CS CM122 Discussion

Apr 7 2023

TA contact information

Luke Li

luke0321lrj@gmail.com

Discussion section: Friday 2-4pm Kaplan 169

Office hours: Monday 2-4pm Boelter 3256S-B

Overview for today

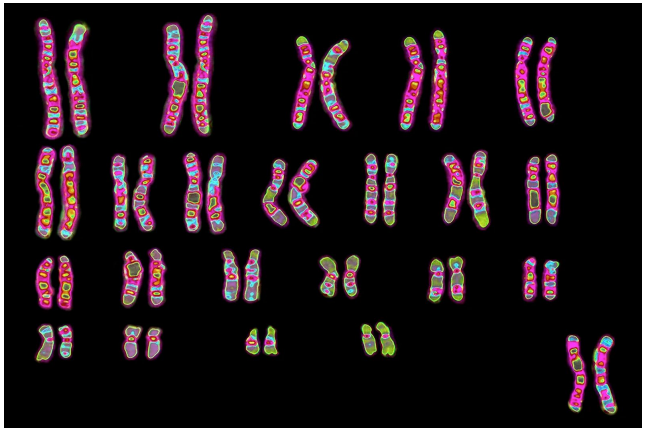
Part 1. Brief refresher on biology / computer science (covering part of chapter 1)

Part 2. Textbook chapter 1

Part 3. Answering questions from the lectures

Biology / CS refresher

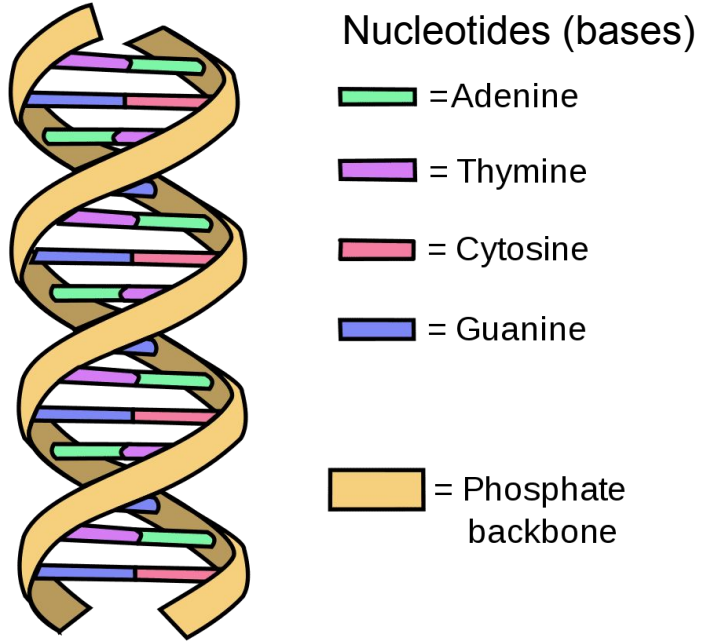
The human genome



https://www.britannica.com/science/human-genetics

The entire human genome consist of 23 pairs of chromosomes (one pair being either XX or XY) and (two copies of) 3 billion base pairs.

https://www.ashg.org/discover-genetics/building-blocks/

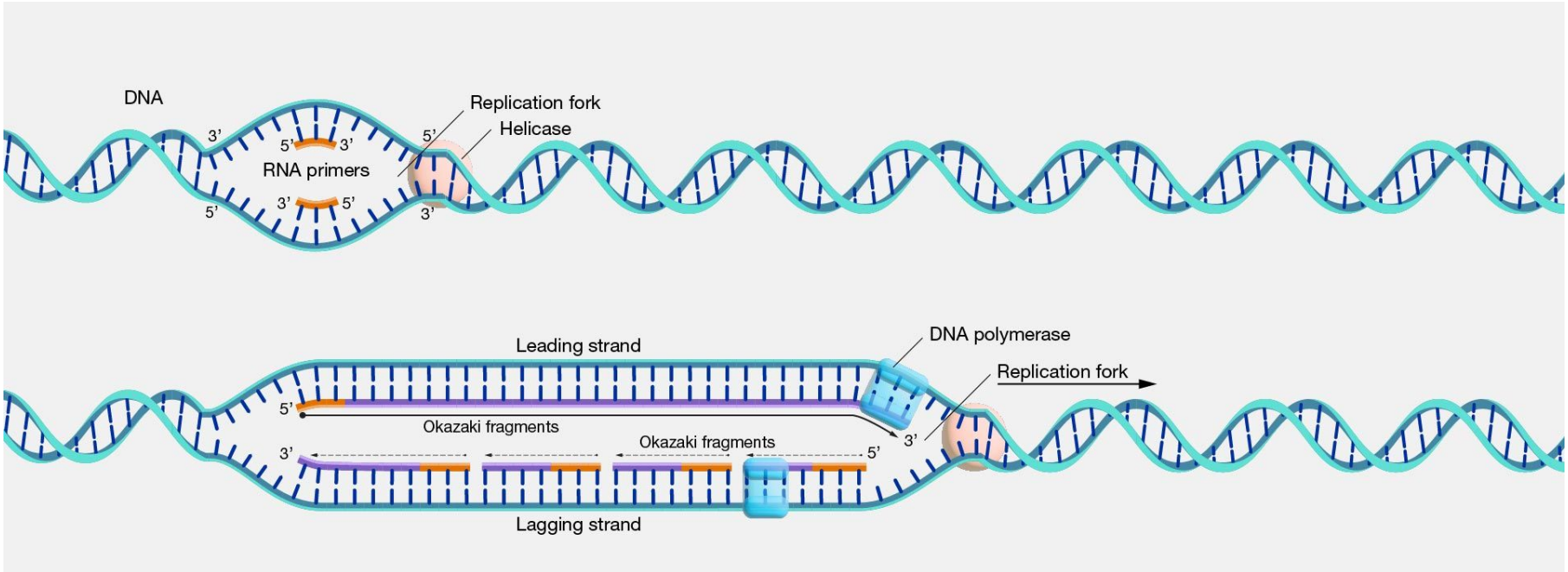


DNA

The DNA is double stranded. The two strands are reverse complements of each other, with As complementing Ts, and Gs complementing Cs. For example, one strand is “ATGG”, the other is “CCAT”

Directionality of DNA replication

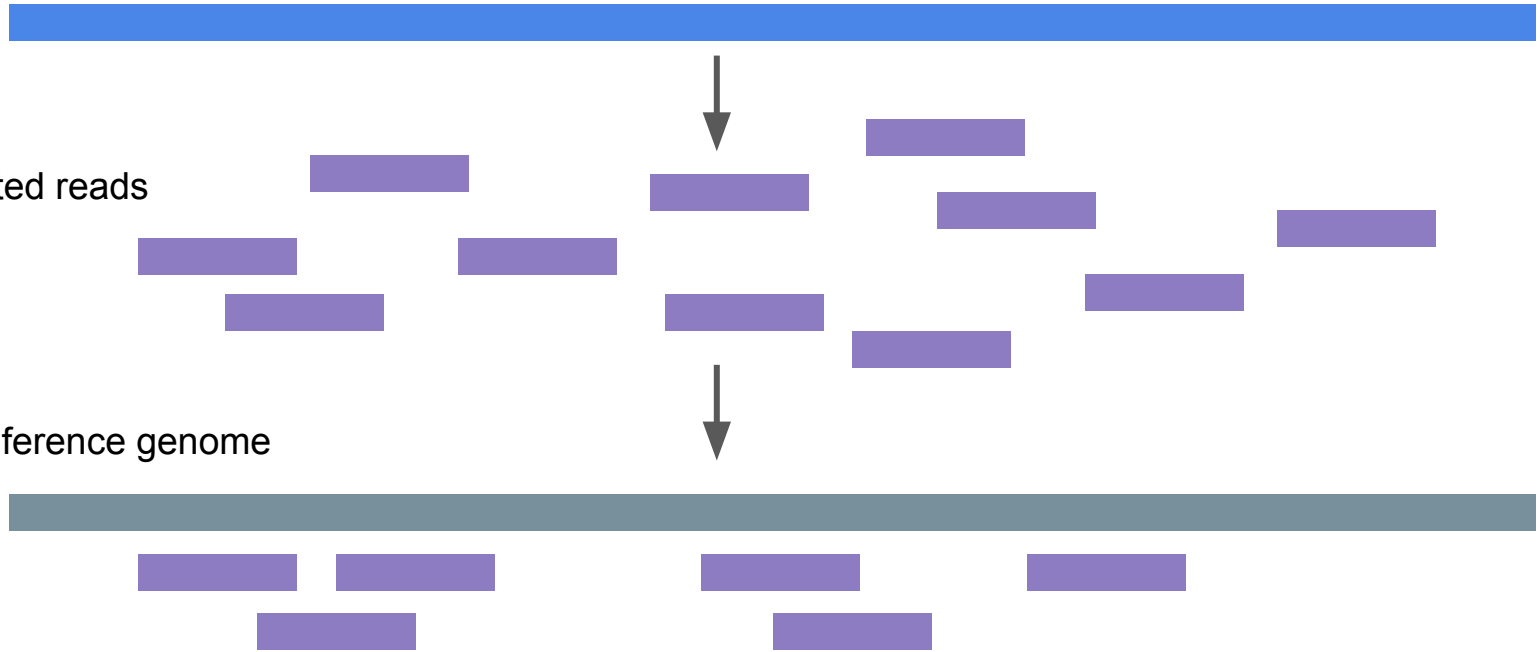
https://www.genome.gov/genetics-glossary/DNA-Replication



- Each DNA strand has a 5’ end and a 3’ end, DNA polymerase can only make a new strand from 5’ to 3’
- The leading strand is also called the reverse strand, The lagging strand is also called the forward strand

Modern genome sequencing

Sample’s genome

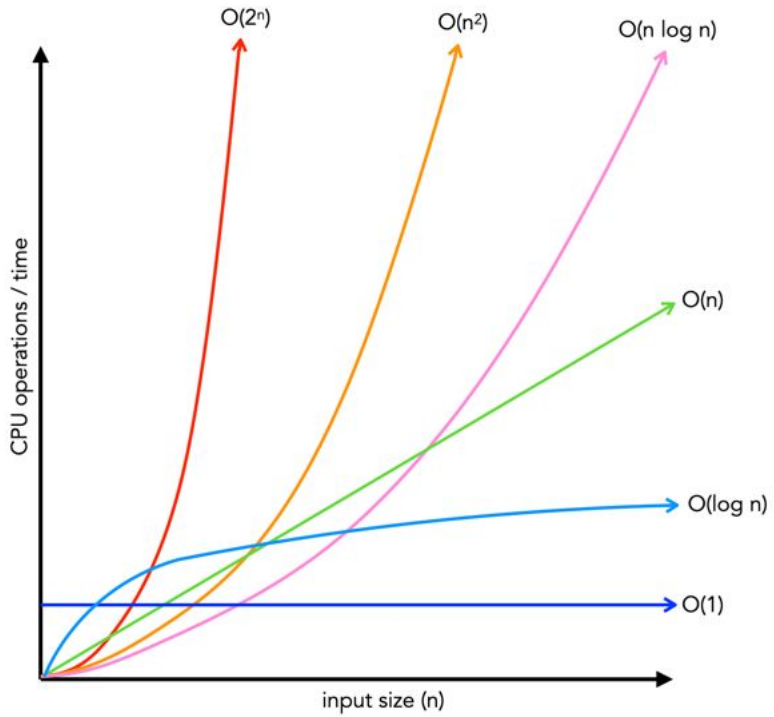


Fragmented reads

Map to reference genome

Computational complexity

craftofcoding.wordpress.com



The Big-O notation represents the running time of an algorithm.

For example, if your algorithm for sorting an array of n numbers takes roughly n^2 operations for the most difficult dataset, then we say that the running time of your algorithm is $O(n^2)$.

* Question: can you give an example of an algorithm belonging to each runtime category in the picture?

Recursion

You want to solve a problem of size N. You don't know how to do it, but:

- You already know how to solve a problem of size 1 (the base case)
- You already know how it relates to a problem of a smaller size, M

Example: computing the n^{th} number in a Fibonacci sequence (0, 1, 1, 2, 3, 5, … 8) where each number is the sum of the previou two

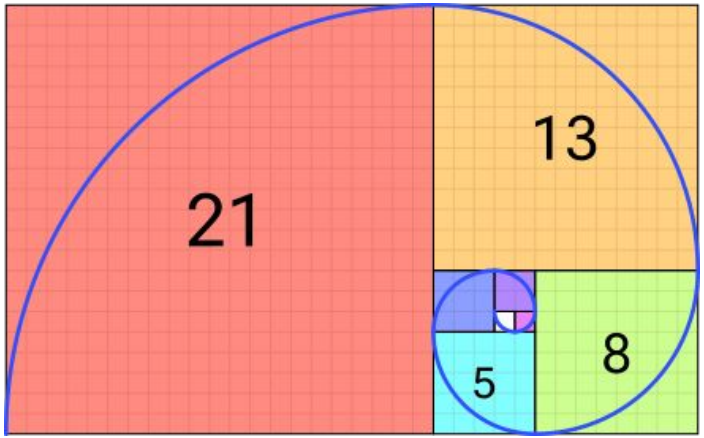
- You already know Fibonacci(0) = 0, Fibonacci(1) = 1
- You already know Fibonacci(n) = Fibonacci(n - 1) + Fibonacci(n - 2)

Question: how to write a function that computes the n^{th} Fibonacci number?

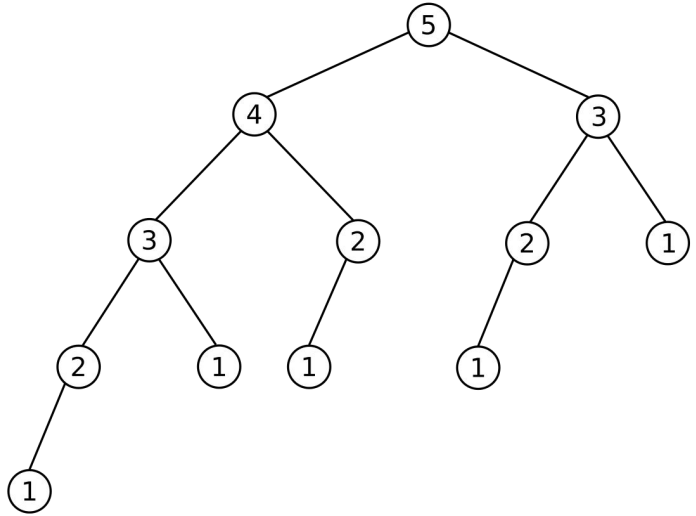
Recursion

developerinsider.co

```
int fibonacci(int num)
{
    if (num <= 1) return num;
    return fibonacci(num - 2) + fibonacci(num - 1);
}
```



The Fibonacci tree



The complexity is $O(2^n)$

Chapter 1

Counting k-mers of a DNA sequence

```
atcaatgatcaacgtaagcttctaagcATGATCAAGgtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagATGATCAAGagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaagaATGATCAAGctgctgctcttgatcatcgtttc
```

In this DNA sequence, the 9-mer “ATGATCAAG” appears 3 times.

Question: given a DNA sequence of length L and a specific k-mer, how to write a function f(sequence, k-mer) that counts its number of occurrences in the DNA sequence?

Counting all k-mers of a DNA sequence

Answer: f(sequence, kmer) -> slide a k-length window across the DNA sequence, advancing 1 base pair at a time. If the current window matches the k-mer, increment the count.

Question: given a DNA sequence of length L and a k, how to write a function that counts the occurrences of all k-mers in the DNA sequence? What data structure can be used?

Counting all k-mers of a DNA sequence

ACG	CGT	GTT	TTT	TTC	TCA	CAC	TTA	TAC	CGG
3	2	2	3	1	1	1	1	1	1

Figure: A table corresponding to counting the number of occurrences of every 3-mer in *Text* = `"ACGTTTCACGTTTTACGG"`.

Answer: Create an empty hash table used for storing the counts of each k-mer. Slide a k-length window across the DNA sequence, advancing 1 base pair at a time. In the hash table, increment the count of the current window.

Finding clumps in a DNA sequence

We defined a *k*-mer as a "clump" if it appears many times within a short interval of the genome. More formally, given integers *L* and *t*, a *k*-mer *Pattern* forms an **(*L*, *t*)-clump** inside a (longer) string *Genome* if there is an interval of *Genome* of length *L* in which this *k*-mer appears at least *t* times. (This definition assumes that the *k*-mer *completely* fits within the interval. This also does not take reverse complements into account yet.) For example, **TGCA** forms a (25,3)-clump in the following *Genome*:

gatcagcataagggtccC**TGCA**A**TGCA**TGACAAGCC**TGCA**GTtgttttac

From our previous examples of *ori* regions, **ATGATCAAG** forms a (500,3)-clump in the *Vibrio cholerae* genome, and **CCTACCACC** forms a (500,3)-clump in the *Thermotoga petrophila* genome. We are now ready to formulate the following problem.

Clump Finding Problem: *Find patterns forming clumps in a string.*

- **Input**: A string *Genome*, and integers *k*, *L*, and *t*.
- **Output**: All distinct *k*-mers forming (*L*, *t*)-clumps in *Genome*.

Finding clumps in a DNA sequence

Clump Finding Problem: *Find patterns forming clumps in a string.*

- **Input**: A string *Genome*, and integers *k*, *L*, and *t*.
- **Output**: All distinct *k*-mers forming (*L*, *t*)-clumps in *Genome*.

The simplest way of doing so is to iterate over all possible L-sized windows in the genome. Create and fill a hash table for each window and select the k-mers that appear >= t times.

However, do you really need to create a separate hash table for each window?

Hint: what's the difference between two hash tables belonging to these two windows:

- Window 1: ATGCGCGTTTG
- Window 2: TGCGCGTTTGG

Hamming distance between sequences

A	T	G	A	C	C	G
A	A	T	A	C	G	G

The Hamming distance is 3

The Hamming distance measures the number of mismatches between corresponding positions in two sequences (assuming they have equal length)

Finding the d-neighborhood of a k-mer

The d-neighborhood of a k-mer Pattern is the set of all k-mers whose Hamming distance from Pattern does not exceed d.

For example, the 1-neighborhood of “AT” consists of seven 2-mers:

AT, AG, AA, AC, TT, GT, CT

Question: give the above seven k-mers how to generate the 1-neighborhood of “CAT”?

Finding the d-neighborhood of a k-mer, with d=1 and k-mer “CAT”

2-mer	Hamming distance to AT	d minus Hamming distance to AT	Possible 3-mer with Hamming distance at most 1 to CAT
AT	0	1 - 0 = 1	**G**AT, **A**AT, **T**AT, **C**AT
AG	1	0	**C**AG
AA	1	0	**C**AA
AC	1	0	**C**AC
TT	1	0	**C**TT
GT	1	0	**C**GT
CT	1	0	**C**CT

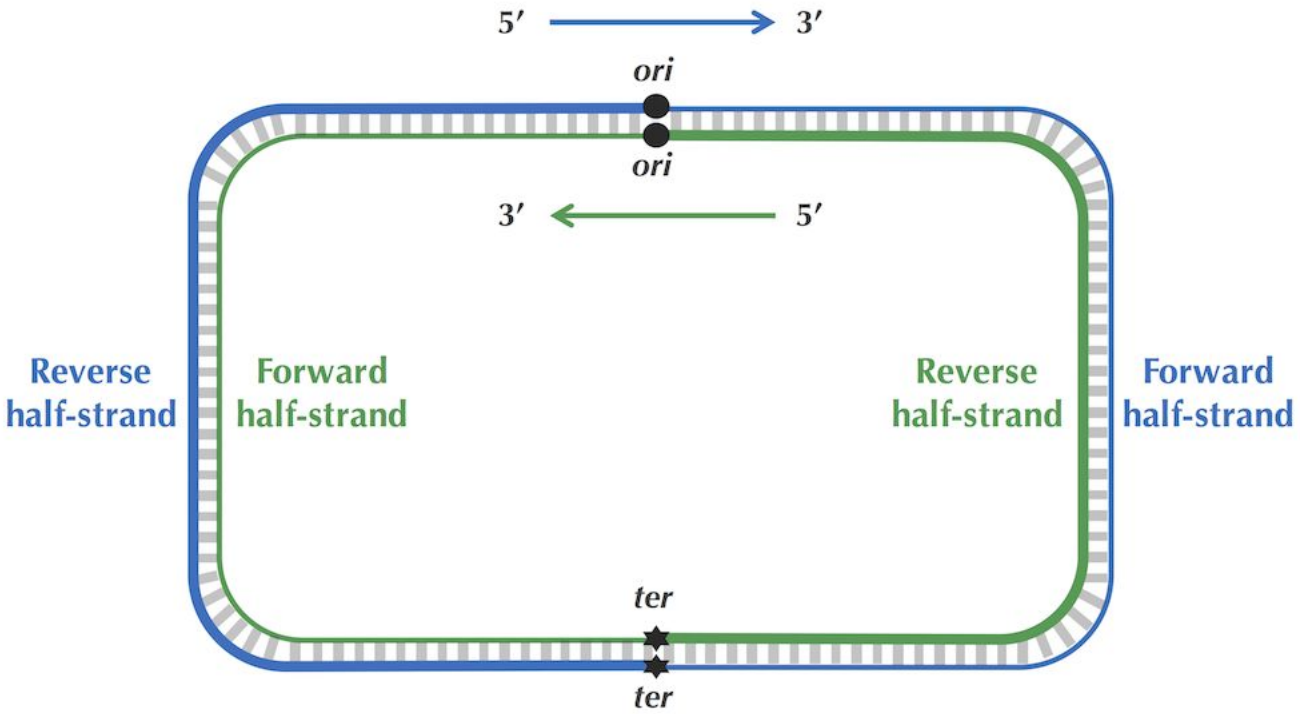
For each 2-mer in the 1-neighborhood of AT, compute its Hamming distance to AT.

- If the distance is already d, you can only add a “C” to the front

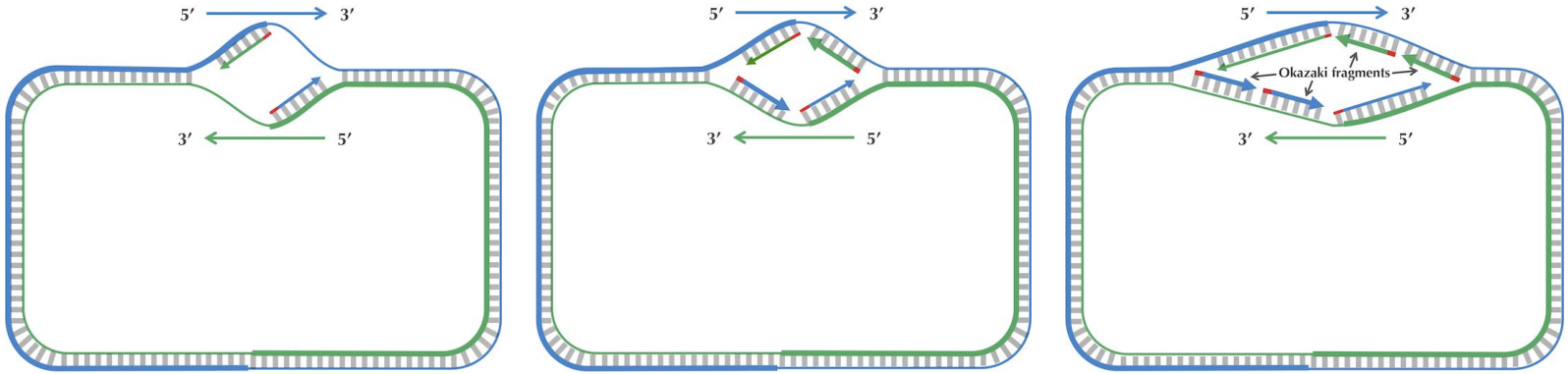
- If the distance is not yet d, you can add any one of A,T,G or C to the front

Conclusion: The d-neighborhood of a k-mer $S = (S_0, S_1, \ldots S_{k-1})$ can be computed if you already know the d-neighborhood of its suffix $(S_1, S_2 \ldots S_{k-1})$

Replication of bacterial DNA

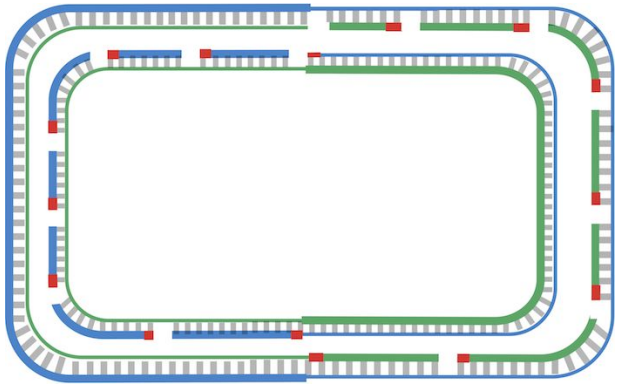


Replication of bacterial DNA

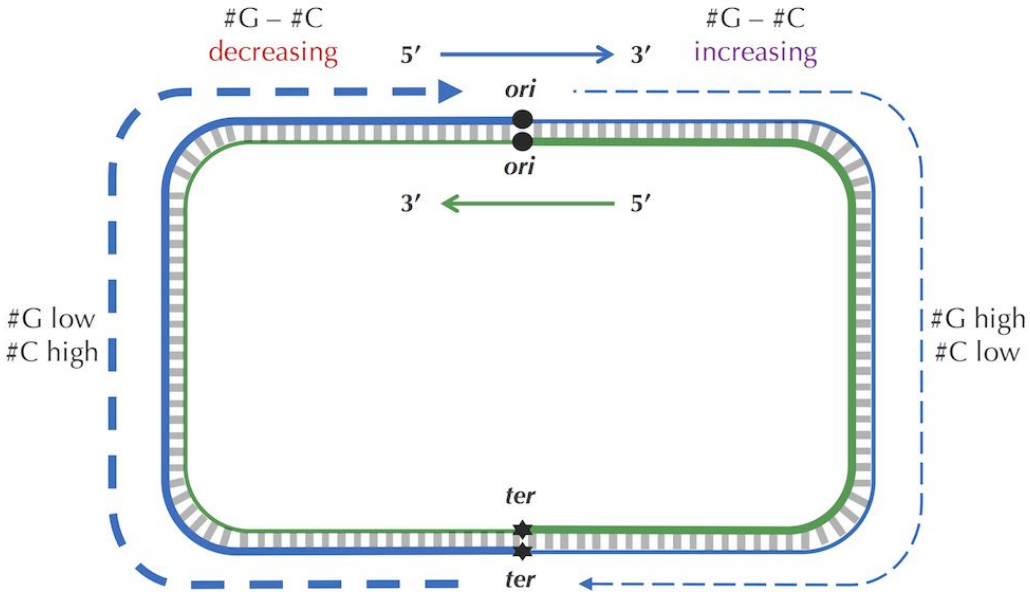


Reminder: DNA polymerase can only make a new strand from 5’ to 3’

Notice that all Okazaki fragments are on the forward half-strands (thin)

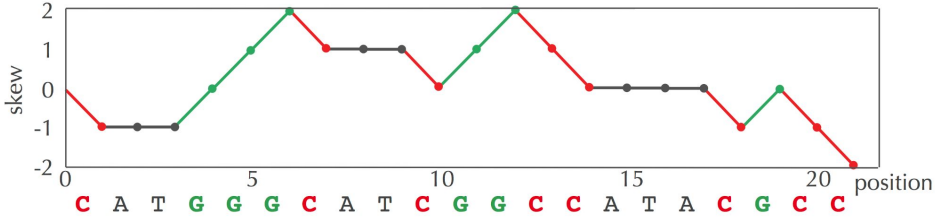


Replication of bacterial DNA

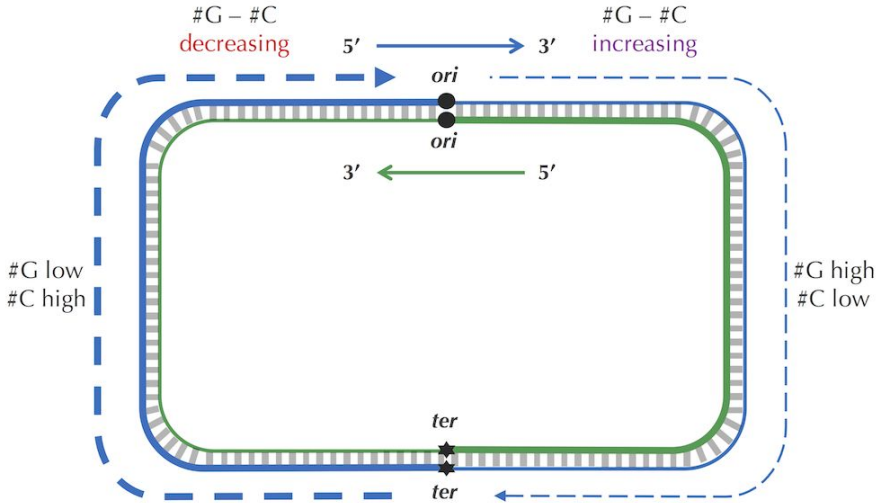


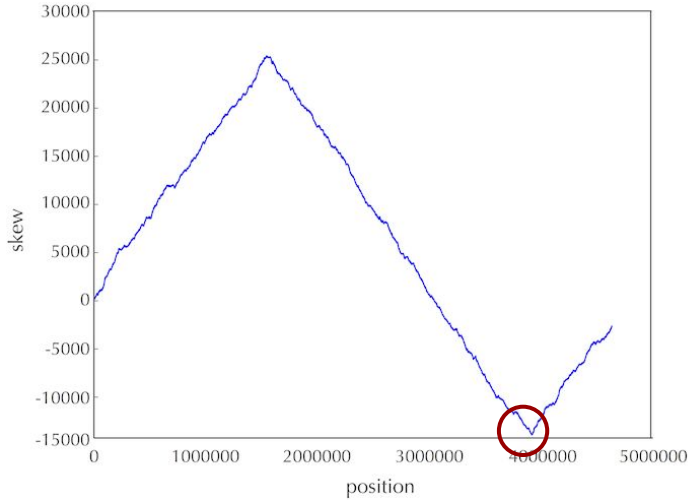
C has a high chance of mutating into T on the forward substrand.
- As you travel through the forward substrand (thin blue), the difference between counts of G and C should be increasing.
- As you travel through the reverse substrand (thick blue), the difference between counts of G and C should be decreasing.

Finding *ori* of bacterial DNA



"Skew" is the running difference between G and C counts





Questions from the lectures

Binomial distribution

You have an unfair coin. Every time you flip it, it has a p chance of landing on heads and (1-p) chance of landing on tails. Now you flip it N times, how many heads will you see?

Let the number of heads be k, k follows the binomial distribution:

$$P(k) = \binom{N}{k} p^k (1-p)^{N-k}$$

Probability of having exactly k heads

Probability of having k heads

"N choose k" or the number of different ways the k heads occur in the sequence of flips
(e.g. "1st 2nd and 5th flips", "3rd 4th and 9th flips", …)

Probability of having N-k tails

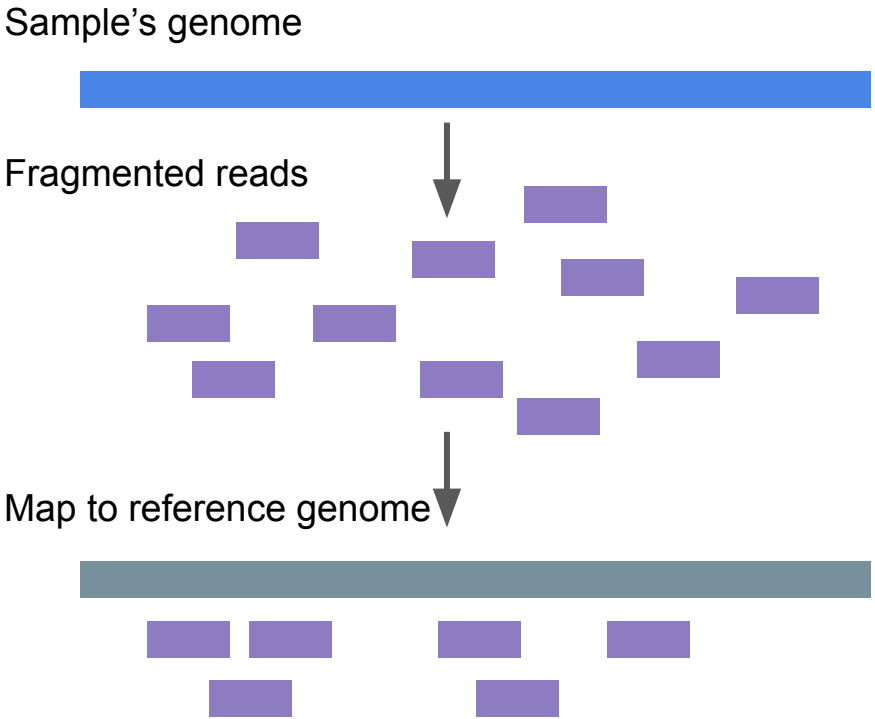
From binomial to Poisson distribution

The Poisson distribution is the limit of binomial distribution when the number of flips N is very large

$$\lim_{N\to\infty} P(k) = \frac{e^{-\lambda}\lambda^{k}}{k!}$$

Where lambda is the mean of the distribution that is N * p, the number of flips times the head probability

Sequencing read coverage follows Poisson distribution



If the length of the genome is N and there are K reads of length L. The overall coverage is KL/N.

Pick a random position in the genome. Each read has a L/N probability of overlapping this position.

Question: how does this relate to the coin flipping problem?

Sequencing read coverage follows Poisson distribution

Answer: This is the same as flipping a coin K times and each coin has L/N probability of landing on heads.

Since K (the number of flips) is now very large, the number of reads that overlap random position follows a Poisson distribution with mean K * (L/N) = KL/N that is exactly the overall coverage.

Burrows-Wheeler transform of “panamabananas$”

Cyclic Rotations	M("panamabananas$")
panamabananas$	$ p a n a m a b a n a n a **s**
$panamabananas	a b a n a n a s $ p a n a **m**
s$panamabanana	a m a b a n a n a s $ p a **n**
as$panamabanan	a n a m a b a n a n a s $ **p**
nas$panamabana	a n a n a s $ p a n a m a **b**
anas$panamaban	a n a s $ p a n a m a b a **n**
nanas$panamaba	a s $ p a n a m a b a n a **n**
ananas$panamab	b a n a n a s $ p a n a m **a**
bananas$panama	m a b a n a n a s $ p a n **a**
abananas$panam	n a m a b a n a n a s $ p **a**
mabananas$pana	n a n a s $ p a n a m a b **a**
amabananas$pan	n a s $ p a n a m a b a n **a**
namabananas$pa	p a n a m a b a n a n a s **$**
anamabananas$p	s $ p a n a m a b a n a n **a**

In the original text, the first letter in a row follows the last letter in the same row.

First-last property of a Burrows-Wheeler transform

$$pa_3na_2ma_1ba_4na_5na_6s\$$$

The i^{th} occurrence of a letter in the last column is exactly the same instance in the original text as the the i^{th} occurrence of the letter in the first column.

$	p	a	n	a	m	a	b	a	n	a	n	a	s
a_1	b	a	n	a	n	a	s	$	p	a	n	a	m
a_2	m	a	b	a	n	a	n	a	s	$	p	a	n
a_3	n	a	m	a	b	a	n	a	n	a	s	$	p
a_4	n	a	n	a	s	$	p	a	n	a	m	a	b
a_5	n	a	s	$	p	a	n	a	m	a	b	a	n
a_6	s	$	p	a	n	a	m	a	b	a	n	a	n
b	a	n	a	n	a	s	$	p	a	n	a	m	a_1
m	a	b	a	n	a	n	a	s	$	p	a	n	a_2
n	a	m	a	b	a	n	a	n	a	s	$	p	a_3
n	a	n	a	s	$	p	a	n	a	m	a	b	a_4
n	a	s	$	p	a	n	a	m	a	b	a	n	a_5
p	a	n	a	m	a	b	a	n	a	n	a	s	$
s	$	p	a	n	a	m	a	b	a	n	a	n	a_6