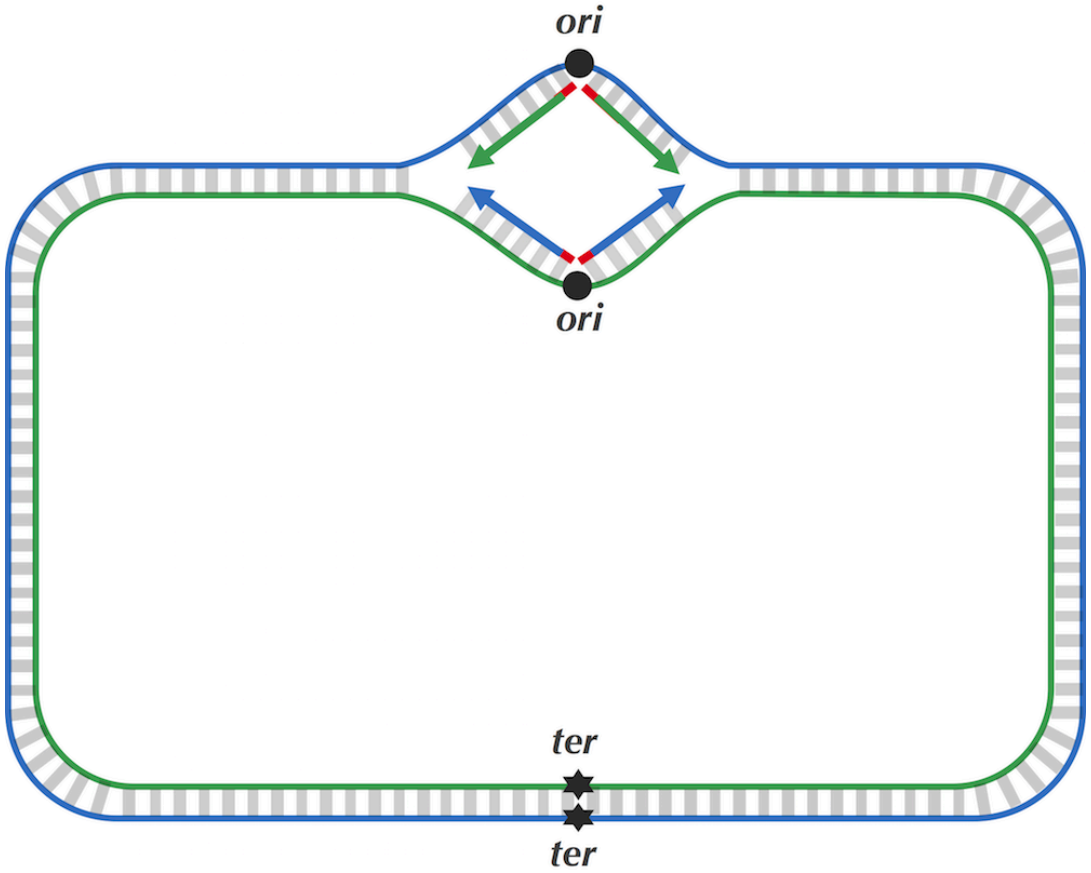
CMSC 423:
Finding Biological Signals

Part 3

Problem: Finding Hidden Messages in the Replication of Origin



- **Input**:

A string *Text* (representing the replication origin of a genome).

- **Output**:

A hidden message in *Text*.

DnaA: protein that binds to a short segment within the *ori* to begin replication

***DnaA* box**: where *DnaA* binds, the "hidden" message within the *ori*

Problem: Finding the Most Frequent Words

```
FrequentWords(Text, k)
    FrequentPatterns ← an empty set
    Count ← an array of length |Text|-k+1
    for i ← 0 to |Text| - k
        Pattern ← Text(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
    return FrequentPatterns
```

How efficient is FrequentWords?

Big-O notation: describes the running time of an algorithm (measures the worst-case running time)

```
FrequentWords(Text, k)
    FrequentPatterns ← an empty set
    Count ← an array of length |Text|-k+1
    for i ← 0 to |Text| - k
        Pattern ← Text(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
    return FrequentPatterns
```

Called **|Text|- k + 1** times (once for each k-mer)

Requires us to check **|Text|- k + 1** times for each k-mer, and requires as many as **k** comparsions

$$O((|Text| - k + 1) * k * (|Text| - k + 1)) = O(|Text|^2 * k)$$

Why is this algorithm so slow?

- Sliding a window of length *k* through *Text* for each step
- We want to slide a window down *Text* just one time!

Converting patterns to numbers

```
A = 0 = 00
C = 1 = 01
G = 2 = 10
T = 3 = 11
```

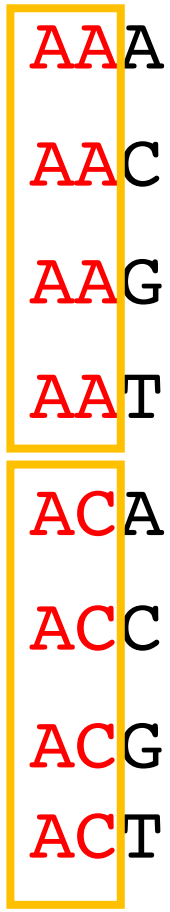
```
A C C A

00010100
```

Why is this algorithm so slow?

- Sliding a window of length *k* through *Text* for each step
- We want to slide a window down *Text* just one time!
- Order all 4^k *k*-mers lexicographically
- Convert the *k*-mers into integers between 0 and $4^k - 1$

Converting patterns to numbers

0	AAA
1	AAC
2	AAG
3	AAT
4	ACA
5	ACC
6	ACG
7	ACT



Every (*k*-1)-mer occurs 4 times

Converting patterns to numbers

= 4 * the number of (*k*-1)-mers before the prefix *
the number of 1-mers before the last symbol

Converting patterns to numbers

- Convert ACG to an integer

- Number of 3-mers before ACG =

  4 * Number of 2-mers before AC +

  the number of 1-mers before G

  = 4 * 1 + 2 = 6

0	AA
1	**AC**
2	AG
3	AT
4	CA
5	CC
6	CG
7	CT
8	GA
9	GC
10	GG
11	GT
12	TA
13	TC
14	TG
15	TT

0	A
1	C
2	**G**
3	T

0	AAA
1	AAC
2	AAG
3	AAT
4	ACA
5	ACC
6	**ACG**
7	ACT

and Think

Convert the *k*-mer CTC to a number.

and Think

0	AA
1	AC
2	AG
3	AT
4	CA
5	CC
6	CG
7	**CT**
8	GA
9	GC
10	GG
11	GT
12	TA
13	TC
14	TG
15	TT

Convert the *k*-mer CTC to a number.

0	A
1	**C**
2	G
3	T

= 4 * the number of 2-mers before CT *
the number of 1-mers before C

= 4 * 7 + 1

= 29

Converting numbers back to patterns?

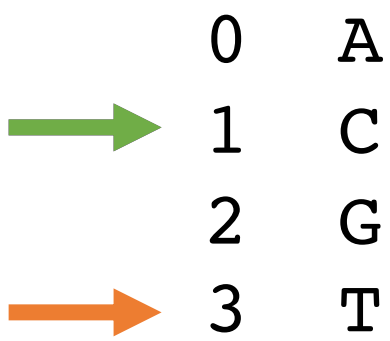
= NUMBER / 4

Prefix of the pattern = the quotient

Last symbol = the remainder

Keep going until you recover all *k* nucleotides

Converting patterns to numbers



Convert 29 back to a 3-mer DNA sequence

29/4 = 7 remainder 1

7/4 = 1 remainder 3

1/4 = 0 remainder 1

C T C

and Think

Convert the number 9904 back to an 8-mer.

and Think

0	A
1	C
2	G
3	T

Convert the the number 9904 back to an 8-mer.

9904/4 = 2476 remainder 0 = A
2476/4 = 619 remainder 0 = A
619/4 = 154 remainder 3 = T
154/4 = 38 reminder 2 = G
38/4 = 9 remainder 2 = G
9/4 = 2 remainder 1 = C
2/4 = 0 remainder 2 = G
0/4 = 0 remainder 0 = A

AGCGGTAA

Speeding up FrequentWords

```
ComputingFrequencies(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) + 1
return FrequencyArray
```

FindAllPatterns(ACTGACTC, 2)

k-mer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Count	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FindAllPatterns(ACTGACTC, 2)

k-mer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Count	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ACTGACTC

AC = 4 * 0 + 1 = 1

FindAllPatterns(ACTGACTC, 2)

k-mer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Count	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0

ACTGACTC

CT = 4 * 1 + 3 = 7

FindAllPatterns(ACTGACTC, 2)

k-mer	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Count	0	2	0	0	0	0	0	2	1	0	0	0	0	1	1	0

Improvement by sorting

```
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)
```

Improvement by sorting

Text = AAGCAAAGGTGGG, *k* = 2

k-mer	AA	AG	GC	CA	AA	AA	AG	GG	GT	TG	GG	GG
Index	0	2	9	4	0	0	2	10	11	14	10	10
Count	1	1	1	1	1	1	1	1	1	1	1	1

Improvement by sorting

Text = AAGCAAAGGTGGG, *k* = 2

k-mer	AA	AA	AA	AG	AG	CA	GC	GG	GG	GG	GT	TG
Sorted Index	0	0	0	2	2	4	9	10	10	10	11	14
Count	1	1	1	1	1	1	1	1	1	1	1	1

Improvement by sorting

```
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
```

Improvement by sorting

Text = AAGCAAAGGTGGG, *k* = 2

k-mer	AA	AA	AA	AG	AG	CA	GC	GG	GG	GG	GT	TG
Sorted Index	0	0	0	2	2	4	9	10	10	10	11	14
Count	1	2	3	1	2	1	1	1	2	3	1	1

Improvement by sorting

```
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 the array count
        for i <- 0 to |Text| - k
                if Count(i) = maxCount
                        Pattern <- NumberToPattern(SortedIndex(i), k)
                        add Pattern to the set FrequentPatterns
        return FrequentPatterns
```

Improvement by sorting

Text = AAGCAAAGGTGGG, *k* = 2

maxCount= 3

k-mer	AA	AA	AA	AG	AG	CA	GC	GG	GG	GG	GT	TG
Sorted Index	0	0	0	2	2	4	9	10	10	10	11	14
Count	1	2	3	1	2	1	1	1	2	3	1	1

Summary

- FrequentWords is slow --- we want to only search our *Text* once!
- Can encode nucleotides (and *k*-mers) as numbers and use frequency arrays to speed up the process
- Can further improve this with sorting