Содержание

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

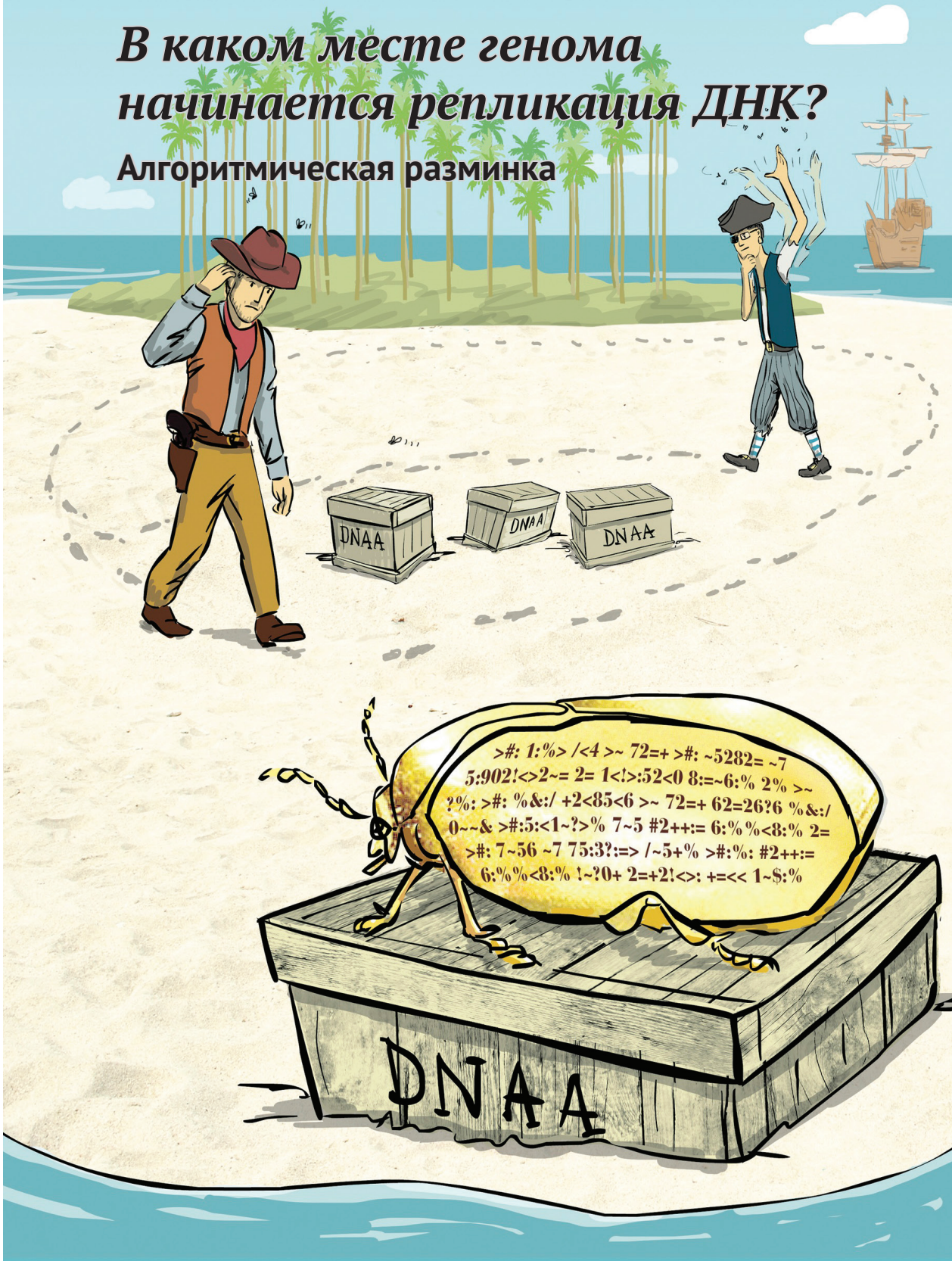
Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Глава 1

В каком месте генома начинается репликация ДНК?

Алгоритмическая разминка



Путешествие в тысячу миль...

Репликация генома – одна из важнейших задач, выполняемых в клетке. Прежде чем клетка сможет делиться, она должна сначала реплицировать свой геном, чтобы каждая из двух дочерних клеток наследовала свою собственную копию генома. В 1953 году Джеймс Уотсон и Фрэнсис Крик завершили свою знаменательную статью о двойной спирали ДНК ставшей теперь популярной фразой:

> *От нашего внимания не ускользнуло, что специфическое выстраивание пар азотистых оснований, которое мы постулировали, сразу предполагает возможный механизм копирования генетического материала.*

Они предположили, что две цепи родительской молекулы ДНК раскручиваются во время репликации, и затем каждая родительская цепь действует как матрица для синтеза новой молекулярной цепи. В результате процесс репликации начинается с пары комплементарных цепей ДНК и заканчивается двумя парами комплементарных цепей, как показано на рис. 1.1.

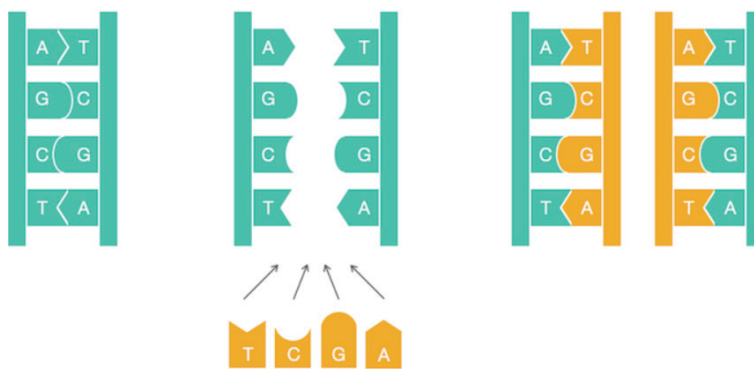


Рис. 1.1 Наивный взгляд на репликацию ДНК. Нуклеотиды аденин (A) и тимин (T) комплементарны друг другу, как и цитозин (C) с гуанином (G). Комплементарные нуклеотиды связываются друг с другом в ДНК

Хотя на рис. 1.1 репликация ДНК представлена на простом уровне, детали репликации оказались гораздо более сложными, чем предполагали Уотсон и Крик; как мы увидим далее, для обеспечения репликации ДНК требуется поразительный механизм молекулярной логистики.

На первый взгляд, ученый-компьютерщик может и не подумать, что эти детали имеют какое-либо значение для вычислений. Чтобы алгоритмически имитировать процесс, показанный на рис. 1.1, нам нужно всего лишь взять строку, представляющую геном, и выдать ее копию! И все же, если мы найдем время для обзора лежащего в основе этого биологического процесса, мы будем вознаграждены новыми алгоритмическими идеями, полученными в анализе процесса репликации.

Репликация начинается в области генома, называемой **точкой начала репликации** (обозначается *ori*), и выполняется молекулярными копировальными машинами, называемыми **ДНК-полимеразами**. Обнаружение *ori* представляет собой важную задачу не только для понимания того, как клетки реплицируются, но и для решения различных биомедицинских задач. Например, в некоторых методах генной терапии используются генетически сконструированные мини-геномы, которые называются **вирусными векторами**, потому что они способны проникать через клеточные стенки (прямо как настоящие вирусы). Вирусные векторы, несущие искусственные гены, использовались в сельском хозяйстве для создания морозоустойчивых томатов и кукурузы, устойчивой к пестицидам. В 1990 году генная терапия была впервые успешно проведена на людях, когда она спасла жизнь четырехлетней девочке, страдающей тяжелым комбинированным иммунодефицитом; девочка была настолько уязвима для инфекций, что была вынуждена жить в стерильной среде.

Идея генной терапии состоит в том, чтобы намеренно заразить пациента, у которого отсутствует важный для жизнедеятельности ген, вирусным вектором, содержащим искусственный ген, кодирующий так называемый терапевтический белок. Оказавшись внутри клетки, вектор реплицируется и в конечном итоге производит множество копий терапевтического белка, который, в свою очередь, лечит болезнь пациента. Чтобы гарантировать, что вектор действительно реплицируется внутри клетки, биологи должны знать, где находится точка начала репликации, *ori,* в геноме вектора, и убедиться, что выполняемые ими генетические манипуляции не влияют на него.

В следующей задаче мы предполагаем, что геном имеет одно начало репликации и представлен в виде **цепи ДНК** или цепи нуклеотидов из четырехбуквенного алфавита {A, C, G, T}.

Задача поиска точки начала репликации: *найти ori в геноме*

Input: ДНК-цепь *Genome*.

Output: локация *ori* в *Genome*.

ОСТАНОВИТЕСЬ и задумайтесь. Является ли эта биологическая задача четко сформулированной вычислительной задачей?

Хотя задача поиска точки начала репликации ставит законный биологический вопрос, она не представляет собой четко сформулированную вычислительную задачу. Действительно, биологи могут немедленно запланировать эксперимент по обнаружению *ori*: например, они могут удалять различные короткие сегменты генома, пытаясь найти сегмент, удаление которого останавливает репликацию. Но специалисты по информатике, с другой стороны,

покачали бы головами и потребовали бы больше информации, прежде чем они смогли бы даже начать думать о задаче.

Почему биологов должно волновать, что думают ученые-компьютерщики? Вычислительные методы в настоящее время являются единственным реальным способом ответить на многие вопросы современной биологии. Во-первых, эти методы намного быстрее, чем экспериментальные методы; во-вторых, результаты многих экспериментов не могут быть интерпретированы без вычислительного анализа. В частности, существующие экспериментальные методы определения локации *ori* требуют много времени. В результате *ori* был экспериментально обнаружен только у нескольких видов. Таким образом, мы хотели бы разработать вычислительный метод поиска *ori*, чтобы биологи могли тратить свое время и деньги на другие задачи.

Скрытые сообщения в точке начала репликации

DnaA-боксы

В оставшейся части этой главы мы сосредоточимся на относительно простом случае обнаружения *ori* в бактериальных геномах, большинство из которых состоит из одной кольцевой хромосомы. Исследования показали, что область бактериального генома, кодирующая *ori*, обычно имеет длину в несколько сотен нуклеотидов. Наш план состоит в том, чтобы начать с бактерии, у которой известны ее *ori*, а затем определить, что делает этот участок генома особенным, чтобы разработать вычислительный метод для нахождения *ori* у других бактерий. Наш пример – *Vibrio cholerae*, бактерия, вызывающая холеру; вот последовательность нуклеотидов, расположенная в ее *ori*:

```
Atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttat
ccacaacctgagtggatgacatcaagataggtcgttgtatctccttcctctcg
tactctcatgaccacggaaagatgatcaagagaggatgatttcttggccatat
cgcaatgaatacttgtgacttgtgcttccaattgacatcttcagcgccatatt
gcgctggccaaggtgacggagcgggattacgaaagcatgatcatggctgttgt
tctgtttatcttgttttgactgagacttgttaggatagacggtttttcatcac
tgactagccaaagccttactctgcctgacatcgaccgtaaattgataatgaat
ttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaagatct
tcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatcat
gtttccttaaccctctatttttacggaagaatgatcaagctgctgctcttga
tcatcgtttc
```

☞ **Загрузить данные 1.1**

Откуда бактериальная клетка знает, что нужно начинать репликацию именно в этом коротком участке гораздо более длинной хромосомы *Vibrio cholerae*,

состоящей из 1 108 250 нуклеотидов? Должно быть какое-то скрытое сообщение в этой области, приказывающее клетке начать репликацию именно здесь. Действительно, мы знаем, что инициация репликации опосредована ***DnaA***, белком, который связывается с коротким сегментом внутри *ori*, известным как ***DnaA*-бокс**. Вы можете думать о *DnaA*-боксе как о сообщении в цепи ДНК, сообщающем *DnaA* белку: «Присоединяйся сюда!» Вопрос в том, как найти это скрытое сообщение, не зная заранее, как оно выглядит, – сможете ли вы его найти? Другими словами, можете ли вы найти что-то, чем выделяется *ori*? Это рассуждение ставит следующую задачу.

Задача поиска скрытого сообщения: *найти скрытое сообщение в точке начала репликации.*

Input: строка *Text* (представляющая начало репликации генома).

Output: скрытое сообщение в *Text*.

ОСТАНОВИТЕСЬ и задумайтесь. Представляет ли эта задача четко сформулированную вычислительную задачу?

Скрытые сообщения в «Золотом жуке»

Хотя задача скрытого сообщения ставит законный интуитивный вопрос, она все еще не имеет абсолютно никакого смысла для ученого-компьютерщика, поскольку понятие скрытого сообщения точно не определено. Область *ori* холерного вибриона в настоящее время так же загадочна, как пергамент, обнаруженный Уильямом Леграном в рассказе Эдгара Аллана По «Золотой жук». На пергаменте было написано следующее:

```
53++!305))6*;4826)4+.)4+);806*;48!8'60))85;1+(;:+*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+;161;:188;+?;
```

Увидев пергамент, рассказчик замечает: «Если бы все драгоценности Голконды ждали меня после решения этой загадки, я совершенно уверен, что не смог бы их заработать». Легран возражает: «Вполне можно сомневаться, способна ли человеческая изобретательность построить такого рода загадку, которую человеческая находчивость при правильном применении не могла бы разрешить». Он обращает внимание, что три последовательных символа «;48» появляются на пергаменте с удивительной частотой:

```
53++!305))6;4826)4+.)4+);806*;48!8'60))85;1+(;:+*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!8
5;4)485!528806*81(+9 ;48;(88;4(+?34;48)4+;161;:188;+?;
```

Легран ранее уже сделал вывод, что пираты говорят по-английски; поэтому он предположил, что высокая частота «;48» подразумевает, что она кодирует наиболее часто встречающееся английское слово – артикль «THE». Заменяя каждый символ, Легран получил немного более простой текст для расшифровки, который в конечном итоге привел его к зарытому сокровищу. Можете расшифровать и это сообщение?

```
53++!305))6*THE26)H+.)H+)TE06*THE!E‘60))E5T1+(T:+*E!E3(EE)5*!T
H6(TEE*96*?TE)*+(THE5)T5*!2:*+(TH956*2(5*H)E’E*TH0692E5)T)6!E)
H++T1(+9THE0E1TE:E+1THE!E5TH)HE5!52EE06*E1(+9THET(EETH(+?3HT
HE)H+T161T:1EET+?T
```

Подсчет слов

Опираясь на предположение, что ДНК – это язык сам по себе, давайте воспользуемся методом Леграна и посмотрим, сможем ли мы найти какие-нибудь неожиданно часто встречающиеся «слова» в *ori* холерного вибриона (*Vibrio cholerae*). Имеет смысл искать часто встречающиеся слова в *ori*, потому что для различных биологических процессов определенные цепи нуклеотидов удивительно часто появляются в небольших областях генома. Это связано с тем, что некоторые белки могут связываться с ДНК только в том случае, если присутствует определенная последовательность нуклеотидов, и если существует больше вхождений этой последовательности, то более вероятно, что связывание произойдет успешно. (Также менее вероятно, что мутация нарушит процесс связывания.)

Например, **ACTAT** – удивительно частая подстрока

ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT.

Мы используем термин «***k*-меры**» для обозначения строки длины *k* и определяем *Count*(*Text, Pattern*) как количество раз, когда *k*-мер *Pattern* появляется как подстрока строки *Text*. Следуя приведенному выше примеру,

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

Обратите внимание, что *Count*(CG**ATATA**TCC**ATA**G, **ATA**) равно 3 (а не 2), так как мы должны учитывать перекрывающиеся вхождения *Pattern* в *Text*.

Наш план вычисления *Count*(*Text, Pattern*) состоит в том, чтобы «сдвигать окно» вдоль строки *Text*, проверяя, соответствует ли каждая подстрока *k*-мера текста *Text* образцу *Pattern*. Поэтому мы будем ссылаться на *k*-мер, начинающийся с позиции *i* текста, как на *Text(i, k)*. В этой книге мы часто будем использовать **ноль-индексацию** (нумерацию на основе нуля), что означает, что мы начинаем отсчет с 0, а не с 1. В этом случае *Text* начинается с позиции 0 и заканчивается в позиции |*Text*| – 1 (|*Text*| обозначает количество символов в тексте). Например, если *Text* = GACCATACTG, то *Text*(4, 3) = ATA. Обратите внимание, что последний *k*-мер *Text* начинается с позиции |*Text*| – *k*, например последний

3-мер GACCATACTG начинается с позиции 10 – 3 = 7. Это рассуждение приводит к следующему псевдокоду для вычисления *Count*(*Text, Pattern*).

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

Важное примечание. В этом тексте мы используем термин **псевдокод** для описания алгоритмов, с которыми сталкиваемся при решении задач современной биологии. Псевдокод – это универсальный метод описания алгоритмов, более точный, чем человеческий язык, но не требующий от нас увязнуть в синтаксисе конкретного языка программирования.

Задача поиска часто встречающихся слов

Мы говорим, что *Pattern* является наиболее частым *k*-мером в *Text*, если он максимизирует *Count*(*Text, Pattern*) среди всех *k*-меров. Вы можете видеть, что **ACTAT** является наиболее частым 5-мером для *Text* = ACA**ACTAT**GCAT**ACTAT**CGGGA**ACTAT**CCT, а **ATA** – наиболее частым 3-мером для *Text* = CG**ATATA**TCC**ATA**G.

ОСТАНОВИТЕСЬ и задумайтесь. Может ли строка иметь несколько наиболее часто встречающихся *k*-меров?

Теперь у нас есть строго определенная вычислительная задача.

Задача поиска часто встречающихся слов: *найдите наиболее часто встречающиеся k-меры в строке.*

Input: строка *Text* и целое число *k*.

Output: все наиболее часто встречающиеся *k*-меры в тексте.

Прямой алгоритм нахождения наиболее часто встречающихся *k*-меров в строке *Text* проверяет все *k*-меры, встречающиеся в этой строке (имеется |*Text*| – *k* + 1 таких *k*-меров), а затем вычисляет, сколько раз каждый *k*-мер появляется в *Text*. Для реализации этого алгоритма, называемого **FrequentWords**, нам потребуется сгенерировать массив *Count*, где *Count*(*i*) содержит *Count*(*Text*, *Pattern*) для *Pattern* = *Text*(*i, k*) (рис. 1.2).

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		

Рис. 1.2 Массив *Count* для *Text* = ACTGACTCCCACCCC и *k* = 3. Например, *Count*(0) = *Count*(4) = 2, потому что ACT (выделен жирным шрифтом) дважды встречается в строке *Text* в позициях 0 и 4

```
FrequentWords(Text, k)
    FrequentPatterns ← пустой набор
    for i ← 0 до |Text| – k
        Pattern ← k-мер Text(i, k)
        Count(i) ← PatternCount(Text, Pattern)
    maxCount ← максимальная величина массива Count
    for i ← 0 до |Text| – k
        if Count(i) = maxCount
            add Text(i, k) до FrequentPatterns
    удалить дубликаты из FrequentPatterns
    return FrequentPatterns
```

ОСТАНОВИТЕСЬ и задумайтесь. Насколько быстро работает **FrequentWords**?

Хотя **FrequentWords** находит наиболее часто встречающиеся *k*-меры, он не очень эффективен. Каждый вызов **PatternCount**(*Text, Pattern*) проверяет, появляется ли *k*-мер *Pattern* в позиции 0 *Text*, позиции 1 *Text* и т. д. Поскольку каждый *k*-мер требует $|Text| - k + 1$ таких проверок, каждая из которых требует *k* сравнений, общее количество шагов **PatternCount**(*Text, Pattern*) равно $(|Text| - k + 1) \cdot k$. Более того, **FrequentWords** должен вызывать **PatternCount** $|Text| - k + 1$ раз (по одному разу для каждого *k*-мера *Text*), так что общее количество шагов равно $(|Text| - k + 1) \cdot (|Text| - k + 1) \cdot k$. Чтобы упростить дело, ученые-компьютерщики часто говорят, что время выполнения **FrequentWords** имеет верхнюю границу $|Text|^2 \cdot k$ шагов, и ссылаются на сложность этого алгоритма как $O(|Text|^2 \cdot k)$ (см. **СОПУТСТВУЮЩИЕ МАТЕРИАЛЫ: Big-*O* («*O* большое»)**).

Если $|Text|$ и *k* малы, как в случае поиска *DnaA*-боксов в типичных бактериальных *ori*, тогда алгоритм со временем работы $O|Текст|^2 \cdot k$ вполне приемлем. Но как только мы найдем новое биологическое приложение, требующее от нас решения задачи часто используемых слов для очень длинного текста, мы быстро столкнемся с проблемами.

Более быстрый подход к задаче частых слов

Если бы вам нужно было решить задачу о часто встречающихся словах вручную для небольшого примера, вы, вероятно, сформировали бы таблицу, подобную