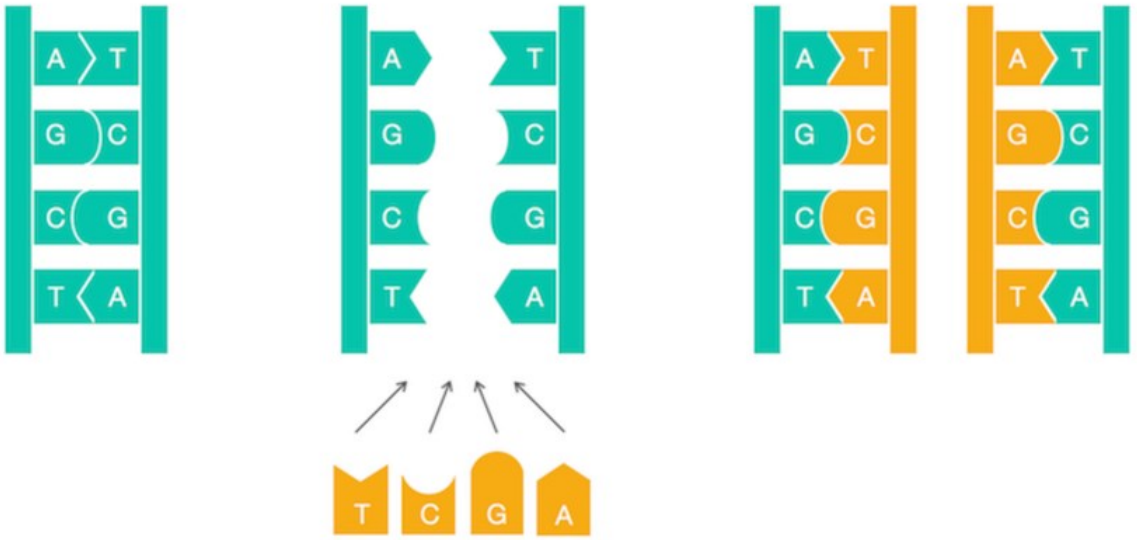
Hidden Messages in DNA*

* Some of the contents are adopted from
MATLAB® for Engineers, Holly Moore, 3rd Ed., Pearson Inc., 2012.

Genome Replication

- **Genome replication** is one of the most important tasks carried out in the cell. Before a cell can divide, it must first replicate its genome so that each of the two daughter cells inherits its own copy.
- **James Watson and Francis Crick** conjectured that the two strands of the parent DNA molecule unwind during replication, and then each parent strand acts as a template for the synthesis of a new strand. As a result, the replication process begins with a pair of complementary strands of DNA and ends with two pairs of complementary strands, as shown below.



A naive view of replication. Nucleotides adenine (A) and thymine (T) are complements of each other, as are cytosine (C) and guanine (G). Complementary nucleotides bind to each other in DNA.

Gene Threaphy

- Replication begins in a genomic region called the **replication origin** (denoted by *oriC*) and is carried out by molecular copy machines called **DNA polymerases**.
- Locating *oriC* presents an important task not only for understanding how cells replicate but also for various biomedical problems.
- For example, some **gene therapy** methods use genetically engineered mini-genomes, which are called **viral vectors** because they are able to penetrate cell walls (just like real viruses). Viral vectors carrying artificial genes have been used in agriculture, such as to engineer frost-resistant tomatoes and pesticide-resistant corn. In 1990, gene therapy was first successfully performed on humans when it saved the life of a four-year-old girl suffering from Severe Combined Immunodeficiency Disorder; the girl had been so vulnerable to infections that she was forced to live in a sterile environment.
- The idea of gene therapy is to intentionally infect a patient who lacks a crucial gene with a viral vector containing an artificial gene that encodes a therapeutic protein. Once inside the cell, the vector replicates and eventually produces many copies of the therapeutic protein, which in turn treats the patient's disease. To ensure that the vector actually replicates inside the cell, biologists must know where *oriC* is in the vector's genome and ensure that the genetic manipulations that they perform do not affect it.

Finding *oriC*

- Biologists would immediately plan an experiment to locate *oriC*: for example, they might delete various short segments from the genome in an effort to find a segment whose deletion stops replication.
- **Computational methods** are now the only realistic way to answer many questions in modern biology.
- First, these methods are much faster than experimental approaches; second, the results of many experiments cannot be interpreted without computational analysis.
- In particular, existing experimental approaches to *oriC* prediction are rather time consuming. As a result, *oriC* has only been experimentally located in a handful of species. Thus, we would like to design a computational approach to find *oriC* so that biologists are free to spend their time and money on other tasks.

DnaA boxes

- In this part, we will focus on the relatively easy case of finding *oriC* in bacterial genomes, most of which consist of a single circular chromosome.
- Research has shown that the region of the bacterial genome encoding *oriC* is typically a few hundred nucleotides long. Our plan is to begin with a bacterium in which *oriC* is known, and then determine what makes this genomic region special in order to design a computational approach for finding *oriC* in other bacteria.
- Our example is *Vibrio cholerae*, the pathogenic bacterium that causes cholera; here is the nucleotide sequence appearing in the *oriC* of*Vibrio cholerae*:

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

DnaA boxes

- How does the bacterial cell know to begin replication exactly in this short region within the much larger*Vibrio cholerae* chromosome, which consists of 1,108,250 nucleotides?
- There must be some "hidden message" in the *oriC* region ordering the cell to begin replication here. Indeed, we know that the initiation of replication is mediated by ***DnaA***, a protein that binds to a short segment within the *oriC* known as a ***DnaA* box**.
- You can think of the *DnaA* box as a message within the DNA sequence telling the *DnaA* protein: "bind here!" The question is how to find this hidden message without knowing what it looks like in advance—can you find it? In other words, can you find something that stands out in *oriC*? This discussion motivates the following problem.

Hidden Message Problem: *Find a "hidden message" in the replication origin.*
Input: A string *Text* (representing the replication origin of a genome).
Output: A hidden message in *Text*.

Hidden messages in "The Gold-Bug"

- Although the Hidden Message Problem poses a legitimate intuitive question, it again makes absolutely no sense to a computer scientist because the notion of a “hidden message” is not precisely defined.
- The *oriC* region of *Vibrio cholerae* is currently just as puzzling as the parchment discovered by William Legrand in Edgar Allan Poe's story "The Gold-Bug". Written on the parchment was

```
53‡‡†305))6·;4826)4‡.)4‡);806·;48†8^60))85;161;:‡·8
†83(88)5·†;46(;88·96·?;8)·‡(;485);5·†2:·‡(;4956·2(5
·—4)8^8·;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4
)485†528806·81(‡9;48;(88;4(‡?34;48)4‡;1‡(;:188;‡?;
```

Hidden messages in "The Gold-Bug"

- We see that the three consecutive symbols **;48** appear with surprising frequency on the parchment.

53‡‡†305))6·**;48**26)4‡.)4‡);806·**;48**†8^60))85;161;:‡·8
†83(88)5·†;46(;88·96·?;8)·‡(**;48**5);5·†2:·‡(;4956·2(5
·—4)8^8·;4069285);)6†8)4‡‡;1(‡9**;48**081;8:8‡1**;48**†85;4
)485†528806·81(‡9**;48**;(88;4(‡?34**;48**)4‡;1‡(;:188;‡?;

- The pirates spoke English; therefore we can assume that the high frequency of **;48** can encode the most frequent English word, **THE**.
- Substituting **;** for **T**, **4** for**H**, and **8** for **E**, we had a slightly easier text to decipher.

53‡‡†305))6·**THE**26)**H**‡.)**H**‡)**TE**06·**THE**†**E**^60))**E**5**T**161**T**:‡·**E**
†**E**3(**EE**)5·†**TH**6(**TEE**·96·?**TE**)·‡(**THE**5)**T**5·†2:·‡(**TH**956·2(5
·—**H**)**E**^**E**·**TH**0692**E**5)**T**)6†**E**)**H**‡‡**T**1(‡9**THE**0**E**1**TE**:**E**‡1**THE**†**E**5**TH**
)**HE**5†52**EE**06·**E**1(‡9**THET**(**EETH**(‡?3**HTHE**)**H**‡**T**1‡(**T**:1**EET**‡?**T**

Counting Words

- Operating under the assumption that DNA is a language of its own, let's borrow this method and see if we can find any surprisingly frequent "words" within the *oriC* of *Vibrio cholerae*.
- We have added reason to look for frequent words in the *oriC* because for various biological processes, certain nucleotide strings often appear surprisingly often in small regions of the genome. For example, **ACTAT** is a surprisingly frequent substring of ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT.
- We will use the term ***k*-mer** to refer to a string of length *k* and define *Count*(*Text*, *Pattern*) as the number of times that a *k*-mer *Pattern* appears as a substring of *Text*. Following the above example,

Count(ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT, ACTAT) = 3.

- We note that *Count*(CG**ATATA**TCC**ATA**G, **ATA**) is equal to 3 (not 2) since we should account for overlapping occurrences of *Pattern* in *Text*.

Counting Words

- To compute *Count*(*Text*, *Pattern*), our plan is to "slide a window" down *Text*, checking whether each *k*-mer substring of *Text* matches *Pattern*.
- We will therefore refer to the *k*-mer starting at position *i* of *Text* as *Text*(*i*, *k*). We will often use **1-based indexing**, meaning that we count starting at 1. In this case, *Text* begins at position 1 and ends at position |*Text*| (|*Text*| denotes the number of symbols in *Text*).
- For example, if *Text* = `GACCATACTG`, then *Text*(5, 3) = `ATA`. Note that the last *k*-mer of *Text* begins at position |*Text*| − *k* + *1*, e.g., the last 3-mer of `GACCATACTG` starts at position 10 − 3 + 1= 8.
- This discussion results in the following *pseudocode for* computing *Count*(*Text*, *Pattern*).

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

The Frequent Words Problem

- We say that *Pattern* is a **most frequent *k*-mer** in *Text* if it maximizes *Count*(*Text*, *Pattern*) among all *k*-mers. You can see that **ACTAT** is a most frequent 5-mer of ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT, and **ATA** is a most frequent 3-mer of CG**ATATA**TCC**ATA**G.
- **STOP and Think**: Can a string have multiple most frequent *k*-mers?

We now have a rigorously defined computational problem.

Frequent Words Problem: *Find the 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

- A straightforward algorithm for finding the most frequent *k*-mers in a string *Text* checks all *k*-mers appearing in this string (there are |*Text*| − *k* + 1 such *k*-mers) and then computes how many times each *k*-mer appears in *Text*.
- To implement this algorithm, called **FREQUENTWORDS**, we will need to generate an array *Count*, where *Count*(*i*) stores *Count*(*Text*, *Pattern*) for *Pattern* = *Text*(*i*, *k*) (see figure below).

Text	**A**	**C**	**T**	G	**A**	**C**	**T**	C	C	C	A	C	C	C	C
COUNT	**2**	1	1	1	**2**	1	1	3	1	1	1	3	3		

The array *Count* for *Text* = ACTGACTCCCACCCC and *k* = 3. For example, *Count*(0) = *Count*(4) = 2 because ACT (shown in boldface) appears twice in *Text*.

- The pseudocode for FrequentWords is shown below.

```
FrequentWords(Text, k)
    FrequentPatterns ← an empty set
    for i ← 1 to |Text| - k + 1
        Pattern ← the k-mer Text(i, k)
        Count(i) ← PatternCount(Text, Pattern)
    maxCount ← maximum value in array Count
    for i ← 1 to |Text| - k + 1
        if Count(i) = maxCount
            add Text(i, k) to FrequentPatterns
    remove duplicates from FrequentPatterns
    return FrequentPatterns
```

Frequent Words in Vibrio cholerae

k	3	4	5	6	7	8	9
count	25	12	8	8	5	4	3
k-mers	tga	atga	gatca tgatc	tgatca	atgatca	atgatcaa	atgatcaag cttgatcat tcttgatca ctcttgatc

The table above reveals the most frequent *k*-mers in the *oriC* region from *Vibrio cholerae* (reproduced below), along with the number of times that each *k*-mer occurs.

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

Frequent Words in Vibrio cholerae

For example, the 9-mer **ATGATCAAG** appears three times in the *oriC* region of *Vibrio cholerae*—is it surprising?

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaag**ATGATCAAG**agaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaagggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcatgtt
tccttaaccctctatttttacggaaga**ATGATCAAG**ctgctgctcttgatcatcgtttc

We highlight a most frequent 9-mer instead of using some other value of *k* because experiments have revealed that bacterial *DnaA* boxes are usually nine nucleotides long. The probability that there exists a 9-mer appearing three or more times in a randomly generated DNA string of length 500 is approximately 1/1300. For more details, see "DETOUR: Probabilities of Patterns in a String" in the print companion.

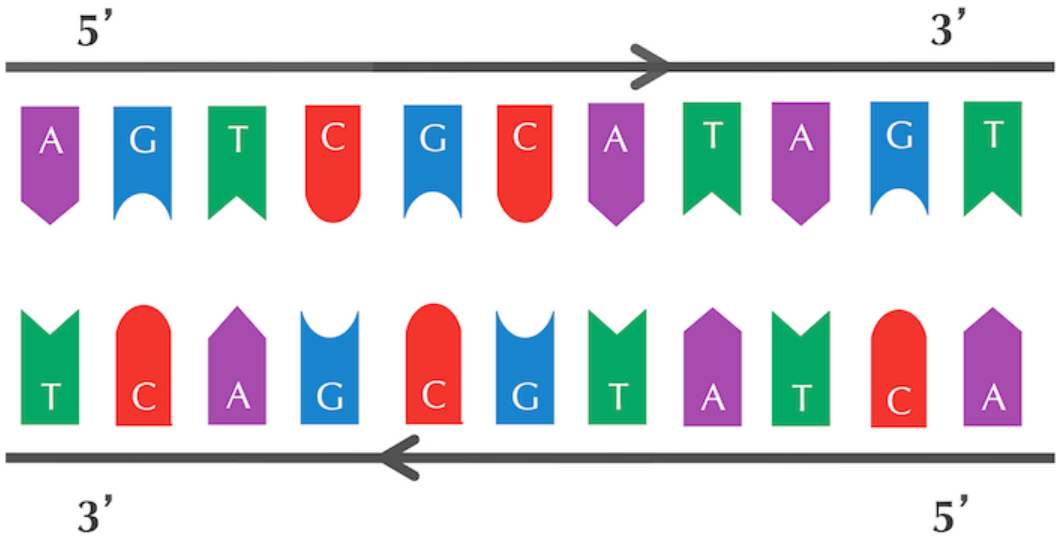
In fact, there are four different 9-mers repeated three or more times in this region: **ATGATCAAG**, CTTGATCAT, TCTTGATCA, and CTCTTGATC. The low likelihood of witnessing even one repeated 9-mer in the *oriC* region of *Vibrio cholerae* leads us to the working hypothesis that one of these four 9-mers may represent a potential *DnaA* box that, when appearing multiple times in a short region, jump-starts replication. But which one?

STOP and Think: Is any one of the four most frequent 9-mers in the *oriC* of *Vibrio cholerae* "more surprising" than the others?

Frequent Words in Vibrio cholerae

Recall that nucleotides A and T are complements of each other, as are G and C. Having one strand and a supply of “free floating” nucleotides, one can imagine the synthesis of a **complementary strand** on a **template strand**. This model of replication was confirmed rigorously by Meselson and Stahl in 1958. The beginning and end of a DNA strand are denoted 5′ (pronounced “five prime”) and 3′ (pronounced “three prime”), respectively. For more details, see "DETOUR: The Most Beautiful Experiment in Biology" in the print companion.

The figure below shows a template strand AGTCGCATAGT and its complementary strand ACTATGCGACT.



Frequent Words in Vibrio cholerae

At this point, you may think that we have made a mistake, since the complementary strand in this figure reads out TCAGCGTATCA from left to right rather than ACTATGCGACT. We have not: each DNA strand has a direction, and the complementary strand runs in the opposite direction to the template strand, as shown by the arrows in the figure. Each strand is read in the 5' → 3' direction.

Given a nucleotide p, we denote its complementary nucleotide as $\overline{p}$. The **reverse complement** of a string $Pattern = p_1 \ldots p_n$ is the string $\overline{Pattern} = \overline{p}_n \ldots \overline{p}_1$ formed by taking the complement of each nucleotide in *Pattern*, then reversing the resulting string. We will need the solution to the following problem throughout this chapter:

Reverse Complement Problem: *Find the reverse complement of a DNA string.*

Input: A DNA string *Pattern*.

Output: $\overline{Pattern}$, the reverse complement of *Pattern*.

Frequent Words in Vibrio cholerae

STOP and Think: Look again at the four most frequent 9-mers in the *oriC* of *Vibrio cholerae* (shown below): **ATGATCAAG**, **CTTGATCAT**, **TCTTGATCA**, and **CTCTTGATC**.

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaag**ATGATCAAG**agaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttac**CTCTTGATCAT**cgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgag**CTCTTGATCAT**gtt
tccttaaccctctatttttacggaaga**ATGATCAAG**ctgctg**CTCTTGATCAT**cgtttc

Now do you notice anything surprising?

Frequent Words in Vibrio cholerae

Interestingly, among the four most frequent 9-mers in *oriC* of *Vibrio cholerae*, **ATGATCAAG** and **CTTGATCAT** are reverse complements of each other, resulting in the six total occurrences of these strings shown below.

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaac
ctgagtggatgacatcaagataggtcgttgtatctccttcctctcgtactctcatgacca
cggaaag**ATGATCAAG**agaggatgatttcttggccatatcgcaatgaatacttgtgactt
gtgcttccaattgacatcttcagcgccatattgcgctggccaaggtgacggagcgggatt
acgaaagcatgatcatggctgttgttctgtttatcttgttttgactgagacttgttagga
tagacggtttttcatcactgactagccaaagccttactctgcctgacatcgaccgtaaat
tgataatgaatttacatgcttccgcgacgatttacct**CTTGATCAT**cgatccgattgaag
atcttcaattgttaattctcttgcctcgactcatagccatgatgagct**CTTGATCAT**gtt
tccttaaccctctatttttacggaaga**ATGATCAAG**ctgctgct**CTTGATCAT**cgtttc

Frequent Words in Vibrio cholerae

Finding a 9-mer that appears six times (either as itself or as its reverse complement) in a DNA string of length 500 is far more surprising than finding a 9-mer that appears three times (as itself). This observation leads us to the working hypothesis that `ATGATCAAG` and its reverse complement `CTTGATCAT` indeed represent *DnaA* boxes in *Vibrio cholerae*. This computational conclusion makes sense biologically because the *DnaA* protein that binds to *DnaA* boxes and initiates replication does not care which of the two strands it binds to. Thus, for our purposes, both `ATGATCAAG` or `CTTGATCAT` represent *DnaA* boxes.

However, before concluding that we have found the *DnaA* box of *Vibrio cholerae*, the careful bioinformatician should check if there are other short regions in the *Vibrio cholerae* genome exhibiting multiple occurrences of `ATGATCAAG` (or `CTTGATCAT`). After all, maybe these strings occur as repeats throughout the entire *Vibrio cholerae* genome, rather than just in the *oriC* region. To this end, we need to solve the following problem.

Pattern Matching Problem: *Find all occurrences of a pattern in a string.*

Input: Strings *Pattern* and *Genome*.

Output: All starting positions in *Genome* where *Pattern* appears as a substring.