

УДК 575.112
ББК 28.071.3
К63

Певзнер П., Компо Ф.

К63 Алгоритмы биоинформатики / пер. с англ. И. Л. Люско. – М.: ДМК Пресс, 2022. – 680 с.: ил.

ISBN 978-5-93700-175-7

Если вам интересно, какую роль играет биоинформатика и теория графов в секвенировании ДНК и антибиотиков, что такое молекулярные часы ДНК, чем различаются геномы человека и мыши, какое животное заразило нас коронавирусом, как строятся эволюционные деревья, как дрожжи научились делать вино, как обнаруживают локацию болезнетворных мутаций в геноме, почему биологи до сих пор не разработали вакцину от ВИЧ, произошла ли курица от тираннозавра и прочее, эта книга для вас!

Издание предназначено для специалистов, а также тех, кто интересуется вышеперечисленными темами. Кроме большого количества алгоритмических псевдокодов, книга содержит массу интереснейшей и самой свежей информации, доступной для специалистов.

УДК 575.112
ББК 28.071.3

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-5-93700-175-7 (рус.)

Copyright © 2015 by Phillip Compeau
and Pavel Pevzner
© Перевод, оформление, издание,
ДМК Пресс, 2022

Содержание

<i>От издательства</i>	17
------------------------------	----

Глава 1. В каком месте генома начинается репликация ДНК?	18
Путешествие в тысячу миль.....	19
Скрытые сообщения в точке начала репликации	21
<i>DnaA</i> -боксы	21
Скрытые сообщения в «Золотом жуке»	22
Подсчет слов.....	23
Задача поиска часто встречающихся слов.....	24
Более быстрый подход к задаче частых слов.....	25
Часто используемые слова в <i>Vibrio cholerae</i>	27
Некоторые скрытые сообщения более примечательны, чем другие	28
Взрыв скрытых сообщений.....	32
Поиск скрытых сообщений в нескольких геномах	32
Задача поиска сгустков	33
Самое простое объяснение процесса репликации ДНК	35
Асимметрия репликации	38
Специфическая статистика прямой и обратной полупепей	42
Неизвестный биологический феномен или статистическая случайность?	42
Дезаминирование.....	44
Диаграмма смещения.....	45
Некоторые скрытые сообщения более неуловимы, чем другие	48
Последняя попытка найти <i>DnaA</i> -боксы в <i>E. Coli</i>	52
Эпилог. Осложнения в предсказаниях <i>ori</i>	54
Открытые проблемы	56
Множественные точки начала репликации в бактериальном геноме	56
Поиск источников репликации у архей	58
Поиск точек начала репликации у дрожжей.....	60
Вычисление вероятностей паттернов в строке	61

Зарядные станции.....	62
Массив частот	62
Преобразование Patterns в Numbers и наоборот	64
Поиск часто встречающихся слов путем сортировки.....	66
Решение задачи поиска сгустков.....	67
Решение задачи часто встречающихся слов с несовпадениями.....	69
Генерация окрестности строки	70
Поиск часто встречающихся слов с несовпадениями путем сортировки ...	72
Сопутствующие материалы	73
Оценка «О большого» (Big-O)	73
Вероятности паттернов в строке	74
Самый красивый эксперимент в биологии.....	79
Направленность цепей ДНК.....	81
Ханойские башни	81
Парадокс перекрывающихся слов	84
Библиографические примечания.....	86

Глава 2. Какие сегменты ДНК играют роль молекулярных часов?

Есть ли у нас «часовой ген»?	87
Найти мотив сложнее, чем вы думаете.....	88
Идентификация вечернего элемента	89
Игра в прятки с мотивами.....	90
Метод грубой силы поиска мотива	92
Считаем мотивы.....	93
От мотивов к матрицам профиля и консенсусным строкам	93
На пути к более адекватной функции оценки мотивов.....	96
Энтропия и motif logo.....	97
От поиска мотива к поиску медианной строки	98
Задача поиска мотива.....	98
Переформулировка задачи поиска мотива.....	99
Задача поиска медианной строки	101
Почему мы переформулировали задачу поиска мотива?	103
Жадный алгоритм поиска мотива	104
Использование матрицы профиля для бросания костей	104
Анализ жадного алгоритма поиска мотива	106
Поиск мотива и Оливер Кромвель	107
Какова вероятность того, что завтра не взойдет солнце?	107
Правило преемственности Лапласа	108
Улучшенный алгоритм жадного поиска мотивов	109
Рандомизированный поиск мотива	112
Игра в кости для поиска мотивов.....	112
Почему рандомизированный поиск мотивов работает.....	114
Почему рандомизированный алгоритм работает так хорошо?.....	116
Сэмплирование по Гиббсу.....	119

Сэмплирование по Гиббсу в действии	121
Эпилог. Как туберкулез впадает в спячку, чтобы спрятаться от антибиотиков?	124
Зарядная станция	127
Решение задачи медианной строки.....	127
Сопутствующие материалы	128
Экспрессия генов	128
ДНК-чипы	128
Игла Бюффона	129
Сложности в поиске мотива.....	132
Относительная энтропия.....	132
Библиографические примечания.....	134
Глава 3. Как мы собираем геномы?	135
Взрывающиеся газеты	136
Задача реконструкции строки.....	139
Сборка генома сложнее, чем вы думаете.....	139
Реконструкция строк из k -меров	139
Повторы усложняют сборку генома	142
Реконструкция строк как прогулка по графу перекрытий.....	143
От строки к графу	143
Геном исчезает	146
Два способа представления графов	147
Гамильтоновы пути и универсальные строки	148
Другой граф для реконструкции строк	150
Склеивание узлов и графы де Брюйна	150
Прогулка по графу де Брюйна	152
Эйлеровы пути	152
Другой способ построения графов де Брюйна	153
Построение графов де Брюйна из композиции k -меров	155
Графы де Брюйна в сравнении с графами перекрытия	156
Семь мостов Кенигсберга	157
Теорема Эйлера.....	160
От теоремы Эйлера к алгоритму нахождения эйлеровых циклов	163
Построение эйлеровых циклов	163
От эйлеровых циклов к эйлеровым путям	164
Создание универсальных строк.....	165
Сборка геномов из рид-пар	167
От ридов к рид-парам	167
Преобразование рид-пар в длинные виртуальные риды	169
От композиции к спаренной композиции.....	170
Парные графы графы де Брюйна	172
Ловушка парных графов де Брюйна	173
Эпилог. Сборка генома – работа с реальными данными секвенирования	176
Разбиваем риды на k -меры	176

Фрагментация генома на контиги	177
Сборка ридов с возможными ошибками	179
Определение кратности ребер в графах де Брюйна	180
Зарядные станции	181
Влияние склейки на матрицу смежности	181
Генерация всех эйлеровых циклов	182
Реконструкция строки, записанной как путь в парном графе де Брюйна	184
Максимальные неветвящиеся пути в графе	186
Сопутствующие материалы	187
Краткая история технологий секвенирования ДНК	187
Повторы в геноме человека	189
Графы	190
Игра «Икосиан»	192
Разрешимые и неразрешимые задачи	193
От Эйлера до Гамильтона и де Брюйна	195
Семь мостов Калининграда	196
Подводные камни сборки двухцепочечной ДНК	196
«ЛУЧШАЯ» теорема	198
Библиографические примечания	198
Глава 4. Как мы секвенируем антибиотики?	200
Открытие антибиотиков	201
Как бактерии производят антибиотики?	202
Как пептиды кодируются геномом	202
Где в геноме <i>Bacillus brevis</i> закодирован тироцидин?	205
От линейных к циклическим пептидам	206
Уклоняясь от центральной догмы молекулярной биологии	207
Секвенирование антибиотиков путем их дробления на части	208
Введение в масс-спектрометрию	208
Задача секвенирования циклопептидов	209
Алгоритм грубой силы для секвенирования циклопептидов	211
Алгоритм ветвей и границ для секвенирования циклопептидов	213
Масс-спектрометрия и гольф	216
От теоретических к реальным спектрам	216
Адаптация секвенирования циклопептидов для спектров с ошибками	217
От 20 до более чем 100 аминокислот	221
Спектральная свертка спасает положение	222
Эпилог. От смоделированных спектров – к реальным	226
Зарядные станции	228
Создание теоретического спектра пептида	228
Насколько быстро выполняется алгоритм CyclopeptideSequencing?	230
Сокращение списка пептидов Leaderboard	231
Сопутствующие материалы	232
Гаузе и «лысенковщина»	232

Открытие кодонов.....	234
Чувство кворума.....	235
Молекулярная масса	235
Сленоцистеин и пирролизин.....	236
Псевдополиномиальный алгоритм для Теоремы магистрали	236
Расщепленные гены.....	237
Библиографические примечания.....	239
Глава 5. Как мы сравниваем участки ДНК?	240
Взлом нерибосомного кода.....	241
Клуб галстуков РНК.....	241
От сравнения белков к нерибосомному коду	241
Что общего между онкогенами и факторами роста?.....	243
Введение в выравнивание последовательностей.....	244
Выравнивание последовательности как игра.....	244
Выравнивание последовательностей и самая длинная общая подпоследовательность.....	246
Туристическая задача Манхэттена	247
Какова наилучшая стратегия осмотра достопримечательностей?	247
Достопримечательности в произвольном ориентированном графе.....	251
Выравнивание последовательности – это замаскированная туристическая задача Манхэттена	252
Введение в динамическое программирование: задача размена монет.....	256
Жадный обмен денег	256
Рекурсивный обмен денег	257
Размениваем деньги с помощью динамического программирования.....	258
Новый взгляд на туристическую задачу Манхэттена	260
От Манхэттена к произвольному DAG	265
Выравнивание последовательности как построение графа в стиле Манхэттена.....	265
Динамическое программирование в произвольном графе DAG	266
Топологические порядки.....	268
Возвращаясь к графу выравнивания	273
Считаем выравнивания	276
Что не так с моделью LCS?.....	276
Матрицы счета	278
От глобального к локальному выравниванию	279
Глобальное выравнивание	279
Ограничения глобального выравнивания.....	280
Бесплатные поездки на такси в графе выравнивания	283
Меняющиеся грани выравнивания последовательности	286
Задача 1. Расстояние редактирования.....	286
Задача 2. Настройка выравнивания	287
Задача 3. Выравнивание с перекрытием	288
Штрафы за вставки и удаления при выравнивании последовательности.....	289

Штрафы за аффинные пробелы	289
Строительство графа Манхэттена на трех уровнях.....	292
Компактное выравнивание последовательности.....	295
Вычисление счета выравнивания с использованием линейной памяти....	295
Задача среднего узла	297
Удивительно быстрый и экономичный алгоритм выравнивания	299
Задача среднего ребра.....	302
Эпилог. Множественное выравнивание последовательностей	304
Построение трехмерного Манхэттена.....	304
Жадный алгоритм множественного выравнивания.....	306
Сопутствующие материалы	309
Светлячки и нерибосомный код	309
Поиск LCS без постройки города	310
Построение топологической сортировки	311
Матрица счета PAM	312
Алгоритмы «разделяй и властвуй».....	313
Счет множественных выравниваний.....	315
Библиографические примечания.....	317

Глава 6. Есть ли в человеческом геноме «хрупкие» области?

области?	318
О мышцах и людях	319
Насколько различаются геномы человека и мыши?.....	320
Синтенные блоки.....	320
Реверсии.....	321
Точки перестановки.....	323
Модель эволюции хромосом со случайными разрывами	324
Сортировка по реверсиям.....	327
Жадный алгоритм сортировки по реверсиям.....	331
Точки останова.....	333
Что такое точки останова?.....	333
Счет точек останова	334
Сортировка по реверсиям для устранения точек останова.....	335
Рекомбинации в геномах опухолей.....	337
От монохромосомных к мультихромосомным геномам.....	338
Транслокации, слияния и расщепления	338
От генома к графу.....	339
Двойные разрывы.....	340
Графы точек останова	343
Вычисление дистанции двойного разрыва.....	346
Горячие точки рекомбинации в геноме человека.....	349
Модель случайных разрывов соответствует теореме о дистанции двойного разрыва.....	349
Модель хрупких разрывов	350
Эпилог. Конструирование синтенных блоков	352

Геномные точечные диаграммы и общие k-меры.....	352
Поиск общих k-меров	353
Построение синтенных блоков из общих k-меров.....	356
Синтенные блоки как связные компоненты в графах	358
Зарядные станции.....	362
От геномов к графу точек останова.....	362
Решение задачи сортировки по двойным разрывам.....	365
Сопутствующие материалы	367
Почему генный состав X-хромосом так консервативен?.....	367
Открытие геномных рекомбинаций.....	367
Экспоненциальное распределение	368
Сортировка блинов Билла Гейтса и Дэвида Х. Козна.....	369
Сортировка линейных перестановок по реверсиям	370
Библиографические примечания.....	372

Глава 7. Какое животное заразило нас

коронавирусом?	373
Самая быстрая вспышка.....	374
Проблемы в отеле «Метрополь».....	374
Эволюция SARS	374
Преобразование матриц расстояний в эволюционные деревья	376
Построение матрицы расстояний из геномов коронавируса.....	376
Эволюционные деревья в виде графов	377
Построение филогении по расстояниям	381
На пути к алгоритму построения филогении по расстоянию	384
В поисках соседних листьев	384
Вычисление длины ветвей	386
Аддитивная филогения	389
Обрезка дерева	389
Прикрепление ветви	390
Алгоритм реконструкции филогении по расстоянию.....	391
Построение эволюционного дерева коронавирусов.....	392
Использование метода наименьших квадратов для построения приблизительных филогений	393
Ультраметрические эволюционные деревья.....	395
Алгоритм объединения соседей	400
Преобразование матрицы расстояний в матрицу объединения соседей...400	
Анализ коронавирусов с помощью алгоритма объединения соседей.....404	
Ограничения методов реконструкции эволюционного дерева по расстояниям	406
Реконструкция эволюционного дерева по признакам	406
Таблицы признаков.....	406
От анатомических к генетическим признакам.....	407
Сколько раз эволюция изобретала крылья для насекомых?.....	408
Задача минимального показателя экономии	409

Задача максимальной экономии.....	416
Эпилог. Эволюционные деревья в борьбе с преступностью.....	423
Сопутствующие материалы.....	424
Когда HIV перешел от приматов к человеку?.....	424
Поиск дерева с помощью настройки матрицы расстояний.....	425
Условие четырех точек.....	427
Заразили ли нас атипичной пневмонией летучие мыши?.....	428
Почему алгоритм объединения соседей работает?.....	430
Вычисление длин ветвей в алгоритме объединения соседей.....	434
Большая панда: медведь или енот?.....	435
Откуда пришли люди?.....	436
Библиографические примечания.....	439

Глава 8. Как дрожжи научились делать вино?.....

Эволюционная история виноделия.....	441
Как давно мы зависим от алкоголя?.....	441
Диауксический сдвиг.....	442
Идентификация генов, ответственных за диауксический сдвиг.....	442
Две эволюционные гипотезы с разными судьбами.....	442
Какие гены дрожжей вызывают диауксический сдвиг.....	443
Введение в кластеризацию.....	444
Анализ экспрессии генов.....	444
Кластеризация генов дрожжей.....	448
Принцип правильной кластеризации.....	449
Кластеризация как задача оптимизации.....	451
Самый дальний первый обход.....	453
Самый дальний первый обход.....	453
Кластеризация k -средних.....	455
Искажение квадрата ошибки.....	455
Кластеризация k -средних и центр тяжести.....	457
Алгоритм Ллойда.....	459
От центров к кластерам и обратно.....	459
Инициализация алгоритма Ллойда.....	462
Инициализатор k -means++.....	463
Кластеризация генов, вовлеченных в диауксический сдвиг.....	463
Ограничения кластеризации k -средних.....	465
Ограничения кластеризации k -средних.....	465
От подбрасывания монеты к кластеризации k -средних.....	467
Подбрасывание монет с неизвестной симметрией.....	467
В чем же состоит вычислительная задача?.....	470
От подбрасывания монеты к алгоритму Ллойда.....	471
Вернемся к кластеризации.....	473
Принятие мягких решений при подбрасывании монет.....	474
Максимизация ожиданий: E-шаг.....	474
Максимизация ожиданий: M-шаг.....	475

Алгоритм максимизации ожидания	477
Мягкая кластеризация k -средних	477
Применение алгоритма максимизации ожидания к кластеризации	477
От центров к мягким кластерам	477
От мягких кластеров к центрам	480
Иерархическая кластеризация	481
Введение в кластеризацию по расстояниям	481
Определение кластеров по структуре дерева	484
Анализ диауксического сдвига с иерархической кластеризацией	487
Эпилог. Кластеризация образцов опухолей	490
Сопутствующие материалы	491
Полногеномная дупликация или серия дупликаций?	491
Измерение экспрессии генов	492
ДНК-микрочипы	492
Доказательство теоремы о центре тяжести	494
Матрица экспрессии генов и матрица расстояний/сходств	495
Кластеризация и испорченные клики	495
Библиографические примечания	498

Глава 9. Как мы обнаруживаем локацию болезнетворных мутаций?

.....	499
Что вызывает синдром Одо?	500
Введение во множественное выравнивание последовательностей	501
Объединение Patterns в префиксное дерево	502
Построение префиксного дерева Trie	502
Применение префиксного дерева к множественному выравниванию	504
Предварительная обработка генома как альтернатива	507
Суффиксные попытки (suffix tries)	507
Использование суффиксных попыток для сопоставления последовательностей	508
Суффиксные деревья (suffix trees)	510
Суффиксные массивы	514
Выравнивание паттерна с суффиксным массивом	515
Преобразование Барроуза–Уилера	517
Сжатие генома	517
Построение преобразования Барроуза–Уилера	517
От повторов к сериям	519
Первая попытка инвертирования преобразования Барроуза–Уилера	520
Свойство «первый–последний» и инвертирование преобразования Барроуза–Уилера	523
Свойство «первый–последний»	523
Использование свойства «первый–последний» для инвертирования преобразования Барроуза–Уилера	526
Сопоставление последовательностей с помощью преобразования Барроуза–Уилера	530

Первая попытка сопоставления паттернов Барроуза–Уилера	530
Перемещение по последовательности назад	531
Маппинг «последний–первый»	533
Делаем сопоставление паттернов по Барроузу–Уилеру быстрее	535
Замена маппинга «последний–первый» оценочными массивами	535
Удаление первого столбца матрицы Барроуза–Уилера	538
Где находятся совпадающие паттерны?	539
Барроуз и Уиллер устанавливают контрольные точки	541
Эпилог. Устойчивое к несовпадениям картирование рида	543
Сведение приблизительного сопоставления с паттерном к точному	543
BLAST: Сравнение последовательности с базой данных	545
Приблизительное сопоставление последовательностей с помощью преобразования Барроуза–Уилера	546
Сопутствующие материалы	548
Построение суффиксного дерева	548
Решение задачи самой длинной общей подстроки	551
Построение частичного суффиксного массива	553
Эталонный геном человека	554
Рекомбинации, вставки и делеции в геномах человека	554
Алгоритм Ахо–Корасик	555
Суффиксные массивы и суффиксные деревья	556
Бинарный поиск	561
Библиографические примечания	562

Глава 10. Почему биологи до сих пор не разработали вакцину от ВИЧ?	563
Классификация фенотипа ВИЧ	564
Каким образом ВИЧ ускользает от иммунной системы человека?	564
Ограничения метода выравнивания последовательностей	566
Азартные игры с якудза	568
Две монеты в рукаве у дилера	569
Поиск CG-островов	571
Скрытые марковские модели	572
От подбрасывания монеты к скрытой марковской модели	572
Диаграмма НММ	573
Переформулировка задачи казино	574
Задача декодирования	577
Граф Витерби	577
Алгоритм Витерби	579
Насколько быстр алгоритм Витерби?	580
Поиск наиболее вероятного результата НММ	582
Профильные НММ для выравнивания последовательностей	584
Как НММ связаны с выравниванием последовательностей?	584
Создание профильной НММ	587
Вероятности перехода и эмиссии профильной НММ	590

Классификация белков с помощью профильных НММ.....	593
Выравнивание белков по профильной НММ.....	593
Возвращение псевдосчетов.....	594
Проблема с молчащими состояниями.....	597
Действительно ли профильные НММ так полезны?.....	602
Обучение параметров НММ.....	604
Определение параметров НММ, когда скрытый путь известен.....	604
Обучение Витерби.....	605
Мягкие решения для определения параметров.....	607
Задача мягкого декодирования.....	607
Алгоритм «вперед–назад».....	608
Обучение Баума–Уэлча.....	611
Многоликость НММ.....	613
Эпилог. Природа – мастер, а не изобретатель.....	614
Сопутствующие материалы.....	616
Эффект Красной Королевы.....	616
Гликозилирование.....	616
Метилирование ДНК.....	616
Условная вероятность.....	618
Библиографические примечания.....	619

Глава 11. Является ли T. rex всего лишь гигантской курицей?

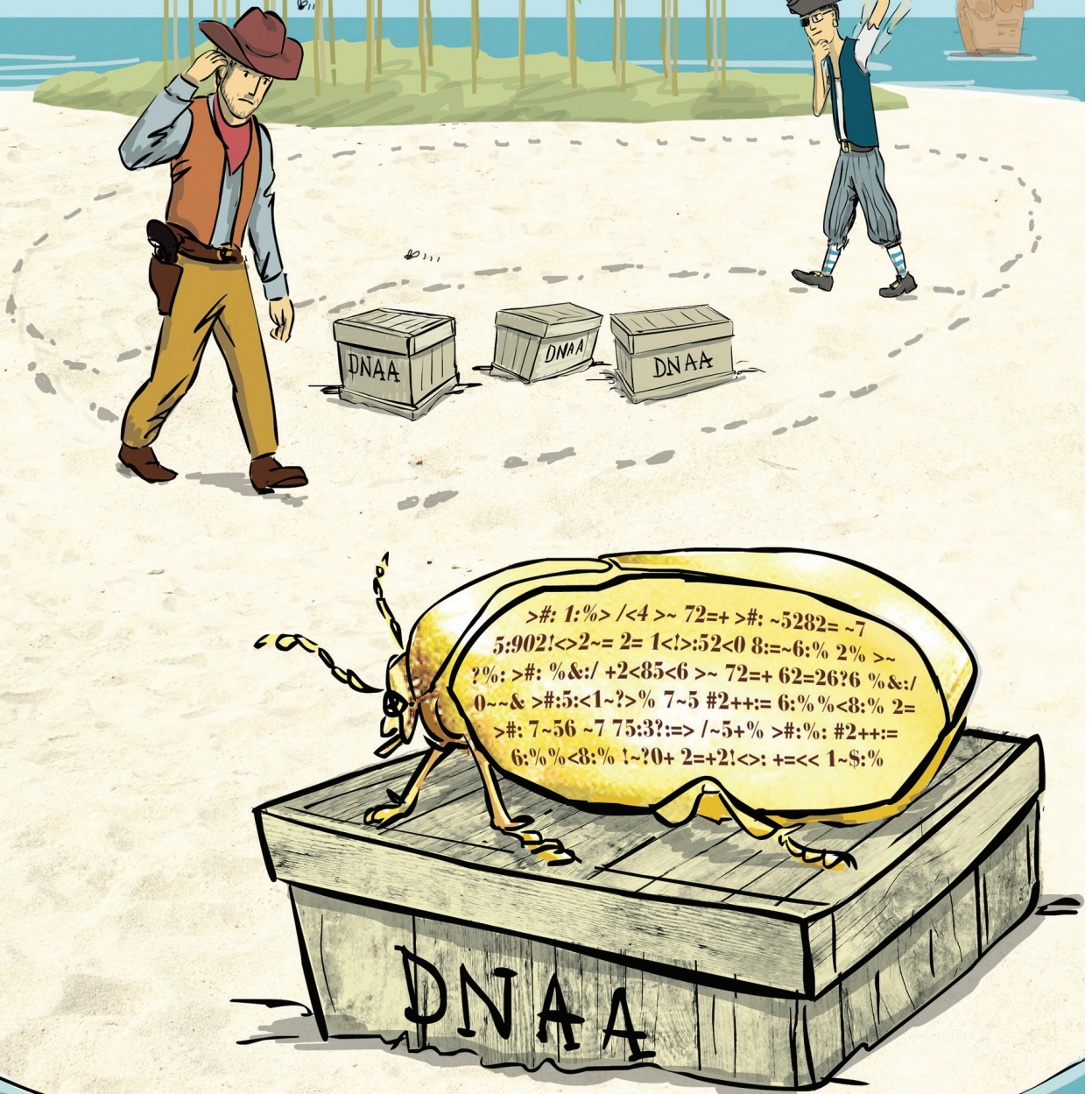
Палеонтология встречается с информатикой.....	621
Какие белки присутствуют в этом образце?.....	622
Расшифровка идеального спектра.....	623
От идеального спектра к реальному.....	626
Секвенирование пептидов.....	629
Определение пептидов по спектрам.....	629
Где находятся суффиксные пептиды?.....	631
Алгоритм секвенирования пептидов.....	634
Идентификация пептидов.....	636
Задача идентификации пептидов.....	636
Идентификация пептидов в неизвестном протеоме T. rex.....	637
Поиск совпадений пептидов со спектром.....	637
Идентификация пептидов и теорема о бесконечных обезьянах.....	639
Частота ложных открытий.....	639
Статистическая значимость пептид–спектр–совпадений.....	642
Спектральные словари.....	644
Пептиды T. rex: постороннее загрязнение или древнее сокровище?.....	648
Загадка гемоглобина.....	648
Споры о ДНК динозавров.....	650
Эпилог. От немодифицированных к модифицированным пептидам. (Часть 1).....	651

Посттрансляционные модификации	651
Поиск модификаций как задача выравнивания	652
Построение сетки Манхэттена для спектрального выравнивания	654
Эпилог. От немодифицированных к модифицированным пептидам (Часть 2)	657
Алгоритм спектрального выравнивания	657
Сопутствующие материалы	660
Предсказание генов	660
Поиск всех путей в графе	661
Задача антисимметричного пути	662
Преобразование спектров в спектральные векторы	663
Теорема о бесконечных обезьянах	667
Вероятностное пространство пептидов в словаре	668
Действительно ли динозавры являются предками птиц?	669
Библиографические примечания	670
<i>Предметный указатель</i>	671

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.

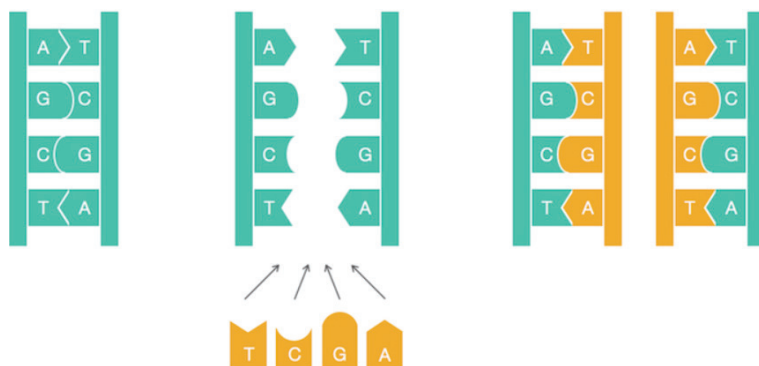


Рис. 1.1 Наивный взгляд на репликацию ДНК. Нуклеотиды аденин (А) и тимин (Т) комплементарны друг другу, как и цитозин (С) с гуанином (G). Комплементарные нуклеотиды связываются друг с другом в ДНК

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 копирующими 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
ссасаассctgagtggatgacatcaagataggtcgttgatctccttcctctcg
tactctcatgaccacggaagatgatcaagagaggatgatttcttgccatat
cgcaatgaataacttgtgacttgtgcttccaattgacatcttcagcgccatatt
gсgctggccaaggtgacggagcgggattacgaaagcatgatcatggctgttgt
tctgtttatcttgttttgactgagacttgttaggatagacggtttttcatcac
tgactagccaaagccttactctgcctgacatcgaccgtaaattgataatgaat
ttacatgcttccgсgacgatttacctcttgatcatcgatccgattgaagatct
tcaattgttaattctcttgсctcgactcatagccatgatgagctcttgatcat
gtttccttaaccctctattttttacggaagaatgatcaagctgctgctcttga
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, **АСТАТ** – удивительно частая подстрока

АСА**АСТАТ**GCАТА**АСТАТ**CGGGA**АСТАТ**ССТ.

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,

$$Count(АСА**АСТАТ**GCАТА**АСТАТ**CGGGA**АСТАТ**ССТ, **АСТАТ**) = 3.$$

Обратите внимание, что $Count(CGАТАТАТССАТАG, АТА)$ 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 $Text = GАССАТАСТG$, то $Text(4, 3) = АТА$. 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 **АСТАТ** является наиболее частым 5-мером для $Text = \text{ACAАСТАТGCАТАСТАТCGGGAАСТАТCCT}$, а **АТА** – наиболее частым 3-мером для $Text = \text{CGАТАТАТССАТАG}$.



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).

<i>Text</i>	A	C	T	G	A	C	T	C	C	C	A	C	C	C	C
<i>Count</i>	2	1	1	1	2	1	1	3	1	1	1	3	3		

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

FrequentWords(*Text*, k)

FrequentPatterns \leftarrow пустой набор

for $i \leftarrow 0$ до $|Text| - k$

Pattern $\leftarrow k$ -мер *Text*(i , k)

$Count(i) \leftarrow$ **PatternCount**(*Text*, *Pattern*)

maxCount \leftarrow максимальная величина массива *Count*

for $i \leftarrow 0$ 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, является 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 («О большое»»)**).

Если $|Text|$ и k малы, как в случае поиска *DnaA*-боксов в типичных бактериальных *ori*, тогда алгоритм со временем работы $O(|Text|^2 \cdot k)$ 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

данной таблице ниже, для текста = «ACGTTTCACGTTTTACGG» и k , равного 3. Будем сдвигать окно длиной k . *Text*, и если текущая k -мер подстрока *text* не встречается в таблице, то вы должны создать для нее новую запись. В противном случае вы бы добавили 1 к записи, соответствующей текущей k -мерной подстроке *Text*. Мы называем эту таблицу **таблицей частот** для *Text* и k .

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

Таблица, соответствующая подсчету количества вхождений каждого 3-мера в *Text* = «ACGTTTCACGTTTTACGG»

В предыдущем алгоритме **FrequentWords** мы также делаем один проход по тексту, но каждый раз, сталкиваясь в окне с k -мером, мы вызываем подпрограмму **PatternCount**, которая требует собственного прохода по всей длине *Text*. Но когда мы строим таблицу частот, то делаем один проход по тексту, и каждый раз, когда мы сталкиваемся с k -мером, просто добавляем 1 к счету k -мера.

Мы знаем, что массив длины n представляет собой упорядоченную таблицу значений, где мы обращаемся к значениям, используя целочисленные индексы от 0 до $n - 1$. Таблица частот представляет собой обобщенную версию массива, называемого **картой (map)** или **словарем (dictionary)**, в котором индексы могут быть произвольными значениями (в данном случае это строки). Точнее, индексы карты называются **ключами (keys)**.

Имея карту *dict*, мы можем получить доступ к значению, связанному с ключом *key*, используя обозначение *dict[key]*. В случае таблицы частот с именем *freq* мы можем получить доступ к значению, связанному с некоей ключевой строкой *pattern*, используя обозначение *freq[pattern]*. Следующая функция псевдокода принимает строку *text* и целое число k в качестве входных данных и выдает таблицу частот в виде сопоставления ключей строки с целочисленными значениями.

```

FrequencyTable(Text,  $k$ )
  freqMap ← пустой map
   $n$  ← |Text|
  for  $i$  ← 0 to  $n - k$ 
    Pattern ← Text( $i$ ,  $k$ )
    if freqMap[Pattern] не существует
      freqMap[Pattern] ← 1
    else
      freqMap[pattern] ← freqMap[pattern] + 1
  return freqMap

```

После того как мы построили таблицу частот для данного *Text* и k , можно найти все часто встречающиеся k -меры, если определим максимальное значение в таблице, а затем идентифицируем ключи таблицы частот, достигающие этого значения, добавляя каждый из них, который мы нашли в растущем списке. Те-

перь мы готовы написать функцию **BetterFrequentWords** для решения задачи часто встречающихся слов. Эта функция основана на функции **MaxMap**, которая принимает в качестве входных данных карту строк в целых числах и выдает максимальное значение этой карты в качестве выходных данных.

```

BetterFrequentWords(Text, k)
    FrequentPatterns ← массив строк длины 0
    freqMap ← FrequencyTable(Text, k)
    max ← MaxMap(freqMap)
    for всех строк Pattern в freqMap
        if freqMap[Pattern] = max
            добавить Pattern к FrequentPatterns
    return FrequentPatterns
    
```



Упражнение. Напишите функцию **MaxMap**. Убедитесь, что ваша функция будет работать, даже если все значения отрицательные.

Часто используемые слова в *Vibrio cholerae*

На рис. 1.3 показаны наиболее часто встречающиеся *k*-меры в области *ori* бактерии *Vibrio cholerae*.

<i>k</i>	3	4	5	6	7	8	9
count	25	12	8	8	5	4	3
<i>k</i> -меры	tga	atga	gatca tgatc	tgatca	atgatca	atgatcaa	atgatcaag cttgatcat tcttgatca ctcttgatc

Рис. 1.3 Наиболее часто встречающиеся *k*-меры в области *ori* *Vibrio cholerae* для *k* от 3 до 9, а также количество раз, когда каждый *k*-мер встречается

atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttat
 ccacaacctgagtgatgacatcaagataggtcgttgatctccttcctctcg
 tactctcatgaccacggaagatgatcaagagaggatgatttcttgccatat
 cgcaatgaatacttgtgacttgtgcttccaattgacatcttcagcgccatatt
 gcgctggccaaggtgacggagcgggattacgaaagcatgatcatggctgttgt
 tctgtttatcttgttttgactgagacttgttaggatagacggtttttcatcac
 tgactagccaaagccttactctgcctgacatcgaccgtaaattgataatgaat
 ttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaagatct
 tcaattgttaattctcttgctcgaactcatagccatgatgagctcttgatcat
 gtttccttaaccctctattttttacggaagaatgatcaagctgctgctcttga
 tcatcgtttc



ОСТАНОВИТЕСЬ и задумайтесь. Кажутся ли какие-либо цифры на рис. 1.3 неожиданно большими?

Например, 9-мер **ATGATCAAG** трижды появляется в области *ori* генома *Vibrio cholerae* – разве это удивительно?

```
atcaatgatcaacgtaagcttctaagcATGATCAAGgtgctcacacagtttat
ccacaacctgagtgatgacatcaagataggtcgttgatctccttcctctcg
tactctcatgaccacggaagATGATCAAGagaggatgatttcttgccatat
cgcaatgaatacttgtgacttgtgcttccaattgacatcttcagcgcctatt
gcgctggccaaggtgacggagcgggattacgaaagcatgatcatggctgttg
tctgtttatcttgttttgactgagacttgttaggatagacggtttttcatcac
tgactagccaaagccttactctgcctgacatcgaccgtaaattgataatgaat
ttacatgcttccgcgacgatttacctcttgatcatcgatccgattgaagatct
tcaattgttaattctcttgccctcgactcatagccatgatgagctcttgatcat
gtttccttaacctctattttttacggaagaATGATCAAGctgctgctcttga
tcatcgtttc
```

Мы выделяем наиболее часто встречающийся 9-мер вместо того, чтобы использовать какое-либо другое значение k , потому что эксперименты показали, что бактериальные *DnaA*-боксы обычно имеют длину девять нуклеотидов.

Вероятность того, что существует 9-мер, появляющийся три или более раз в случайно сгенерированной цепи ДНК длиной 500, составляет приблизительно $1/1300$ (см. **СОПУТСТВУЮЩИЕ МАТЕРИАЛЫ: Вероятность паттернов в строке**).

На самом деле в этой области есть четыре разных 9-мера, повторяющихся три или более раз: **ATGATCAAG**, **СТТГАТКАТ**, **ТСТТГАТСА** и **СТСТГАТС**. Низкая вероятность обнаружения хотя бы одного повторяющегося 9-мера в области *ori* холерного вибриона приводит нас к рабочей гипотезе о том, что один из этих четырех 9-меров может представлять собой вероятный *DnaA*-бокс, который, появляясь несколько раз в короткой области, запускает процесс репликации. Но какой?



ОСТАНОВИТЕСЬ и задумайтесь. Является ли какой-либо из четырех наиболее часто встречающихся 9-меров в *ori* холерных вибрионов «более примечательным», чем другие?

Некоторые скрытые сообщения более примечательны, чем другие

Напомним, что нуклеотиды **A** и **T** комплементарны друг другу, как и **C** с **G**. Имея одну цепь ДНК и запас «свободно плавающих» нуклеотидов, как показано на

рис. 1.1, можно представить себе синтез **комплементарной цепи на матричной цепи**. Эта модель репликации была подтверждена Мезельсоном и Шталем в 1958 году (см. **СОПУТСТВУЮЩИЕ МАТЕРИАЛЫ: Самый красивый эксперимент в биологии**). На рис. 1.4 показаны матричная цепь **TCAGCGTATCA** и комплементарная ей цепь **ACTATGCGACT**.

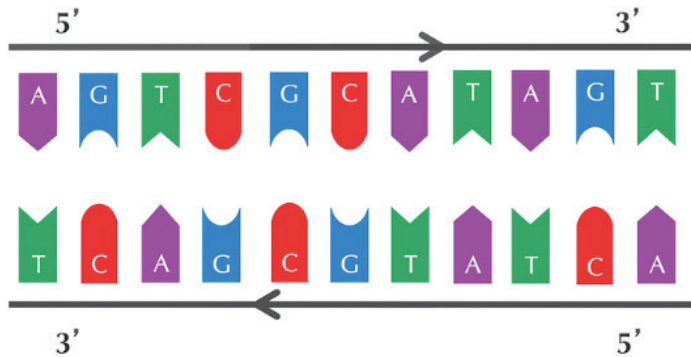


Рис. 1.4 Комплементарные цепи идут в противоположных направлениях. Каждая цепь читается в направлении 5' → 3'

В этот момент вы можете подумать, что допустили ошибку, поскольку комплементарная цепь на рис. 1.4 считывает **TCAGCGTATCA** слева направо, а не **ACTATGCGACT**. Но нет: у каждой цепи ДНК есть направление, а комплементарная цепь идет в направлении, противоположном направлению цепи матрицы, как показано стрелками на рис. 1.4. Каждая цепь читается в направлении 5' → 3' (см. **СОПУТСТВУЮЩИЕ МАТЕРИАЛЫ: Направленность цепей ДНК**, чтобы узнать, почему биологи обозначают начало и конец цепи ДНК как 5' и 3').

Возьмем нуклеотид p и обозначим его комплементарный нуклеотид как p^* . **Обратным комплементом** строки $Pattern = p_1 \dots p_n$ является строка $Pattern_{rc} = p_n^* \dots p_1^*$, образованная путем взятия комплемента каждого нуклеотида в $Pattern$ и последующего обращения полученной строки. В этой главе нам понадобится решение следующей задачи.

Задача поиска обратного комплемента: найдите комплементарное дополнение данной цепи ДНК.

Input: строка ДНК $Pattern$.

Output: $Pattern_{rc}$, реверсный комплемент $Pattern$.



ОСТАНОВИТЕСЬ и задумайтесь. Еще раз посмотрите на четыре наиболее часто встречающихся 9-мера в области *ori* холерного вибриона с рис. 1.3. Заметили ли вы что-нибудь удивительное теперь?

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagttta
 tccacaacctgagtgatgacatcaagataggtcgttgtatctccttcctctcgc
 tactctcatgaccacggaag**ATGATCAAG**agaggatgatttcttggccata
 tcgcaatgaatacttgtgacttgtgcttccaattgacatcttcagcgccatatt
 gcgctggccaaggtgacggagcgggattacgaaagcatgatcatggctggtgtt
 ctgtttatcttgttttgactgagacttgttaggatagacggttttcatcactg
 actagccaaagccttactctgcctgacatcgaccgtaaattgataatgaattta
 catgcttccgcgacgatttac**CTCTTGATCAT**cgatccgattgaagatcttc
 aattgttaattctcttgcctcgactcatagccatgatgag**CTCTTGATCAT**g
 tttccttaaccctctattttttacggaaga**ATGATCAAG**ctgctg**CTCTTG**
ATCATcgtttc

Интересно, что среди четырех наиболее часто встречающихся 9-меров в области *ori* холерного вибриона **ATGATCAAG** и **CTTGATCAT** реверсно комплементарны друг другу, что приводит к следующим шести вхождением этих строк.

atcaatgatcaacgtaagcttctaagc**ATGATCAAG**gtgctcacacagtttatc
 sacaacctgagtgatgacatcaagataggtcgttgtatctccttcctctcgcgta
 ctctcatgaccacggaag**ATGATCAAG**agaggatgatttcttggccatatacgc
 aatgaatacttgtgacttgtgcttccaattgacatcttcagcgccatattgcgc
 tggccaaggtgacggagcgggattacgaaagcatgatcatggctggtgttctgt
 ttatcttgttttgactgagacttgttaggatagacggttttcatcactgacta
 gssaaagccttactctgcctgacatcgaccgtaaattgataatgaatttacatg
 cttccgcgacgatttacct**CTTGATCAT**cgatccgattgaagatcttcaattgt
 taattctcttgcctcgactcatagccatgatgagct**CTTGATCAT**gtttcctta
 accctctattttttacggaaga**ATGATCAAG**ctgctgct**CTTGATCAT**cgtttc

Обнаружение 9-мера, который появляется шесть раз (либо сам по себе, либо как его реверсный комплемент) в цепи ДНК длиной 500 нуклеотидов, гораздо более удивительно, чем обнаружение 9-мера, который появляется три раза (как сам по себе). Это наблюдение приводит нас к рабочей гипотезе, что **ATGATCAAG** и его реверсный комплемент **CTTGATCAT** действительно представляют собой *DnaA*-боксы в *Vibrio cholerae*.

Этот вычислительный вывод имеет биологический смысл, потому что белку *DnaA*, который связывается с *DnaA*-боксом и инициирует репликацию, все равно, с какой из двух цепей он связывается. Таким образом, для наших целей и **ATGATCAAG**, и **CTTGATCAT** представляют собой *DnaA*-боксы.

Однако, прежде чем сделать вывод, что мы нашли *DnaA*-боксы холерного вибриона, внимательный биоинформатик должен проверить, есть ли другие короткие области в геноме холерного вибриона, демонстрирующие множественные точки входа **ATGATCAAG** (или **CTTGATCAT**). В конце концов, возможно, эти цепи повторяются во всем геноме *Vibrio cholerae*, а не только в области *ori*. Для этого нам необходимо решить следующую задачу.

Задача поиска вхождений *Pattern* в строку: найти все точки вхождения *Pattern* в строку.

Input: строки *Pattern* и *Genome*.

Output: все стартовые позиции в *Genome*, где *Pattern* появляется как подстрока.



Упражнение. Выдайте список начальных позиций, разделенных пробелами (в порядке возрастания), где **СТТGATCAT** появляется как подстрока в геноме холерного вибриона.

[👉 Загрузить данные 1.2](#)

Примечания.

1. Перерывы на упражнения – это возможность расширить свои знания по теме. Они необязательны, поэтому вы можете свободно продолжать чтение.
2. Мы покажем все стартовые позиции **ATGATCAAG** в этом геноме далее.

После решения задачи сопоставления с паттерном мы обнаруживаем, что **ATGATCAAG** встречается 17 раз в следующих позициях генома холерного вибриона:

116556, 149355, **151913**, **152013**, **152394**, 186189, 194276, 200076, 224527, 307692, 479770, 610980, 653338, 679985, 768828, 878903, 985368

За исключением трех точек входа **ATGATCAAG** в *ori* в начальных положениях **151913**, **152013** и **152394**, никакие другие экземпляры **ATGATCAAG** не образуют **сгустков (clumps)**, т. е. не появляются близко друг к другу в небольшой области генома (Эти сгустки (clumps) не надо путать с DNA clamps, так называемых белков скользящего зажима. – *Прим. ред.*). Вы можете убедиться, что такой же вывод делается и при поиске **СТТGATCAT**. Теперь у нас есть убедительные статистические доказательства того, что **ATGATCAAG/СТТGATCAT** может представлять собой скрытое сообщение для *DnaA* о старте репликации.



ОСТАНОВИТЕСЬ и задумайтесь. Можем ли мы заключить, что **ATGATCAAG/СТТGATCAT** также представляет собой *DnaA*-бокс в геномах других бактерий?

Взрыв скрытых сообщений

Поиск скрытых сообщений в нескольких геномах

Мы не должны делать поспешных выводов о том, что **ATGATCAAG/CTTGATCAT** является скрытым сообщением для всех бактериальных геномов, не проверив предварительно, появляется ли оно вообще в известных областях *ori* других бактерий. В конце концов, возможно, эффект скопления **ATGATCAAG/CTTGATCAT** в области *ori* холерного вибриона является просто статистической случайностью, не имеющей ничего общего с репликацией. Или, возможно, разные бактерии имеют разные *DnaA*-боксы...

Давайте проверим предполагаемую область *ori* *Thermotoga petrophila*, бактерии, которая процветает в чрезвычайно жаркой среде; ее название происходит от места ее обнаружения в воде под нефтяными резервуарами, где температура может превышать 80 °C.

```
aactctatacctcctttttgtcgaatttggtgatttatagagaaaatct
tattaactgaaactaaaatggtaggtttggtaggttttgtgtacat
tttgtagtatctgatttttaattacataccgtatattgtattaaattga
cgaacaattgcatggaattgaatataatgcaaaacaaacctaccaccaaac
tctgtattgaccatthtaggacaacttcagggtggtaggtttctgaagct
ctcatcaatagactatthtagtctttacaacaatattaccgttcagatt
caagattctacaacgctgtthtaatgggctgttcagaaaacttaccacct
aaatccagtatccaagccgatttcagagaaacctaccacttacctaccact
tacctaccaccgggtggttaagttgcagacattatthaaacctcatcag-
aagcttgthcaaaaatthcaatactcgaac cctaccacctgcgtcccctatt
atthactactactaataatagcagtataattgatctga
```

Эта область не содержит ни одного вхождения **ATGATCAAG** или **CTTGATCAT**! Таким образом, разные бактерии могут использовать разные *DnaA*-боксы в качестве скрытых сообщений для белка *DnaA*.

Применение задачи частых слов к описанной выше области *ori* показывает, что следующие шесть 9-меров появляются в этой области три или более раз:

AACCTACCA AACCTACC ACCTACCAC
CTACCACC GGTAGGTTT TGGTAGGTT

Должно происходить что-то необычное, потому что крайне маловероятно, что шесть разных 9-меров будут так часто встречаться в коротком участке случайной строки. Мы немного схитрим и проконсультируемся с **Ori-Finder**, патентованным программным инструментом для поиска точек начала репликации в цепях ДНК. Это программное обеспечение выбирает **CTACCACC** (вместе с его обратным комплементом **GGTGGTAGG**) в качестве рабочей гипотезы для *DnaA*-боксы у *Thermotoga petrophila*. Вместе эти два комплементарных 9-мера появляются пять раз в точке начала репликации:

```

aactaaaatggtaggtttGGTGGTAGGttttgtgtacattttgtagtadc
tgatttttaattacataaccgtatattgtattaaattgacgaacaattgc
atggaattgaatataatgcaaaacaaaCCTACCACCaaactctgtattga
ccatttttaggacaacttcagGGTGGTAGGtttctgaagctctcatcaata
gactatttttagtctttacaacaatattaccgttcagattcaagattcta
caacgctgttttaatggcggttgcaaaaaacttaccacctaataatccagt
atccaagccgatttcagagaaaactaccacttacctaccacttaCCTACCA
CCcgggtggaagtggcagacattattaaaaacctcatcagaagcttgttc
aaaaatttcaatactcgaaaCCTACCACCtgcgtcccctattattacta-
ctactaataatagcagtataattgatctga

```

Задача поиска сгустков

Теперь представьте, что вы пытаетесь найти *ori* в недавно секвенированном бактериальном геноме. Поиск сгустков **ATGATCAAG/CTTGATCAT** или **CCTACCACC/GGTGGTAGG** вряд ли поможет, так как этот новый геном может использовать совершенно другое скрытое сообщение. Прежде чем мы потеряем всякую надежду, давайте изменим наше направление поиска: вместо того, чтобы искать сгустки определенного k -мера, давайте попытаемся найти *каждый* k -мер, который образует сгусток в геноме. Будем надеяться, что расположение этих сгустков прольет свет на местонахождение *ori*.

Наш план состоит в том, чтобы перемещать окно фиксированной длины L вдоль генома в поисках области, где k -мер появляется несколько раз подряд. Значение параметра $L = 500$ отражает типичную длину *ori* в бактериальных геномах.

Мы определили k -мер как сгусток (clump), если он появляется много раз в пределах короткого интервала генома. Более формально для заданных чисел L и t k -мер *Pattern* образует **(L, t)-clump** внутри (более длинной) строки *Genome*, если существует интервал *Genome* длины L , в котором этот k -мер появляется по крайней мере t раз. (Это определение предполагает, что k -мер полностью укладывается в интервал.) Например, **TGCA** образует (25, 3)-clump в следующем *Genome*:

```
gatcagcataaggggtccCTGCAATGCAATGACAAGCCCTGCAAGTtgttttac
```

Из наших предыдущих примеров областей *ori* **ATGATCAAG** образует (500, 3)-clump в геноме *Vibrio cholerae*, а **CCTACCACC** образует (500, 3)-clump в геноме *Thermotoga petrophila*. Теперь мы готовы сформулировать следующую задачу.

Задача поиска сгустков: найдите *Patterns*, образующие сгустки в строке.

Input: строка *Genome* и целые числа k, L и t .

Output: все различные k -меры, образующие (L, t)-clump в *Genome*.



ЗАРЯДНАЯ СТАНЦИЯ: Решение задачи поиска сгустков.

Вы можете решить задачу поиска сгустков, просто применив свой алгоритм для задачи частых слов к каждому окну длины L в *Genome*. Однако, если ваш алгоритм решения задачи часто встречающихся слов не очень эффективен, такой подход может оказаться нецелесообразным. Например, вспомните, что **FrequentWords** имеет время выполнения $O(L^2 \cdot k)$. Применение этого алгоритма к каждому окну длины L в *Genome* приведет к продолжительности расчета $O(L^2 \cdot k \cdot |\text{Genome}|)$. Более того, даже если мы используем более быстрый алгоритм для поиска часто встречающихся слов, время выполнения останется большим, даже когда мы пытаемся проанализировать небольшой бактериальный геном, не говоря уже о человеческом геноме.

Задача поиска сгустков сложнее, чем та, с которой мы сталкивались до сих пор, и написать функцию, решающую ее с нуля, будет сложно. Однако именно здесь модульность в написании программ так полезна. У нас уже есть функция **FrequencyTable**, которая создаст таблицу частот для данного окна из строки длины L . Если мы применим ее к данному окну, то нам просто нужно проверить, есть ли в таблице какие-либо ключи строк, значения которых по крайней мере равно t . Мы будем добавлять любые такие ключи, которые мы еще не видели в каком-либо другом текстовом окне, к растущему списку строк. В конце концов этот список строк будет содержать (L, t) -сгустки текста. Это обрабатывается следующей функцией **FindClumps**.

```

FindClumps(Text, k, L, t)
  Patterns ← набор строк длиной 0
  n ← |Text|
  for каждого целого  $i$  между 0 and  $n - L$ 
    Window ← Text( $i, L$ )
    freqMap ← FrequencyTable(Window, k)
    for каждого ключа  $s$  в freqMap
      if freqMap[ $s$ ]  $\geq t$ 
        добавить  $s$  к Patterns
  удалить дубликаты из Patterns
  return Patterns

```

Давайте поищем сгустки в геноме *Escherichia coli* (*E. coli*), рабочей лошадке бактериальной геномики. Мы находим сотни различных 9-меров, образующих (500, 3)-сгустков в геноме *E. coli*, и совершенно неясно, какой из этих 9-меров может представлять собой *DnaA*-бокс в области *ori* бактерии.



ОСТАНОВИТЕСЬ и задумайтесь. Должны ли мы сдаться? Если нет, что бы вы сделали сейчас?

В этот момент неопытный исследователь мог бы сдаться, так как оказалось, что у нас недостаточно информации, чтобы определить местонахождение *ori* в *E. coli*. Но бесстрашный биоинформатик-ветеран попытался бы узнать больше о деталях репликации в надежде, что они обеспечат новые алгоритмические идеи для поиска *ori*.

Самое простое объяснение процесса репликации ДНК

Теперь мы готовы обсудить процесс репликации более подробно. Как показано на рис. 1.5 (вверху), две комплементарные цепи ДНК, идущие в противоположных направлениях вокруг кольцевой хромосомы, разделяются, начиная с локации *ori*. Когда цепи раскручиваются, они создают две **репликационные вилки**, которые расширяются в обоих направлениях вдоль хромосомы до тех пор, пока цепи полностью не разделятся на **конце репликации** («replication terminus», или «остановка репликации», обозначается *ter*). Конец репликации расположен примерно напротив *ori* в хромосоме.

Важно знать о репликации то, что ДНК-полимераза не ждет, пока две родительские цепи полностью разделятся, прежде чем инициировать репликацию; вместо этого она начинает процесс копирования, пока цепи еще распутываются. Таким образом, всего четыре ДНК-полимеразы, каждая из которых отвечает за одну комплементарную часть цепи, могут начинать с *ori* и реплицировать всю хромосому. Чтобы начать репликацию, ДНК-полимеразе нужен **праймер** – короткий комплементарный сегмент (показан красным на рис. 1.5), который связывается с исходной цепью и запускает ДНК-полимеразу. После того как цепи начинают разделяться, каждая из четырех ДНК-полимераз начинает репликацию, добавляя нуклеотиды, начиная с праймера и двигаясь вдоль хромосомы от *ori* к *ter* либо по часовой стрелке, либо против часовой стрелки. Когда все четыре ДНК-полимеразы достигли *ter*, ДНК хромосомы будет полностью реплицирована, что приведет к образованию двух пар комплементарных цепей (рис. 1.5 (внизу)), и клетка готова к делению.

Пока вы читали описание выше, профессора биологии писали петицию, чтобы нас уволили и отправили обратно в «Биологию 101» (Сериял NBC про студентов-биологов. – Прим. ред.). И они были бы правы, потому что наше изложение страдает серьезным недостатком; но мы намеренно описали процесс репликации именно таким образом, чтобы вы могли лучше понять то, что мы собираемся до вас донести дальше.

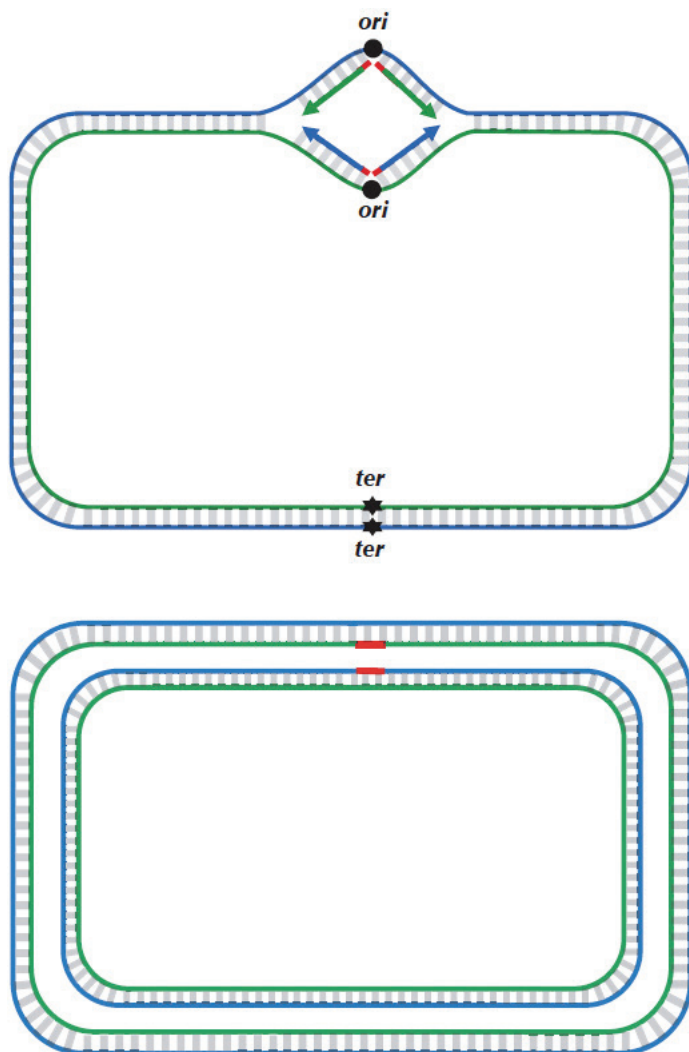


Рис. 1.5 (Вверху) Четыре воображаемых ДНК-полимеразы в процессе репликации хромосомы по мере того, как вилки репликации передвигаются от *ori* до *ter*. Синяя цепь направлена по часовой стрелке, а зеленая – против часовой стрелки. (Внизу) Репликация завершена

Проблема с нашим нынешним описанием заключается в том, что оно предполагает, что ДНК-полимеразы могут копировать ДНК в *любом* направлении вдоль цепи ДНК (т. е. как в направлении $5' \rightarrow 3'$, так и в направлении $3' \rightarrow 5'$). Однако природа еще не наделила ДНК-полимеразы такой способностью, поскольку они являются **однаправленными**, что означает, что они могут передвигаться по

матричной цепи ДНК только в направлении $3' \rightarrow 5'$. Обратите внимание, что это направление, противоположное направлению $5' \rightarrow 3'$ цепи ДНК.



ОСТАНОВИТЕСЬ и задумайтесь. Если бы вы использовали одностороннюю ДНК-полимеразу, как бы вы реплицировали ДНК? Сколько ДНК-полимераз потребовалось бы для выполнения такой задачи?

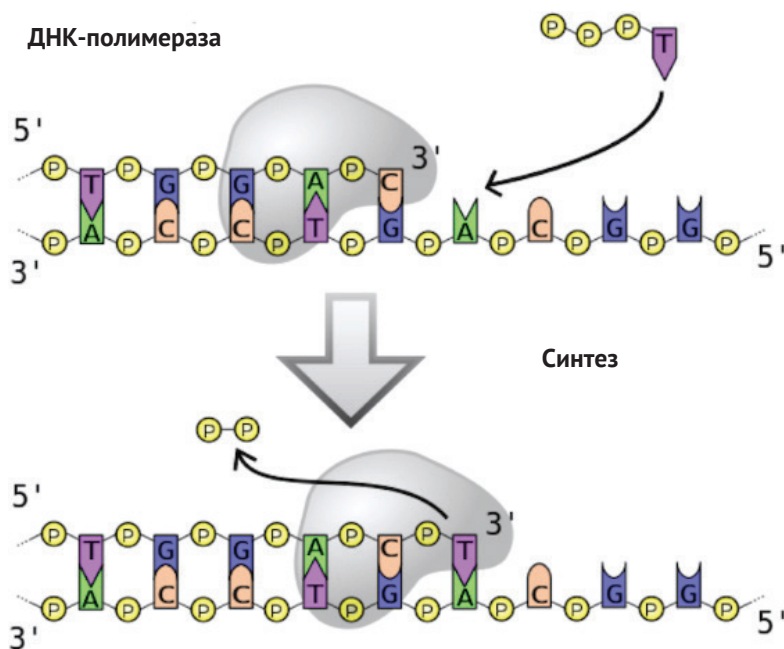


Рис. 1.6 ДНК-полимераза копирует цепь матрицы в направлении $3' \rightarrow 5'$. Обратите внимание, что она создает дочернюю цепь в направлении $5' \rightarrow 3'$

Однонаправленность ДНК-полимеразы требует серьезного пересмотра нашей наивной модели репликации. Представьте, что вы решили пройти по ДНК от *ori* до *ter*. Существуют четыре разные полуцепи родительской ДНК, соединяющие *ori* с *ter*, как показано на рис. 1.7. Две из этих полуцепей проходят от *ori* в направлении $5' \rightarrow 3'$ и поэтому называются **прямыми полуцепями** (представлены тонкими синими и зелеными линиями на рис. 1.7). Две другие полуцепи проходят от *ori* к *ter* в направлении $3' \rightarrow 5'$ и поэтому называются **обратными полуцепями** (обозначены толстыми синими и зелеными линиями на рис. 1.7).

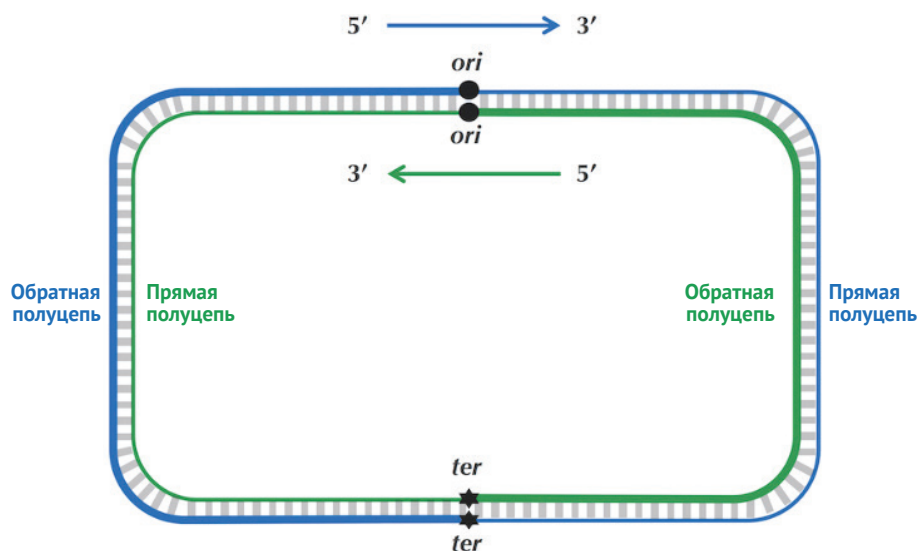


Рис. 1.7 Комплементарные цепи ДНК с прямой и обратной полуцепями, показанными тонкими и толстыми линиями соответственно

Асимметрия репликации

В то время как биологи будут чувствовать себя как дома со следующим описанием репликации ДНК, ученые-компьютерщики могут найти его перегруженным новыми терминами. Если это кажется слишком сложным с биологической точки зрения, не стесняйтесь пролистывать этот раздел, если вы верите нам, что процесс репликации **асимметричен**, т. е. что прямые и обратные полуцепи при репликации имеют очень разные судьбы.

Поскольку ДНК-полимераза может двигаться только в обратном ($3' \rightarrow 5'$) направлении, она может безостановочно копировать нуклеотиды от *ori* до *ter* вдоль обратных полуцепей. Однако репликация на прямых полуцепях сильно отличается, потому что ДНК-полимераза не может двигаться в прямом ($5' \rightarrow 3'$) направлении; на этих полуцепях ДНК-полимераза должна реплицироваться в *обратном направлении*, к *ori*. Взгляните на рис. 1.8, чтобы понять, почему это так.

На прямой полуцепи, чтобы реплицировать ДНК, ДНК-полимераза должна ждать, пока репликационная вилка немного откроется (примерно на 2000 нуклеотидов), пока на *конце* репликационной вилки не сформируется новый праймер; после этого ДНК-полимераза начинает реплицировать небольшой фрагмент ДНК, начиная с этого праймера и двигаясь *назад* в направлении к *ori*. Когда две ДНК-полимеразы на передних полуцепях достигают *ori*, возникает ситуация, показанная на рис. 1.9 ниже. Обратите внимание на разницу между этим рисунком и рис. 1.5.

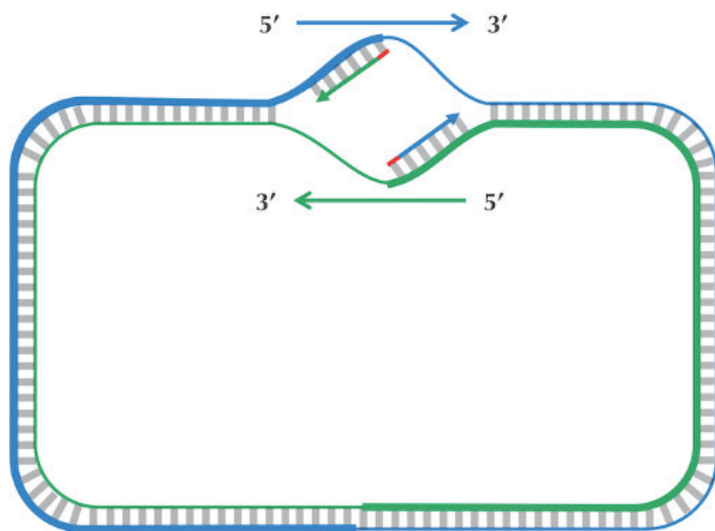


Рис. 1.8 Репликация начинается с *ori* (праймеры показаны красным), с синтеза фрагментов на обратных полуцепях (показаны толстыми линиями). ДНК-полимераза должна дождаться, пока репликационная вилка откроется на некоторое (небольшое) расстояние, прежде чем она начнет копировать передние полуцепи (показаны тонкими линиями) обратно к *ori*

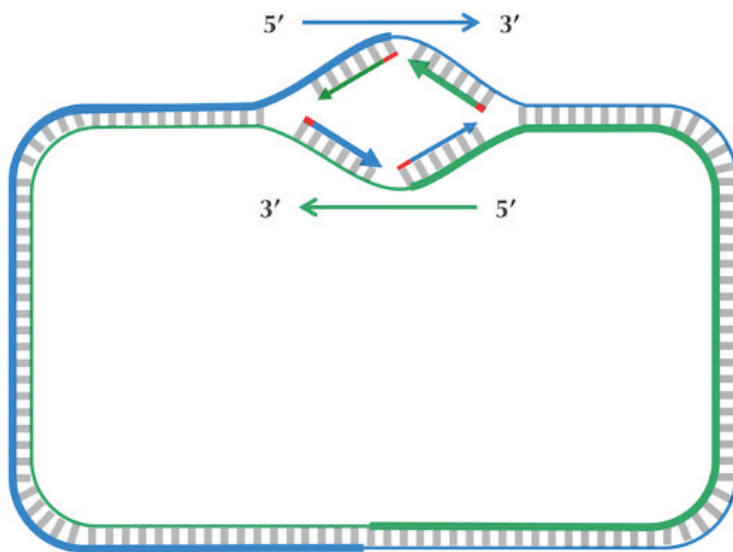


Рис. 1.9 Теперь дочерние фрагменты синтезируются (с некоторой задержкой) на прямых полуцепях (показаны тонкими линиями)

После этого репликация на каждой обратной полуцепи продолжается непрерывно; однако у ДНК-полимеразы на прямой полуцепи нет другого выбора, кроме как снова ждать, пока репликационная вилка не откроет еще 2000 нуклеотидов или около того. Затем требуется новый праймер, чтобы начать синтез другого фрагмента в обратном направлении. В целом репликация на прямой полуцепи требует периодической остановки и перезапуска, что приводит к синтезу коротких **фрагментов Оказаки**, комплементарных интервалам на прямой полуцепи. Вы можете увидеть, как формируются эти фрагменты, на рис. 1.10 (вверху).

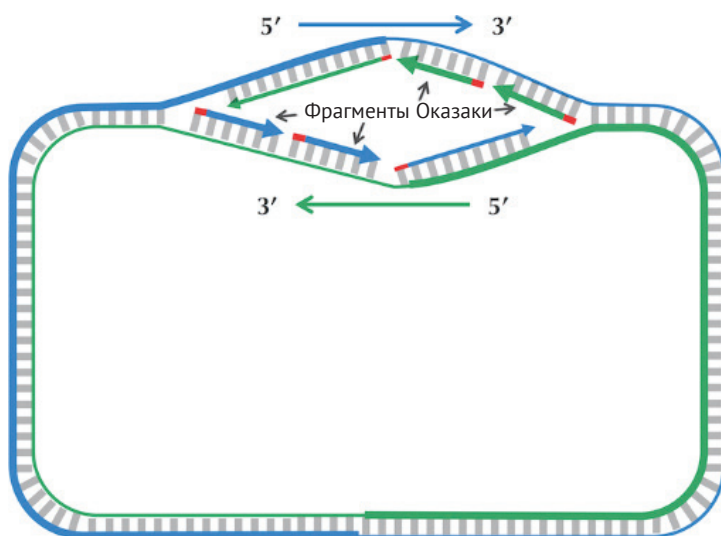


Рис. 1.10 Вилка репликации продолжает расти. Для каждой из обратных полуцепей (показанных жирными линиями) требуется только один праймер, в то время как прямые полуцепи (показаны тонкими линиями) требуют нескольких праймеров для синтеза фрагментов Оказаки. Два из этих праймеров показаны красным на каждой прямой полуцепи

Когда репликационная вилка достигает *ter*, процесс репликации почти завершен, но между несвязанными фрагментами Оказаки все еще остаются промежутки, как показано ниже.

Наконец, последовательные фрагменты Оказаки сшиваются ферментом, называемым **ДНК-лигазой**, в результате чего образуются две интактные дочерние хромосомы, каждая из которых состоит из одной родительской цепи и одной вновь синтезированной дочерней цепи, как показано на рис. 1.12.

На самом деле ДНК-лигаза не ждет, пока все фрагменты Оказаки будут реплицированы, чтобы начать сшивать их вместе.

Биологи называют обратную полуцепь **ведущей**, поскольку всего одна ДНК-полимераза проходит эту полуцепь без остановок, а переднюю полуцепь они

называют **отстающей**, поскольку над ней работают многие ДНК-полимеразы, которые многократно останавливают и вновь начинают репликацию.

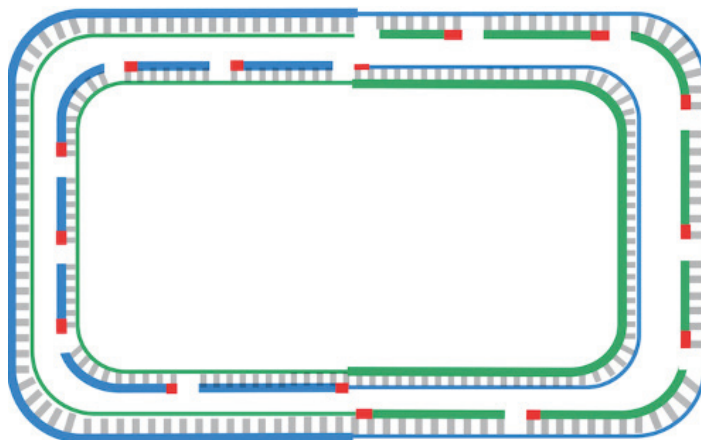


Рис. 1.11 Продолжение процесса репликации

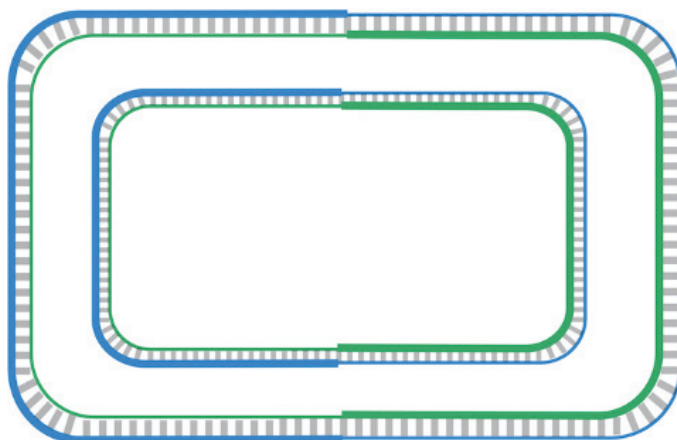


Рис. 1.12 Завершение процесса репликации

Если вы не понимаете различий между ведущими и отстающими полужцепочками, вы не одиноки – мы и легионы студентов-биологов тоже в замешательстве. Путаница усугубляется тем фактом, что в разных учебниках используется разная терминология в зависимости от того, намерены ли авторы ссылаться на ведущую полужцепь матрицы, с которой синтезируется дочерняя цепь, или на ту, которая синтезируется из (отстающей) полужцепи матрицы. Надеюсь, вы понимаете, почему мы выбрали термины «обратная» и «прямая» полужцепь в попытке смягчить некоторую путаницу.

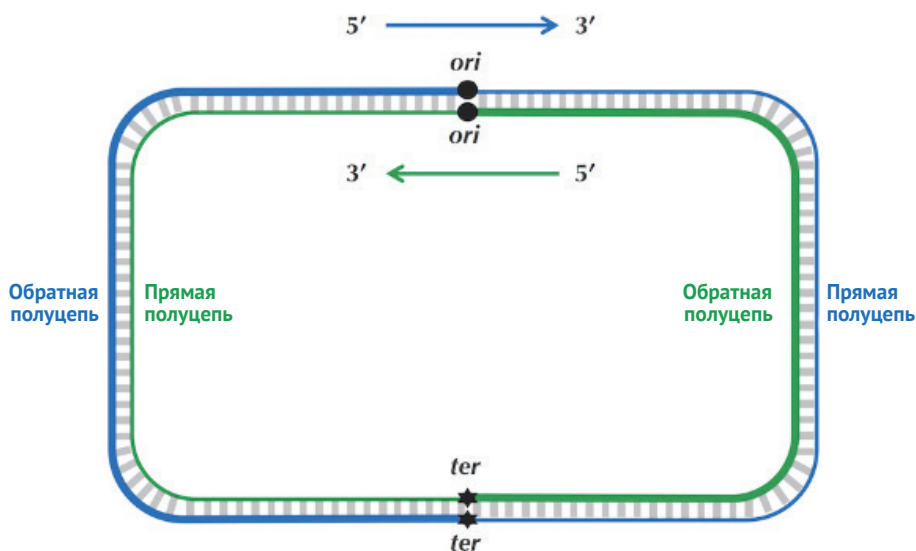


Рис. 1.13 Прямая и обратная полуцепи

Специфическая статистика прямой и обратной полуцепей

Неизвестный биологический феномен или статистическая случайность?

Рисунок 1.14 (вверху слева) показывает удивительную закономерность. Мы разделили геном *E. coli* на 46 фрагментов одинакового размера примерно по 100 000 нуклеотидов, начиная с экспериментально подтвержденного конца репликации, а затем вычислили частоту цитозина в каждом окне. Первые 23 фрагмента (начиная с *ter*) представляют обратную полуцепь, а последние 23 фрагмента (начиная с *ori*) – прямую полуцепь (рис. 1.7). Большинство фрагментов обратной полуцепи имеет высокую частоту цитозина (выше 25 %), тогда как большинство фрагментов прямой полуцепи – низкую цитозиновую частоту (ниже 25 %). Напротив, как показано на рис. 1.14 (вверху справа), большинство фрагментов на обратной полуцепи имеет низкую частоту гуанина (ниже 25 %), тогда как большинство фрагментов на прямой полуцепи – высокую (выше 25 %).

На рис. 1.14 (внизу слева) показаны различия частот G и C в каждом фрагменте генома и представлена еще более поразительная визуализация своеобразной статистики частот нуклеотидов на обратной и прямой полуцепях. Даже если мы предположим, что нам заранее неизвестно местонахождение *ori*, картина все равно проявляется, когда мы начинаем с произвольной позиции генома *E. coli*, как показано на рис. 1.14 (внизу справа).

Если закономерность, которую мы нашли на рис. 1.14, не является статистической случайностью, то мы нашли подсказку о том, как найти *ori*: можно просто пройтись по геному и проверить, где разница между частотой гуанина и цитозина переключается с отрицательной на положительную! Но с какой стати такой простой тест позволяет нам найти источник репликации бактерии?

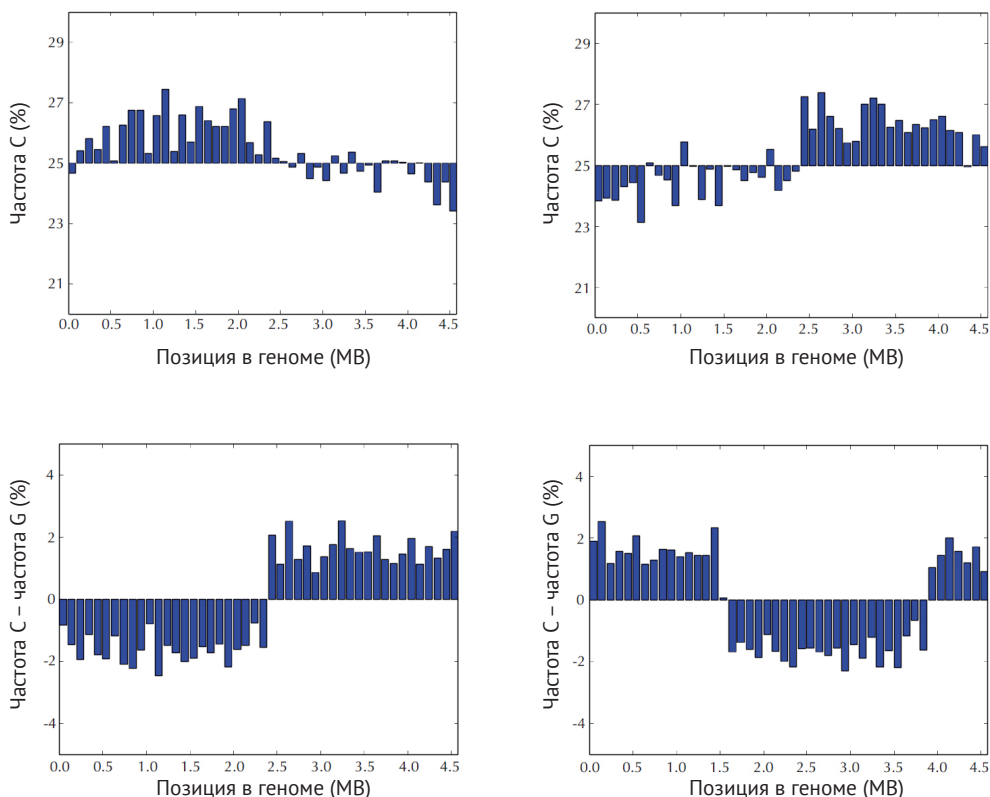


Рис. 1.14 (Вверху слева) Частота цитозина в каждом из 46 непересекающихся фрагментов равной длины (примерно 100 000 нуклеотидов каждый), покрывающих геном *E. coli*. Сайт *ter* находится в положении 0, тогда как сайт *ori* расположен почти напротив *ter* на кольцевой хромосоме *E. coli* примерно в 2,3 млн нуклеотидов от *ter*. Обратная полуплеть охватывает первую половину гистограммы (от 0 до *ori*), тогда как прямая полуплеть – вторую половину гистограммы (начиная с *ori*). (Вверху справа) Частота гуанина в тех же 46 фрагментах генома *E. coli*. (Внизу слева) Разница между частотами встречаемости гуанина и цитозина в 46 фрагментах генома *E. coli* при условии, что геном начинается с экспериментально подтвержденного *ter* *E. coli*. (Внизу справа) Разница между частотами встречаемости гуанина и цитозина в 46 фрагментах генома *E. coli* при условии, что геном начинается в произвольно выбранном месте