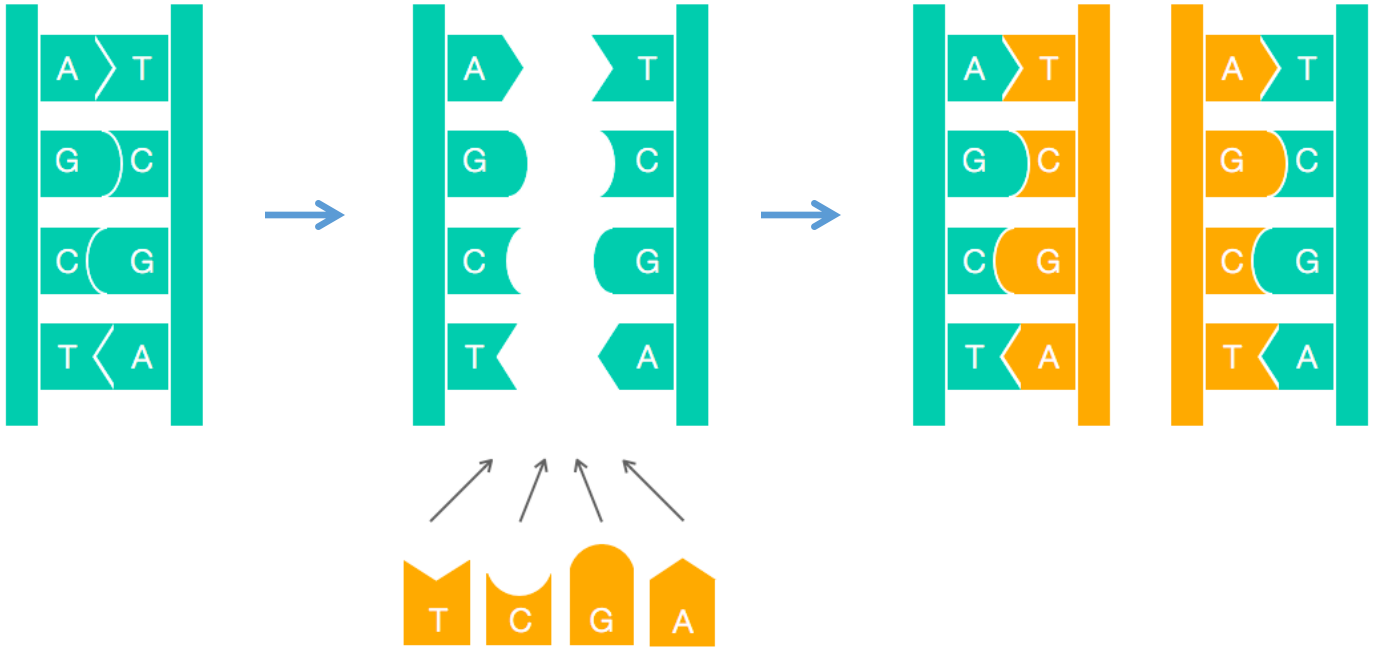
Where in a Genome Does DNA Replication Begin?

Algorithmic Warm-Up

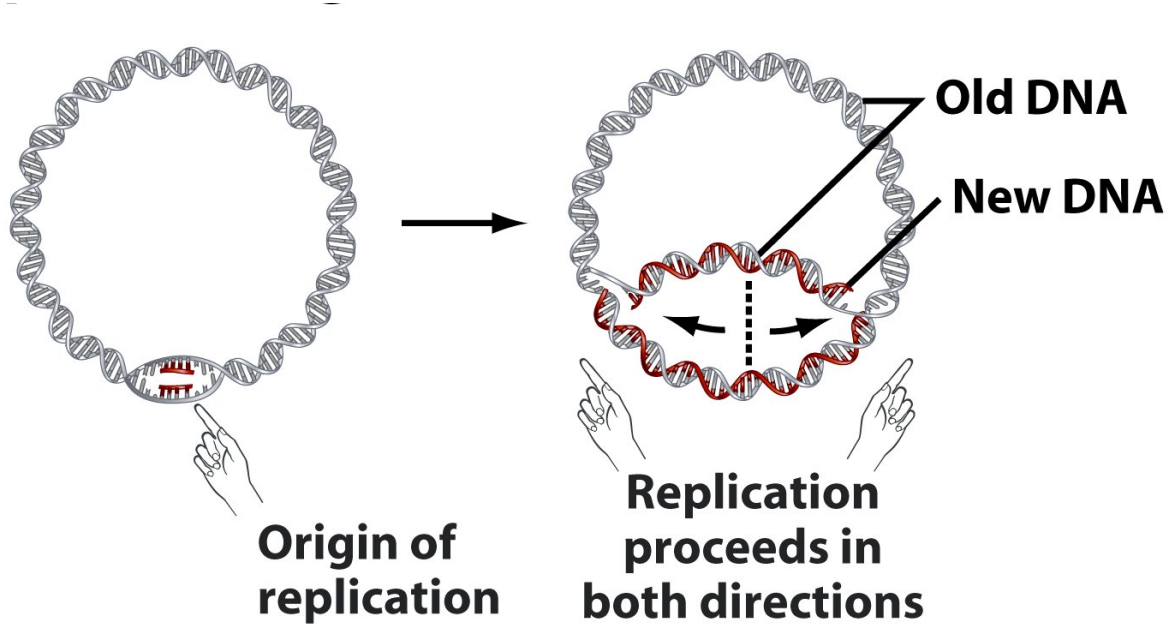
Phillip Compeau and Pavel Pevzner

Bioinformatics Algorithms: an Active Learning Approach

Before a Cell Divides, it Must Replicate its Genome



Replication begins in a region called the **replication origin** (*oriC*)



Where in a genome does it all begin?

Finding Origin of Replication

Finding *oriC* Problem: Finding *oriC* in a genome.

- **Input.** A genome.
- **Output.** The location of *oriC* in the genome.

OK – let's cut out this DNA fragment.
Can the genome replicate without it?

This is not a computational problem!

How Does the Cell Know to Begin Replication in Short *oriC*?

Replication origin of *Vibrio cholerae* (≈500 nucleotides):

```
atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaagatgatcaagagaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaagaatgatcaagctgctgctcttgatcatcgtttc
```

There must be a **hidden message** telling the cell to start replication here.

The Hidden Message Problem

Hidden Message Problem. Finding a hidden message in a string.

- **Input.** A string *Text* (representing replication origin).
- **Output.** A hidden message in *Text*.

The notion of "**hidden message**" is not precisely defined.

The Hidden Message Problem Revisited

Hidden Message Problem. Finding a hidden message in a string.

- **Input.** A string *Text* (representing *oriC*).
- **Output.** A hidden message in *Text*.

The notion of "**hidden message**" is not precisely defined.

Hint: For various biological signals, certain words appear surprisingly frequently in small regions of the genome.

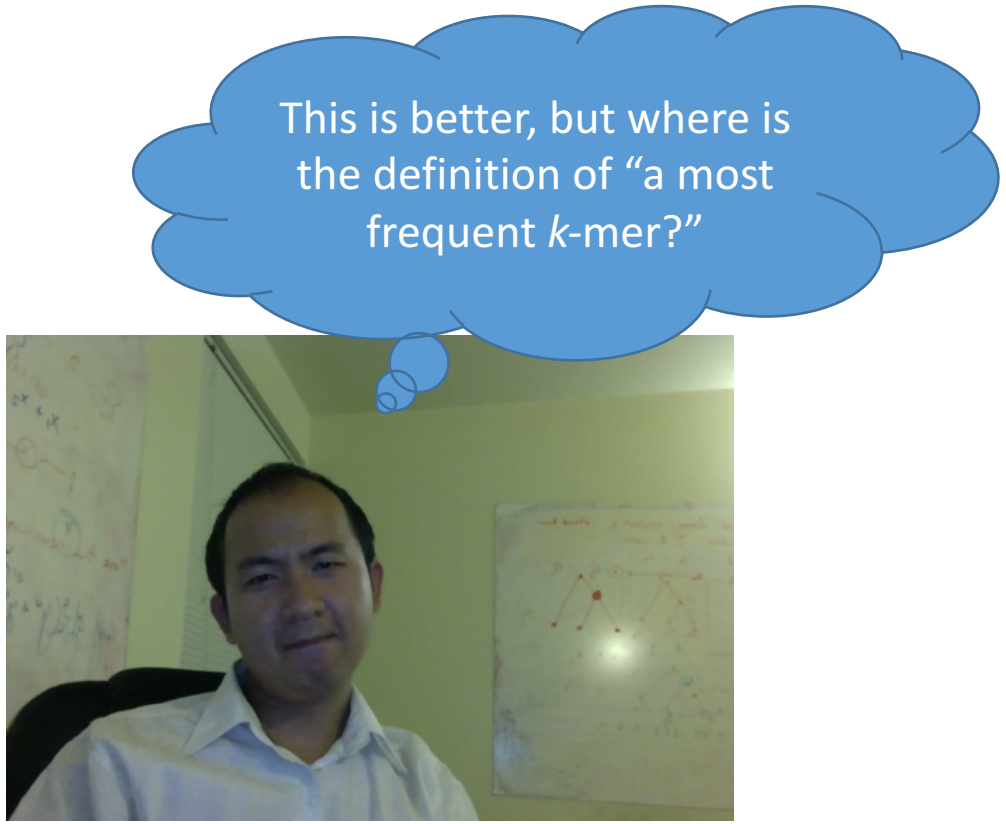
AATTT is a surprisingly frequent 5-mer in:

ACAAATTTGCATAATTTCGGGAAATTTCCT

The Frequent Words Problem

Frequent Words Problem. Finding most frequent *k*-mers in a string.

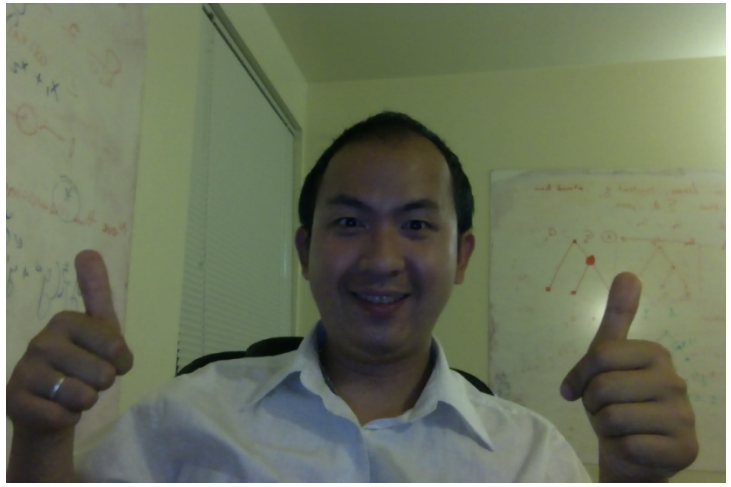
- **Input.** A string *Text* and an integer *k*.
- **Output.** All **most frequent *k*-mers** in *Text*.



The Frequent Words Problem

Frequent Words Problem. Finding most frequent *k*-mers in a string.

- **Input.** A string *Text* and an integer *k*.
- **Output.** All **most frequent *k*-mers** in *Text*.



Son Pham, Ph.D., kindly gave us permission to use his photographs and greatly helped with preparing this presentation. **Thank you Son!**

A *k*-mer ***Pattern*** is a **most frequent *k*-mer** in a text if no other *k*-mer is more frequent than *Pattern.*

AATTT is a most frequent 5-mer in:

ACA**AATTT**GCAT**AATTT**CGGGA**AATTT**CCT

Does the Frequent Words Problem Make Sense to Biologists?

Frequent Words Problem. Finding most frequent *k*-mers in a string.

- **Input.** A string *Text* and an integer *k*.
- **Output.** All **most frequent *k*-mers** in *Text*.

Replication is performed by **DNA polymerase** and the initiation of replication is mediated by a protein called ***DnaA***.

DnaA binds to short (typically 9 nucleotides long) segments within the replication origin known as a ***DnaA* box**.

A *DnaA* box is a hidden message telling *DnaA*: "**bind here!**" And *DnaA* wants to see multiple *DnaA* boxes.

What is the simplest way to get most frequent k-mers?

```
FREQUENTWORDS (Text, k)
        FrequentPatterns <- an empty set
        for i <-0  to |Text| - k
                Pattern <- the k-mer (i, k)
                COUNT(i)<- PATTERNCOUNT(Text, Pattern)
        maxCount <- maximum value in array COUNT
         for i <-0  to |Text| - k
                if COUNT(i) = maxCount
                        add Text(i, k) to FrequentPatterns
        remove duplicates from FrequentPatterns

PATTERNCOUNT(Text, Pattern)
        count <- 0
        for i <-0  to |Text| - |Pattern|
                if Text(i,| Pattern|) = Pattern
                        count<- count + 1
        return count
```

What is the problem with the previous algorithm ?

Human Genome is about 3 billion base pairs
$O(|text|^2 . k)$ will take forever!

How can we make FREQUENTWORDS faster?

What are the possible k-mers of length k = 3 in Alphabet A, T, C, G?

AAA
AAT
AAC
AAG
ATA
ATT
ATC
ATG
ACA
ACT
ACC
ACG
AGA
AGT
AGC
AGG…..

Number of possible combinations at k=3

$4^3 = 64$

Generally Number of possible combinations is 4^k

```
FASTERFREQUENTWORDS (Text, k)
        FrequentPatterns <- an empty set
        FREQUENCYARRAY <- COMPUTINGFREQENCIES (Text, k)
        maxCount <- maximum value in array FREQUENCYARRAY
        for i <-0  to 4^k -1
                if FREQUENCYARRAY (i) = maxCount
                        Pattern <– NumberToPattern (i, k)
                        add Pattern to FrequentPatterns
        remove duplicates from FrequentPatterns

COMPUTINGFREQENCIES (Text, k)
        for i <-0  to 4^k -1
                FREQUENCYARRAY (i) <- 0
        for i <-0  to |Text| - k
                Pattern <- Text(i, k)
                j <- PatternToNumber(Pattern)
                 FREQUENCYARRAY (j) <- FREQUENCYARRAY (j) + 1
        return FREQUENCYARRAY
```

Another idea!

Sort all k-mers and then count there frequency.

Will this improve complexity?

```
FINDINGFREQUENTWORDSBYSORTING (Text, k)
        FrequentPatterns <- an empty set
        for i <-0  to |Text| - k
                Pattern <- Text(i, k)
                INDEX(i) <-PatternToNumber(Pattern)
                COUNT(i) <- 1
        SORTEDINDEX <- SORT(INDEX)
        for i <-1  to |Text| - k
                if SORTEDINDEX (i)  = SORTEDINDEX (i-1)
                        COUNT (i) = COUNT (i-1) + 1
        maxCount <- maximum value in array COUNT
        for i <-1  to |Text| - k
                if COUNT (i) = maxCount
                        Pattern <– NumberToPattern (SORTEDINDEX (i), k)
                        add Pattern to FrequentPatterns
        remove duplicates from FrequentPatterns
```

How do we know that the frequencies are meaningful and not random?

Probabilities!

What is the probability of generating a palindromic (e.g., ATCGAAGCTA) ?

What is the probability that k-mer k=2 appears at least once in a binary string of length 4?

Say we want probability of 01

0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111

Probability is $\frac{11}{16}$

We made an assumption that text is not overlapping what if the pattern is AAAAAAAA?

What is the probability that k-mer k=2 appears at least once in a binary string of length 4?

Say we want probability of 11

0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111

Probability is $\frac{8}{16}$

What is the probability that *some* k-mer appears t times in a text?

Lets define some variables:

- $\Pr(N, A, Pattern, t)$: Probability that k-mer $Pattern$ appears t times in a text with length N and alphabet A.
- Let n be number of ways to intersect t instances of k-mer $Pattern$ into a fixed text of length N

  $$n = N - t.k$$

- So we have $n + t$ options in which we select t for the placement of $Pattern$ giving total $\binom{n+t}{t}$

What is the probability that *some* k-mer appears t times in a text? cont ..

- We then multiply $\binom{n+t}{t}$ by the number of strings of length n in which we can insert t instances of $Pattern$ to have approximate total of $\binom{n+t}{t} A^n$
- To get the probability we divide by the number of strings of length N

$$\Pr(N, A, Pattern, t) \approx \frac{\binom{n+t}{t} A^n}{A^N}$$

What is the probability of generating a palindromic (e.g., ATCGAAGCTA) in a DNA of length 1000 once?

$\Pr(1000, 4, ATCGAAGCTA, 1)$

What if the DNA has length 1×10^6 ?

$\Pr(1\times10^6, 4, ATCGAAGCTA, 1)$

What is the probability that *any* k-mer of length k appears **at least** t times in a text?

- Let $p = \Pr(N, A, Pattern, t) \approx \frac{\binom{n+t}{t} A^n}{A^N}$
- The approximate probability that a pattern doesn't appear t or more times is $1 - p$
- The probability that all patterns of length k appear **fewer** then t times in a random string is $(1-p)^{A^k}$
- The probability that there exists a k-mer appearing t or more times is $1 - (1-p)^{A^k}$
- To simplify the above equation lets assume p is the same for any pattern so now $\Pr(N, A, k\ t) \approx p.A^k \approx \frac{\binom{n+t}{t} A^n}{A^N} . A^k$