv coding, 440–456
Lempel-Ziv compression, 360
Lempel-Ziv parsing, 427
letter, 105, 168–171, 174, 175, 209, 210, 224, 226, 233
Leung, C.S.K., 593, 609, 610, 696, 711
Levin, L.A., 507, 707
Levinson algorithm, 419
Levy's martingale convergence theorem, 647
lexicographic order, 327, 472
Li, M., 508, 707
Liao, H., 10, 609, 708
liar paradox, 483
Lieb, E.J., 693
likelihood, 20, 365, 377, 404, 482, 508
likelihood ratio, 482
likelihood ratio test, 377, 378, 385, 389
Lin, S., 708
Lind, D., 708
Linde, Y., 708
Linder, T., 708
Lindley, D., 708
linear algebra, 211
linear code, 214
linear inequalities, 534
linear predictive coding, 416
list decoding, 517, 575
Liversidge, A., 708
Lloyd, S.P., 708
Lloyd aglorithm, 303
local realism, 56
logarithm,
  base of, 14
lognormal distribution, 662
log likelihood, 65, 67, 405
log-optimal portfolio, 616–624, 626–629, 649, 653, 654, 656
  competitive optimality, 627, 651
log sum inequality, 31–33, 44
Longo, G., 697
Lotto, 178
Louchard, G., 708
Lovasz, L., 226, 241, 708
low density parity check (LDPC) codes, 215
Lucky, R.W., 170, 171, 708
Lugosi, G., 698, 707, 708
LZ77, 441
LZ78, 441

MacKay, D.J.C., 215, 708, 709
macrostate, 55, 409, 411, 412
MacWilliams, F.J., 708
Madhow, U., 704
magnetic recording, 94, 101, 105, 158
Malone, D., 175
Mandelbrot set, 471
Marcus, B., 158, 708
margin, 181
marginal distribution, 297, 333
Markov approximation, 160, 646
Markov chain, 35, 36, 39, 40, 47, 52, 71–100, 144, 206, 258, 294, 295, 423, 458, 470, 497, 499, 578–580, 584, 652, 697
  aperiodic, 72, 78
  functions of, 84
  irreducible, 72, 78, 98
  stationary distribution, 73

Markov Chains

Source: Thomas M. Cover Text

Indexing DNA

Index of T
C G T G C : 0

T : C G T G C G T G C T T

Indexing DNA

Index of T
C G T G C : 0
G T G C G : 1

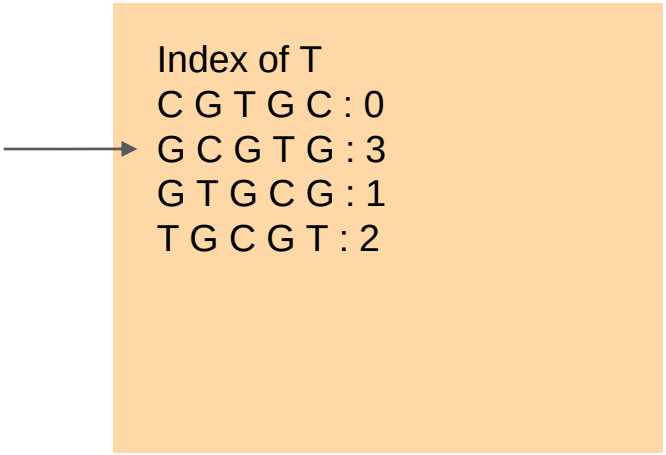
T : C G T G C G T G C T T

Indexing DNA

Index of T
C G T G C : 0
G T G C G : 1
T G C G T : 2

T : C G T G C G T G C T T

Indexing DNA



T : C G T G C G T G C T T

Indexing DNA

T : C G T G C G T G C T T

Indexing DNA

Index of T
C G T G C : 0 , 4
G C G T G : 3
G T G C G : 1
G T G C T : 5
T G C G T : 2
T G C T T : 6

5-mer index

T : C G T G C G T G C T T

Querying the index

Index of T
C G T G C : 0 , 4
G C G T G : 3
G T G C G : 1
G T G C T : 5
T G C G T : 2
T G C T T : 6

5-mer index

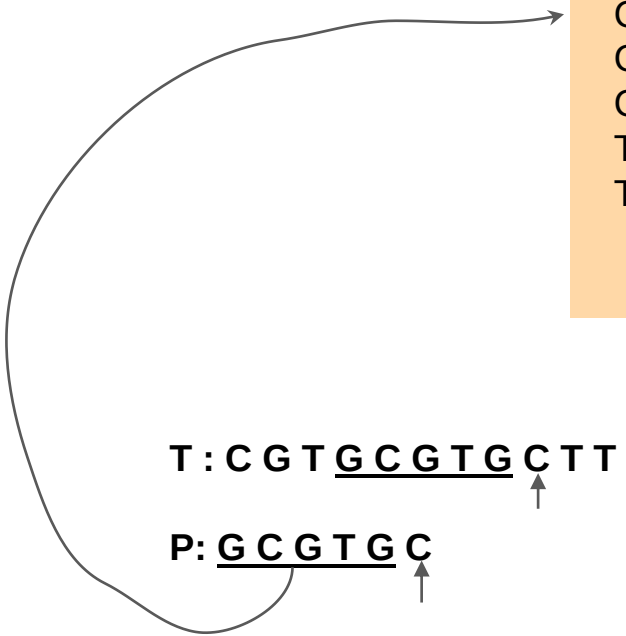
T : C G T G C G T G C T T

P: G C G T G C

Querying the index

Index of T
C G T G C : 0 , 4
G C G T G : 3
G T G C G : 1
G T G C T : 5
T G C G T : 2
T G C T T : 6

5-mer index



Querying the index

Index of T
C G T G C : 0 , 4
G C G T G : 3
G T G C G : 1
G T G C T : 5
T G C G T : 2
T G C T T : 6

5-mer index

T : C G T G C G T G C T T

P: G C G T G C

Querying the index

Index of T
C G T G C : 0 , 4
G C G T G : 3
G T G C G : 1
G T G C T : 5
T G C G T : 2
T G C T T : 6

5-mer index

Querying the index

Index of T
C G T G C : **0** , **4**
G C G T G : 3
G T G C G : 1
G T G C T : 5
T G C G T : 2
T G C T T : 6

5-mer index

T : C G T G C G T G C T T

P: G C G T G C

k-Mer Matching of the Index

- After dividing and encoding (preprocessing) the entire text into multiple k-mers and storing them in the index, how do we find the substring of pattern P in the index?

k-Mer Matching of the Index

Binary Search:
T : G T G C G T G G G G G
P: G C G T G G

1. create the index file.

Binary Search

Binary Search:

T : G T G C G T G G G G G

P: G C G T G G

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

Binary Search

Binary Search:

T : G T G C G T G G G G G

P: G C G T G G

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

After first bisection

Binary Search

Binary Search:

T : G T G C G T G G G G G

P: G C G T G G

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

Match at offset 7

After second bisection

Binary Search

- This algorithm is very useful in k-mer matching of index because the index file is sorted alphabetically.
- Time Complexity: Let's assume the worst case where the match occurs at last bisection and so `O(log(n))`
- Python command : `bisect_left(index table, k-mer)`

Conclusion

- Exploration of data structure algorithms are used for analyzing DNA sequencing data.
- Algorithms are used for overcoming read alignment problem
- Various algorithms are used to encode and sequence genome data.
- Naive algorithm and Boyer-Moore algorithm is studied for reference based DNA sequencing and their performance is compared.
- Implementation of matching algorithm is done in python.
- Offline algorithm for encoding and compressing the genome file is studied.
- Compression techniques of FASTQ file format is studied.
    - Reference free Read compression
    - Meta data compression