**Министерство образования и науки Российской Федерации**

УДК: 004.021
ГРНТИ: 20.01.01, 34.05.25
Инв. №: 310276

| **УТВЕРЖДЕНО:** |
|---|
| **Исполнитель:**<br>**федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»** |
| **Ректор ФГБОУ ВПО «СПбНИУ ИТМО»**<br><br>**_____/В.Н. Васильев/**<br>**М.П.** |

# НАУЧНО-ТЕХНИЧЕСКИЙ ОТЧЕТ

**о выполнении второго этапа Государственного контракта**
**№ 16.740.11.0495 от 16 мая 2011 г.**

| |
|---|
| **Исполнитель: федеральное государственное бюджетное образова-тельное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»** |
| **Программа (мероприятие): Федеральная целевая программа «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг., в рамках реализации мероприятия № 1.2.1. Проведение научных исследований научными группами под руководством докторов наук.** |
| **Проект: Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям** |
| **Руководитель проекта:**<br>**_____/Шалыто Анатолий Абрамович**<br>(подпись) |

**Санкт-Петербург**
**2011 г.**

# СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

**по Государственному контракту 16.740.11.0495 от 16 мая 2011 на выполнение поисковых научно-исследовательских работ для государственных нужд**

Организация-Исполнитель: федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики».

Руководитель темы:

| | | |
|---|---|---|
| доктор технических наук, профессор | _____ | Шалыто А. А. |

Исполнители темы:

| | | |
|---|---|---|
| Кандидат технических наук, без ученого звания | _____ | Корнеев Г. А. |
| Кандидат технических наук, без ученого звания | _____ | Князев Е. Г. |
| без ученой степени, без ученого звания | _____ | Царев Ф. Н. |
| без ученой степени, без ученого звания | _____ | Федотов П. В. |
| без ученой степени, без ученого звания | _____ | Александров А. В. |
| без ученой степени, без ученого звания | _____ | Сергушичев А. А. |

без ученой степени, без ученого звания _____ Казаков С. В.

## РЕФЕРАТ

Отчет 64 с., 2 гл., 5 рис., 3 табл., 12 источн., 4 прил.

Исправление ошибок секвенирования, восстановление фрагментов, сборка генома, парные чтения

В отчете представлены результаты исследований, выполненных по 2 этапу Государственного контракта № 16.740.11.0495 «Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям» (шифр «2011-1.2.1-201-007») от 16 мая 2011 по направлению «Проведение научных исследований научными группами под руководством докторов наук в следующих областях: – биокаталитические, биосинтетические и биосенсорные технологии; – биомедицинские и ветеринарные технологии жизнеобеспечения и защиты человека и животных; – геномные и постгеномные технологии создания лекарственных средств; – клеточные технологии; – биоинженерия; – биоинформационные технологии» в рамках мероприятия 1.2.1. «Проведение научных исследований научными группами под руководством докторов наук», мероприятия 1.2. «Проведение научных исследований научными группами под руководством докторов наук и кандидатов наук», направления 1. «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы.

Цель работы. Общей целью выполнения НИР в рамках мероприятия является обеспечение достижения научных результатов мирового уровня, подготовку и закрепление в сфере науки и образования научных и научно-педагогических кадров, формирование эффективных и жизнеспособных научных коллективов.

Целью выполнения НИР является разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям.

Целями второго этапа являются:

1. Разработка алгоритма исправления ошибок в данных секвенирования.
2. Программная реализация алгоритма исправления ошибок в данных секвенирования.

При выполнении второго этапа ПНИР был использован следующий инструментарий:

1. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036527.
2. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036528.
3. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036529.
4. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036530.
5. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036531.
6. Компьютер *Aquarius Elt E50 S46*. Инв. номер. 110104.7.0036532.

При подготовке научно-технического отчета были использованы следующие нормативные документы:

- Постановление Правительства Российской Федерации от 4 мая 2005 г. № 284 «О государственном учете результатов научно-исследовательских, опытно-конструкторских и технологических работ гражданского назначения»;
- Гражданский кодекс РФ;
- ГОСТ 7.32-2001 «Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления»;

• ГОСТ 15.101.98 «Система разработки и постановки продукции на производство. Порядок выполнения научно-исследовательских работ»;

• ГОСТ Р 15.011-96 «Система разработки и постановки продукции на производство. Патентные исследования. Содержание и порядок проведения».

В ходе выполнения второго этапа были получены следующие результаты:

• разработан алгоритм исправления ошибок в данных секвенирования;

• осуществлена программная реализация алгоритма исправления ошибок в данных секвенирования;

Многие современные задачи биологии и медицины требуют знания генома живых организмов, который состоит из нескольких нуклеотидных последовательностей ДНК. В связи с этим возникает необходимость в дешевом и быстром методе секвенирования, то есть определения последовательности нуклеотидов в образце ДНК.

Устройства, осуществляющие чтение нуклеотидных последовательностей, допускают при чтении ошибки, которые необходимо исправить. Для текущего этапа выполнения НИР была поставлена задача разработки алгоритма исправления ошибок в чтениях нуклеотидной последовательности, который может использоваться как часть процесса секвенирования генома. Разработанный в соответствии с поставленной задачей алгоритм основан на частотном анализе встречаемости *k*-меров.

## ОГЛАВЛЕНИЕ

# ВВЕДЕНИЕ

В настоящем отчете излагаются результаты выполнения *поисковых научно-исследовательских работ по теме «Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям»*, выполняемых в рамках государственного контракта, заключенного между Министерством образования и науки Российской Федерации и федеральным государственным бюджетным образовательным учреждением высшего профессионального образования «Санкт-Петербургским национальным исследовательским университетом информационных технологий, механики и оптики» в соответствии с решением Конкурсной комиссии Министерства образования и науки Российской Федерации № 1 (протокол от 26.04.2011 г. № 3/01731000037110000032) по лоту шифр «2011-1.2.1-201-007» «Проведение научных исследований научными группами под руководством докторов наук в следующих областях: – биокаталитические, биосинтетические и биосенсорные технологии; – биомедицинские и ветеринарные технологии жизнеобеспечения и защиты человека и животных; – геномные и постгеномные технологии создания лекарственных средств; - клеточные технологии; – биоинженерия; - биоинформационные технологии» в рамках федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009-2013 годы».

Целями настоящего этапа являются:

1. Разработка алгоритма исправления ошибок в данных секвенирования.
2. Программная реализация алгоритма исправления ошибок в данных секвенирования.

Отчет имеет следующую структуру. В первой главе приводится описание алгоритма исправления ошибок в данных секвенирования и пример использования данного алгоритма на тестовых данных.

Во второй главе приводится описание программной реализации разработанного алгоритма исправления ошибок. Также вторая глава содержит результаты работы разработанного программного средства на примере исправления ошибок в геноме. При этом несмотря на то, что тестовый геном является синтетическим, он представляет собой смесь геномов различных живых организмов. Таким образом, экспериментальные данные являются близкими к реальным.

Каждая из глав снабжена выводами, кратко резюмирующими содержание главы. В заключении дается общая оценка работ по этапу.

Изучение генома человека и других живых существ имеет важное прикладное значение. На основании результатов сборки генома конкретного человека возможна реализация индивидуальной медицины – определения предрасположенности к различным болезням, создание индивидуальных лекарств и т. д. Кроме этого на основе результатов исследования геномов растений и животных с использованием методов биоинженерии могут быть выведены новые их виды, обладающие определенными свойствами.

Задача разработки методов сборки геномных последовательностей является среди всех задач биоинформатики, в определенном смысле, центральной. Это объясняется тем, что без ее решения нельзя приступить к детальному изучению генома живого существа и его анализу с применением других алгоритмов биоинформатики.

Сложность задачи сборки геномной последовательности обусловлена следующими факторами:

- большой объем входных данных, который составляет порядка десятков и сотен гигабайт;
- сложность структуры генома – наличие в нем повторов и полиморфизмов;
- наличие ошибок в исходных данных – чтениях, полученных с устройств-секвенаторов.

Для решения указанных проблем сборку геномной последовательности разбивают на три этапа:

1. Исправление ошибок в исходных данных – чтениях геномной последовательности;
2. Сборка контигов – достаточно длинных непрерывных фрагментов искомой геномной последовательности;
3. Построение скэффолдов – наборов контигов, для которых с большой степенью уверенности определено их взаимное расположение в геномной последовательности.

В данной работе предлагается алгоритм, позволяющий осуществить первый из указанных этапов. Решения для следующих этапов сборки геномной последовательности будут изложены при выполнении следующих этапов работ по контракту.

# 1. РАЗРАБОТКА АЛГОРИТМА ИСПРАВЛЕНИЯ ОШИБОК В ДАННЫХ СЕКВЕНИРОВАНИЯ

В настоящем разделе приводятся описание алгоритма исправления ошибок в данных секвенирования.

Предлагаемый в данной работе алгоритм основан на частотном анализе содержащихся в чтениях *k*-меров и не использует графа де Брюина [1 – 3]. Для эффективного исправления ошибок необходимо, чтобы каждая позиция генома была прочитана несколько раз, так как это единственный способ отличить правильно прочитанный нуклеотид от прочитанного неверно. Это, ввиду невысокой вероятности ошибки [4, 5], дает основания полагать, что на каждой позиции нуклеотид, считанный наибольшее число раз, является верным. На практике используются наборы чтений, покрывающие геном несколько десятков раз [6]. Важно отметить, что не только отдельные позиции всего генома были прочитаны несколько десятков раз, но и небольшие его подстроки (не длиннее самих чтений) встречаются в чтениях несколько раз, причем, чем длиннее подстрока, тем меньше шансов, что несколько различных чтений ее содержат. Последнее соображение вытекает не только из увеличения вероятности попадания чтения на конкретную подстроку при увеличении ее длины, но и из факта наличия в чтениях ошибок и небольшой вероятности того, что одна и та же ошибка была допущена в одной и той же позиции генома более одного раза (если считать, что ошибочные прочтения различных позиций генома являются независимыми событиями, а средняя вероятность ошибки равна *p*, то вероятностный вес конкретной подстроки длины *k*, содержащей *n* ошибок, составляет $C_k^n (1-p)^{k-n} p^n \frac{1}{3^n}$, что при $k = 30,\ n = 1,\ p = 0.001$ равно примерно $0.009$).

## 1.1. Идея алгоритма

Для каждого *k*-мера, присутствующего в чтениях (прямых или обратно-комплементарных [7]) хотя бы раз, подсчитаем, сколько раз он встречается в чтениях. На основании этой статистики все *k*-меры можно разделить на две группы – «надежные» *k*-меры и «подозрительные» [8]. «Надежные» *k*-меры – это те, которые встречаются в чтениях достаточно большое число раз: не меньше значения некоторого порога *t*, которое является параметром алгоритма. С «надежными» *k*-мерами делать ничего не нужно, предполагается, что в них ошибок нет. «Подозрительные» же *k*-меры вызваны либо плохим покрытием тех частей генома, откуда они были прочитаны, либо, что гораздо более вероятно, наличием в них одной или нескольких ошибок, которые надлежит исправить.

После выделения «подозрительных» *k*-меров для каждого из них необходимо решить, в какой именно позиции могла быть совершена ошибка. Для этого предлагается перебрать все позиции *k*-мера (их ровно *k*) и все возможные нуклеотиды, попробовать заменить имеющийся нуклеотид на перебираемый и проанализировать получившийся *k*-мер. Если новый *k*-мер попадает в группу «надежных», значит, возможно, рассматриваемый *k*-мер является результатом его ошибочного прочтения. Если в течение перебора был найден только один «надежный» *k*-мер, получающийся из «подозрительного» путем замены одного нуклеотида на другой, полагается, что данный «подозрительный» *k*-мер исправлен, а соответствующее исправление запоминается. Если таких *k*-меров несколько, неясно, какое из исправлений запоминать, поэтому в таких случаях «подозрительные» *k*-меры не исправляются. И, наконец, если не было найдено ни одного способа исправить «подозрительный» *k*-мер, полагается, что в нем было совершено больше одной ошибки. После этого запускается аналогичная процедура, но с исправлением не одного, а сразу двух нуклеотидов. Аналогичным образом можно пытаться изменить не только пары, но и тройки, а также кортежи из большего числа нуклеотидов, однако данное обобщение ощутимо сказывается на быстродействии алгоритма.

## 1.2. Выбор длины *k*-мера

К выбору величины $k$ следует подходить достаточно серьезно, так как этот параметр сильно влияет на работу алгоритма. При выборе значения этого параметра следует учитывать следующие простые, но важные соображения:

- Величина $k$ должна быть значительно меньше длины чтений. Если длина $k$-мера будет сравнима с длиной чтения, то большинство $k$-меров будет встречаться в чтениях один раз, что не даст никакой информации для исправления ошибок.
- Величина $k$ должна быть достаточно большой, чтобы вероятность того, что случайный $k$-мер заданной длины встречается в геноме, была ничтожно малой. В противном случае некоторые $k$-меры, содержащие ошибку, не будут исправлены только потому, что имеется достаточно большая вероятность того, что эти $k$-меры на самом деле встречаются в геноме, а тогда они могут много раз встречаться в чтениях.

В соответствии с этим, сделаем оценку на величину $k$. Для нормальной работы алгоритма исправления ошибок необходимо, чтобы «надежных» $k$-меров было достаточно много. В противном случае, при замене нуклеотидов в «подозрительных» $k$-мерах, они не будут попадать в группу «надежных», а значит, ошибки исправлены не будут.

Для произвольного значения $k$ оценим вероятность того, что некоторый $k$-мер был прочитан хотя бы $t$ раз. Введем обозначения: пусть $L$ – длина генома, $n$ – число чтений, $r$ – длина некоторого чтения, $n(r)$ – число чтений длины $r$. Тогда вероятность того, что произвольное чтение имеет длину $r$, равна $pr(r) = \frac{n(r)}{n}$.

Зафиксируем $k$-мер, начинающийся в геноме с позиции $pos$. Посчитаем число позиций, с которых может начинаться чтение длины $r$ таких, что это чтение содержит данный $k$-мер. Для $r \geq k$ их число равно $r - k + 1$ (рис. 1). Теперь рассмотрим все чтения, и для каждого чтения отметим его начало. Тогда в первом приближении можно считать, что начала чтений равномерно распределены по всей длине генома. В этом случае вероятность того, что данный $k$-мер, начинающийся в геноме с позиции $pos$, входил в случайное чтение длины $r$ равна $\frac{r - k + 1}{L - r + 1} \cdot pr(r)$, так как чтение длины $r$ могло начинаться в $L - r + 1$ позиции, а из них подходящими является $r - k + 1$ позиция. Вероятность того, что данный $k$-мер входит в случайное чтение, является суммой соответствующих вероятностей по всем длинам чтений, не меньшим $k$:

$$p = \sum_{r \geq k} \frac{r - k + 1}{L - r + 1} \cdot pr(r).$$

Тогда вероятность того, что данный $k$-мер входит ровно в $m$ чтений равна $C_n^m p^m (1 - p)^{n-m}$. Теперь можно оценить вероятность того, что данный $k$-мер входит хотя бы в $t$ чтений:

$$P(t) = \sum_{m=t}^{n} C_n^m p^m (1 - p)^{n-m}.$$

Значение $P(t)$ дает оценку вероятности того, что произвольный $k$-мер будет надежным. Оценка эта не является строгой, так как, во-первых, мы здесь не учитывается то обстоятельство, что некоторые $k$-меры встречаются в геноме несколько раз, во-вторых, не учитывается вероятность появления ошибки при чтении и, наконец, как было упомянуто выше, рассуждения верны лишь для случая равномерного распределения чтений по геному. Тем не менее, на практике это значение вполне может использоваться для выбора величины $k$.

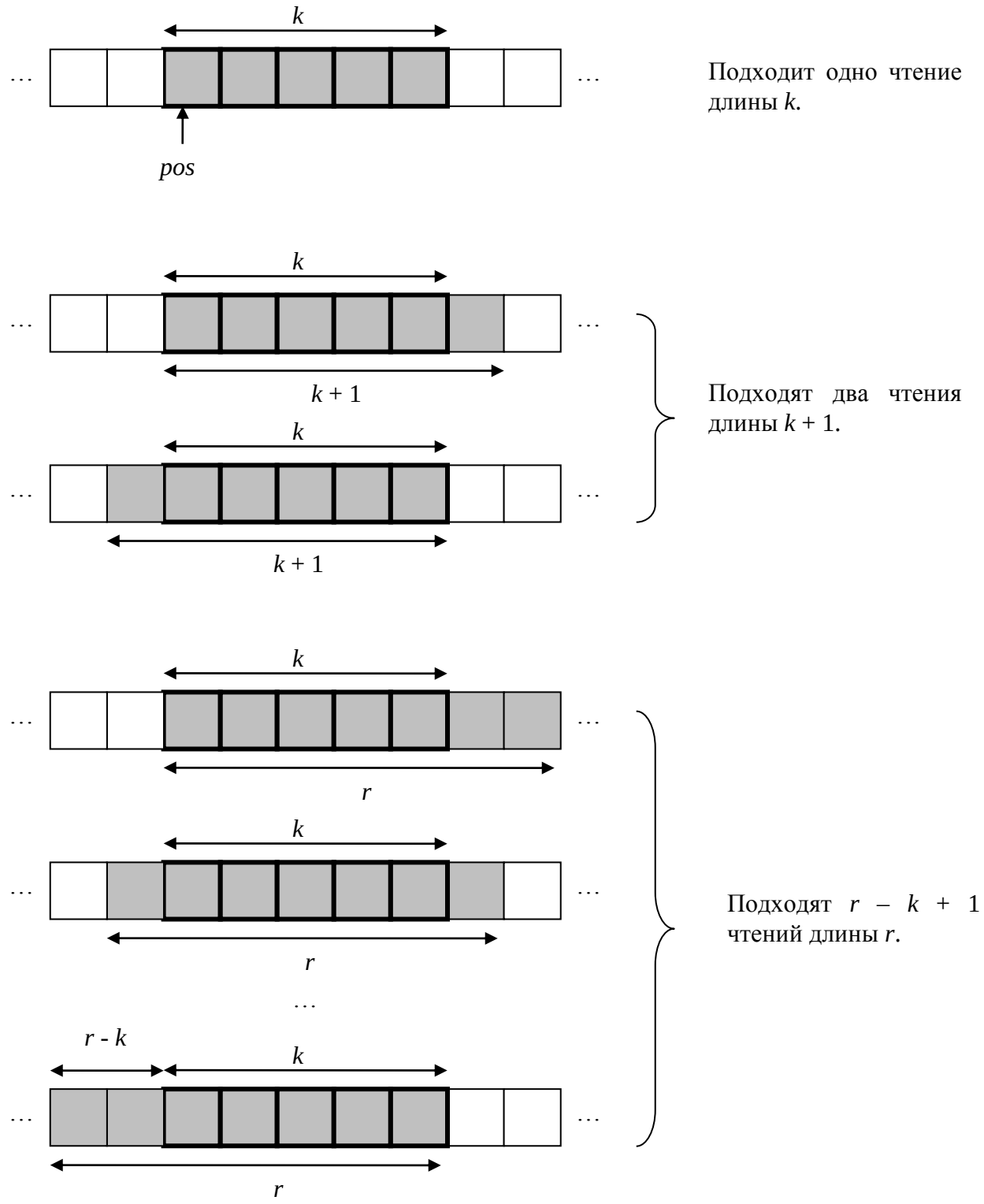



Рис. 1. Число подходящих чтений заданного *k*-мера в зависимости от длины чтения

### 1.3. Предлагаемый алгоритм

Для начала отметим, что качество нуклеотидов для чтений *Illumina* сильно ухудшается к концу чтения, поэтому для данного алгоритма имеет смысл обрезать куски чтений плохого качества, так как, с одной стороны, благодаря им образуется много «подозрительных» *k*-меров, с другой стороны, их не получится исправить из-за большого числа ошибок. Поэтому от каждого отрезается наидлиннейший префикс, в котором каждый нуклеотид имеет вероятность ошибки меньше 0.1 (рис. 2).

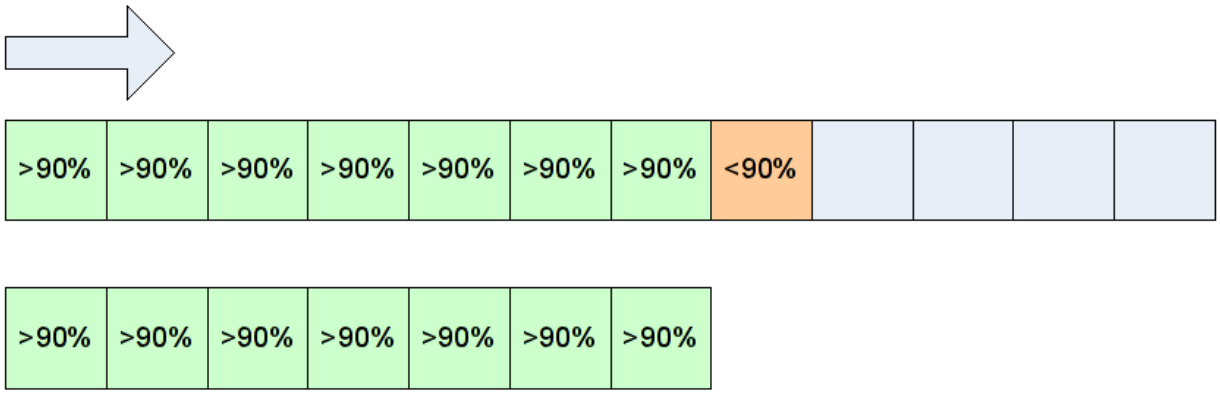



Рис. 2. Обрезание чтений исходя из качества нуклеотидов

Также важно заметить, что все *k*-меры можно разбить на группы на основании префикса небольшой длины, с которого они начинаются. Если при этом не исправлять ошибки в префиксе, то исправление не выводит *k*-мер из группы, что позволяет исправлять ошибки отдельно в группах, то есть еще больше уменьшить объем потребляемой алгоритмом оперативной памяти. Это не повлечет за собой неисправленные ошибки в префиксах, так как при обращении *k*-мера его префикс становится суффиксом (рис. 3).

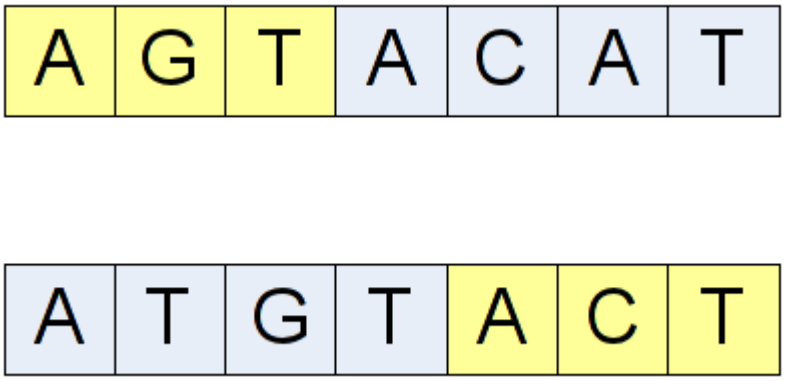



Рис. 3. *k*-мер и его обратно-комплементарная копия. Желтым цветом выделен префикс, на основе которого решается, в какую группу попадет этот *k*-мер

Таким образом, алгоритм состоит из следующих шагов:

1. Обрезание *k*-меров на основании качества.
2. Разбиение *k*-меров на группы на основании префиксов.
3. Выполнение для каждой группы:
   1. Сбор статистики по *k*-мерам.
   2. Исправление ошибок.

Важно отметить, что алгоритм поиска ошибок в *k*-мерах легко распараллеливается, так как для обработки одного *k*-мера ему требуется только доступ на чтение к общей структуре данных, хранящей статистику по содержанию *k*-меров в чтениях, а также кратковременный доступ на запись для сохранения результата.

### 1.4. Выбор параметра *t*

Существует два основных подход к выбору параметра *t* – теоретический и практический. При теоретическом подходе считается, что распределение частот *k*-меров является смесью двух распределений – распределения, соответствующего «надежным» *k*-мерам, и распределения, соответствующего «подозрительным» *k*-мерам. Считается, что позиции ошибок при чтении независимы. При такой модели оба указанных выше распределения являются распределениями Пуассона с разными параметрами.

Для нахождения значения порога находится первый локальный минимум этого распределения. Частота, соответствующая этому минимуму, считается пороговым значением.

В практическом подходе не делается никаких предположений о вероятностных моделях. Вместо этого распределение частот *k*-меров строится прямо из данных чтений. По полученному распределению аналогично теоретическому подходу частота, соответствующая первому локальному минимуму, выбирается в качестве порога.

### 1.5. Приблизительная оценка эффективности предложенного алгоритма

Простота предлагаемого алгоритма позволяет довольно просто оценить, насколько он эффективен. При этом возникают следующие соображения:

1. Предположим, что в каждом *k*-мере исправляется не более *N* ошибок. Тогда те *k*-меры, в которых больше *N* ошибок, точно не будут исправлены. Однако из этого не следует, что необходимо выбирать большое значение *N,* так как увеличение его влечет за собой увеличение трудоемкости алгоритма. Кроме того, при больших значениях *N* велика вероятность наличия *k*-мера, который можно исправить несколькими способами, так как уменьшается число позиций *k*-мера с фиксированными нуклеотидами.
2. Для того, чтобы *k*-мер с ошибкой мог быть исправлен, необходимо и достаточно, чтобы его безошибочный вариант был прочитан хотя бы *t* раз, а также не было другого «надежного» *k*-мера, отличающегося от этого не более, чем в *N* позициях.

Таким образом, при правильном (соответствующем исходным данным) выборе параметров предлагаемый метод не сможет обработать лишь три типа *k*-меров:

- Те, которые были хотя бы *t* раз прочитаны неверно (возможно, по-разному) и отличающиеся от них в не более, чем *N* позициях. Вероятностная мера *t* одинаково неверных *k*-меров равна вероятностной мере одного, возведенной в степень *t.* При значениях *t,* не меньших трех, значение этой меры для $k = 30,\ N = 1,\ p = 0.001$ имеет порядок минус шесть или меньший.
- Те, которые неверно определились как ошибочные в связи с повторами в геноме. Вероятностная мера этих *k*-меров вычисляется при помощи биологических соображений и тоже достаточно мала (имеет порядок минус три или меньший).

- Те, в которых больше *N* ошибок. При достаточно большом *N* таких доля *k*-меров также имеет маленький порядок (минус четыре и ниже).

### 1.6. Пример работы алгоритма

Проиллюстрируем работу разработанного алгоритма на примере исправления ошибок в чтениях тестового генома. Тестовый геном представляет собой фрагмент генома *E. coli* из 70 нуклеотидов:

```
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
```

В табл. 1 представлены чтения тестового генома, длина каждого чтения состовляет 20 нуклеотидов. Производятся парные чтения с разных концов фрагментов длины 50, в приведенной таблице в каждой строке указываются два соответствующих чтения. При этом с некоторой вероятностью в случайных местах чтений вносятся ошибки в соответствии с тем, как реально работают секвенаторы. Далее некоторые чтения обрезаются исходя из качества нуклеотидов. Заметим, что современные секвенаторы помимо каждого чтения возвращают также характеристику качества прочтения каждого из нуклеотидов этого чтения [4, 5, 9 – 11]. Аналогично, к исходным данным чтений в данном примере прикладывались также данные о качестве прочтений.

На основе полученных чтений строится множество *k*-меров (длина *k*-меров в данном случае взята равной 12). Множество построенных *k*-меров представлено в табл. 2. В первом столбце приведены все различные *k*-меры, встречающиеся в чтениях, а во втором столбце для каждого *k*-мера указано, сколько раз данный *k*-мер встречается в чтениях.

Алгоритм на основе данных, представленных в табл. 2, выбирает значение порога *t* (разд. 1.4). В данном примере значение *t* равно четырем. Далее рассматриваются все *k*-меры, которые встречаются менее *t* раз (значение во втором столбце табл. 2 не превышает трех), и предпринимается попытка исправить в них ошибку путем замены одного нуклеотида. В случае, если сумма числа встреч *k*-мера до исправления и после оказывается не менее *t* (не менее четырех в данном случае), и такая ситуация имеет место быть лишь для одного нуклеотида, то совершается замена. Список совершенных замен для данного примера представлен в табл. 3. В ней исправленные нуклеотиды выделены полужирным курсивом.

Всего алгоритм исправил 48 ошибок. Изначально множество различных *k*-меров имело мощность 166, а после исправления ошибок, множество сократилось до 136 различных элементов.

Таблица 1. Фрагменты чтения тестового генома

| Первое чтение | Второе чтение |
|---|---|
| CAGACACTCTTTTTTTAATC | TGACTGCAACGGGCAATATG |
| TTTTTAATCCACACAGAGAC | GCTTTTCATTCTGACTGCAA |
| TGACTGCAACGGGCAATATG | AGACACTCTTTTTTCAATCC |
| ACTCTTTTTTTAATCCACAC | TTTCATTCTGACTGCAACGG |
| AGCTTTTCATTCTGACTGCA | TTTTTTTAATCCACACAGAG |
| ATCAGACACTCTTTTTTGAA | GACTGCAACGGGCAATATGT |
| CAACGGGCAATATGTCTCTG | TGCTATCAGACACTCTTTTT |
| GACTGCAACGGGCAATATGT | TGCTATCAGACACTCTTTTT |
| AGACACTCTTTTTTTAATCC | GACTGCAACGGGCAATATGT |
| GCTTTTCATTCTGACTGCAA | TTTTTTAATCCACACAGAGA |
| ACTCTTTTTTTAATCCACAC | CATTCTGACTGCAACGGGCA |
| TGCTATCAGACACTCTTTTT | CAACGGGCAATATGTCTCTG |
| CTCTTTTTTTAATCCACACA | TTCTGACTGCAACGGGCAAT |
| TATCAGACACTCTTTTTTTA | CTGCAACGGGCAATATGTCT |
| GACTGCAACGGGCAATATGT | CAGACACTCTTTTTTTAATC |
| CTATCAGACACTCTTTTTTA | CAACGGGCAATATGTCTCTG |
| TGCAACGGGCAATATGTCTC | AGACACTCTTTTTTTAAACC |
| AGACACTCTTTTCTTAATCC | TCTGACTGCAACGGGCAATA |
| CGGGCAATATGTCTCTGTGT | GCTGCTATCAGACACTCTTT |
| TCAGACACTCTTTTTTTAAT | CTGACTGCAACGGGCAATAT |
| TTCTGACTGCAACGGGCAAT | ACTCTTTTTTTAATCCACAC |
| CAACGGGCAATATGTCTCTG | GCTATCAGACACTCTTTTTT |
| CTTTTTTTAATCCACACAGA | TTCATTCTGACTGCAACGGG |
| CAGACACTCTTTTTTTAATC | TTCTGACTGCAACGAGCAAT |
| CTTTTTTTAATCCACACAGA | AGCTTTTCATTCTGACTGCA |
| TTCATTCTGACTGCAACGGG | CTCTTTTTTTAATCCACACA |
| CAGACACTCTTTTTTTAATC | CTGACTGCAACGGGCAATAT |
| CTTTTCATTCTGACTGCAAC | TCTTTTTTTAATCCACACAG |
| GCAACGGGCAATATGTCTCT | CTATCAGACACTCTTTTTTT |
| CAACGGGCAATATGTCTCTG | CTATCAGACACTCATTTTTT |
| GCTATCAGACACTCGTTTTT | ACTGCAACGGGCAATATGGC |
| TTCATTCTGACTGCAACGGG | CTCTTTTTTTAATCCACACA |
| CTATCAGACACTCTTTTTTT | ACTGCAACGGGCAATATGTC |
| TTTTAATCCACACAGAGACA | AGCTTTTCATTCTGACTGCA |
| TCTTTTTTTAATCCACACAG | TTTCATTCTGACTGCAACGG |
| GCTTTTCATTCTGACTGCAA | TTTAATCCACACAGAGACAT |
| TTTTAATCCACACAGAGACA | TTTCATTCTGACTTCAACGG |
| TGCTATCAGACACTCTTTTT | CAACGGGCAATATGTCTCTG |
| TTCTGACTGCAACGGGCAAT | ACACTCTTTTTTTAATCCAC |
| TGCAACGGGCAATATGTCTC | TGCTATCAGACACACCTTTT |

Таблица 2. Множество *k*-меров

| *k*-мер | Число до исправления | Число после исправления |
|---|---|---|
| GCCATATTGCCC | 2 | – |
| CAGACACTCATT | 2 | – |
| CATATTGCCCGT | 34 | 34 |
| AGCTTTTCATTC | 6 | 6 |
| TTTAATCCACAC | 32 | 32 |
| ATTGCCCGTTGC | 32 | 32 |
| CTGCTATCAGAC | 2 | 2 |
| GAATGAAAAGCT | 6 | 6 |
| AGACACTCATTT | 2 | 2 |
| GCAGTCAGAATG | 20 | 20 |
| ATATGTCTCTGT | 2 | 2 |
| CAATATGTCTCT | 14 | 14 |
| TGCTATCAGACA | 12 | 12 |
| GAAAAGAGTGTC | 2 | 2 |
| CCGTTGCAGTCA | 26 | 26 |
| TATTGCCCGTTG | 40 | 40 |
| ATTGAAAAAAGA | 2 | – |
| TCTTTTTTTAAT | 30 | 32 |
| TGTGTGGATTAA | 26 | 26 |
| GAGACATATTGC | 20 | 20 |
| TCCACACAGAGA | 10 | 10 |
| TATGTCTCTGTG | 2 | 2 |
| TCTTTTCTTAAT | 2 | 2 |
| CTCTGTGTGGAT | 12 | 12 |
| ACACTCTTTTTT | 28 | 30 |
| ATTAAGAAAAGA | 2 | 2 |
| GTGTGGATTAAA | 32 | 32 |
| CGTTGCAGTCAG | 24 | 24 |
| CTCTTTTCTTAA | 2 | 2 |
| CCACACAGAGAC | 8 | 8 |
| CAGTCAGAATGA | 18 | 18 |
| CTTTTTTTAATC | 32 | 34 |
| GGATTAAGAAAA | 2 | 2 |
| TTGCCCGTTGCA | 30 | 30 |
| CCATATTGCCCG | 2 | – |
| ACAGAGACATAT | 2 | 2 |
| AGAGACATATTG | 14 | 14 |
| GTCAGAATGAAA | 16 | 16 |
| TGACTGCAACGG | 26 | 26 |
| AAAATGAGTGTC | 2 | – |
| GTCTCTGTGTGG | 8 | 8 |
| AAAAAAGAGTGT | 28 | 30 |

| | | |
|---|---|---|
| TGAAAAAAGAGT | 2 | – |
| TGTCTCTGTGTG | 6 | 6 |
| TGTGGATTAAAA | 30 | 30 |
| AAAAAAAGAGTG | 22 | 26 |
| CACTCTTTTTTT | 22 | 26 |
| TCAGACACTCTT | 22 | 24 |
| GTGTCTGATAGC | 14 | 14 |
| TTAAAAAAAGAG | 28 | 30 |
| GACATATTGCCC | 24 | 26 |
| TAAGAAAAGAGT | 2 | 2 |
| ACACTCATTTTT | 2 | – |
| TAAAAAAGAGTG | 2 | 2 |
| CACTCTTTTTTA | 2 | 2 |
| ACTCTTTTTTTA | 24 | 26 |
| TGCAACGGGCAA | 30 | 30 |
| CTCTTTTTTTAA | 28 | 30 |
| TCTTTTTTCAAT | 2 | – |
| CTGACTGCAACG | 24 | 24 |
| ATATTGCCCGTT | 36 | 36 |
| GGATTGAAAAAA | 2 | – |
| CACTCTTTTTTC | 2 | – |
| CGGGCAATATGT | 30 | 32 |
| ACTCTTTTCTTA | 2 | 2 |
| AAAAAGAGTGTC | 34 | 36 |
| TATCAGACACTC | 22 | 22 |
| AGTCAGAATGAA | 22 | 22 |
| TTTTAATCCACA | 30 | 30 |
| CTTTTTTCAATC | 2 | – |
| ACACAGAGACAT | 4 | 4 |
| GCCCGTTGCAGT | 28 | 28 |
| TGCCCGTTGCAG | 28 | 28 |
| AATGAGTGTCTG | 2 | – |
| GATTAAGAAAAG | 2 | 2 |
| CCCGTTGCAGTC | 30 | 30 |
| GACTGCAACGGG | 30 | 30 |
| TTCTGACTGCAA | 22 | 22 |
| GAAAAAAGAGTG | 2 | – |
| ATGAGTGTCTGA | 2 | – |
| CACTCTTTTCTT | 2 | 2 |
| GACACTCATTTT | 2 | – |
| GCAATATGTCTC | 20 | 20 |
| TGTCTGATAGCA | 12 | 12 |
| ATCAGACACTCT | 20 | 22 |
| TTTCATTCTGAC | 16 | 16 |
| AACGGGCAATAT | 36 | 36 |
| CAGAATGAAAAG | 10 | 10 |

| | | |
|---|---|---|
| ATCCACACAGAG | 12 | 12 |
| ATTAAAAAAAGA | 30 | 32 |
| CAACGGGCAATA | 40 | 40 |
| GGGCAATATGTC | 24 | 26 |
| ACTCTTTTTTCA | 2 | – |
| TTTTCATTCTGA | 10 | 10 |
| CAGAGACATATT | 12 | 12 |
| GTCTGATAGCAG | 2 | 2 |
| AAAAAATGAGTG | 2 | – |
| TGAGTGTCTGAT | 2 | – |
| TGCAGTCAGAAT | 20 | 20 |
| GCTTTTCATTCT | 10 | 10 |
| TCTGACTGCAAC | 20 | 20 |
| AGAAAAGAGTGT | 2 | 2 |
| TTTTTTTAATCC | 26 | 28 |
| CAGACACTCTTT | 30 | 32 |
| TTAAGAAAAGAG | 2 | 2 |
| GACACTCTTTTC | 2 | 2 |
| CACAGAGACATA | 2 | 2 |
| CTCTTTTTTCAA | 2 | – |
| TCAGACACTCAT | 2 | – |
| ATGTCTCTGTGT | 4 | 4 |
| AGTGTCTGATAG | 20 | 20 |
| CACTCATTTTTT | 2 | – |
| ACTGCAACGGGC | 28 | 28 |
| TCAGAATGAAAA | 10 | 10 |
| AATATGTCTCTG | 12 | 12 |
| AGACATATTGCC | 22 | 22 |
| GTGGATTAAAAA | 28 | 28 |
| CTGTGTGGATTA | 20 | 20 |
| GATTGAAAAAAG | 2 | – |
| GATTAAAAAAAG | 32 | 34 |
| GGCAATATGTCT | 22 | 22 |
| CATTCTGACTGC | 20 | 20 |
| GTTGCAGTCAGA | 20 | 20 |
| ACGGGCAATATG | 34 | 34 |
| CTGCAACGGGCA | 28 | 28 |
| GCTGCTATCAGA | 2 | 2 |
| ACATATTGCCCG | 30 | 32 |
| GCTATCAGACAC | 14 | 14 |
| AAGAGTGTCTGA | 22 | 24 |
| TTAATCCACACA | 26 | 26 |
| CGGGCAATATGG | 2 | – |
| TTTTCTTAATCC | 2 | 2 |
| GGATTAAAAAAA | 26 | 28 |
| AAATGAGTGTCT | 2 | – |

| | | |
|---|---|---|
| AAAGAGTGTCTG | 30 | 32 |
| AGAGTGTCTGAT | 20 | 22 |
| AAGAAAAGAGTG | 2 | 2 |
| AATCCACACAGA | 16 | 16 |
| ATCAGACACTCA | 2 | – |
| AGACACTCTTTT | 36 | 38 |
| AAAAGAGTGTCT | 36 | 38 |
| TTCATTCTGACT | 22 | 22 |
| AAAAATGAGTGT | 2 | – |
| TTGAAAAAAGAG | 2 | – |
| GAGTGTCTGATA | 22 | 22 |
| TAAAAAAAGAGT | 24 | 26 |
| CTTTTCTTAATC | 2 | 2 |
| TCATTCTGACTG | 18 | 18 |
| CTATCAGACACT | 20 | 20 |
| TCTGTGTGGATT | 16 | 16 |
| GCAACGGGCAAT | 32 | 32 |
| TTTTTAATCCAC | 28 | 28 |
| TTGCAGTCAGAA | 22 | 22 |
| TGGATTAAAAAA | 26 | 26 |
| CACACAGAGACA | 6 | 6 |
| ACACTCTTTTCT | 2 | 2 |
| TTTTTTAATCCA | 26 | 26 |
| TAATCCACACAG | 20 | 20 |
| AGAATGAAAAGC | 10 | 10 |
| GACACTCTTTTT | 34 | 36 |
| TTTTTTCAATCC | 2 | – |
| GGGCAATATGGC | 2 | – |
| TCTGATAGCAGC | 2 | 2 |
| CTTTTCATTCTG | 10 | 10 |
| TCTCTGTGTGGA | 10 | 10 |
| ATTCTGACTGCA | 20 | 20 |

Таблица 3. Исправления ошибок

| *k*-мер до исправления | *k*-мер после исправления |
|---|---|
| GCCATATTGCCC | G***A***CATATTGCCC |
| CAGACACTCATT | CAGACACTC***T***TT |
| AGACACTCATTT | AGACACTC***T***TTT |
| GAAAAGAGTGTC | ***A***AAAAGAGTGTC |
| ATTGAAAAAAGA | ATT***A***AAAAAAGA |
| TCTTTTCTTAAT | TCTTTT***T***TTAAT |
| ATTAAGAAAAGA | ATTAA***A***AAAAGA |
| CTCTTTTCTTAA | CTCTTTT***T***TTAA |
| GGATTAAGAAAA | GGATTAA***A***AAAA |
| CCATATTGCCCG | ***A***CATATTGCCCG |

| | |
|---|---|
| AAAATGAGTGTC | AAAA***A***GAGTGTC |
| TGAAAAAAGAGT | T***A***AAAAAAGAGT |
| TAAGAAAAGAGT | TAA***A***AAAAGAGT |
| ACACTCATTTTT | ACACTC***T***TTTTT |
| TAAAAAAGAGTG | ***A***AAAAAAGAGTG |
| CACTCTTTTTTA | CACTCTTTTTT***T*** |
| TCTTTTTTCAAT | TCTTTTTT***T***AAT |
| GGATTGAAAAAA | GGATT***A***AAAAAA |
| CACTCTTTTTTC | CACTCTTTTTT***T*** |
| ACTCTTTTCTTA | ACTCTTTT***T***TTA |
| CTTTTTTCAATC | CTTTTTT***T***AATC |
| AATGAGTGTCTG | AA***A***GAGTGTCTG |
| GATTAAGAAAAG | GATTAA***A***AAAAG |
| GAAAAAAGAGTG | ***A***AAAAAAGAGTG |
| ATGAGTGTCTGA | A***A***GAGTGTCTGA |
| CACTCTTTTCTT | CACTCTTTT***T***TT |
| GACACTCATTTT | GACACTC***T***TTTT |
| ACTCTTTTTTCA | ACTCTTTTTT***T***A |
| AAAAAATGAGTG | AAAAAAA***A***GAGTG |
| TGAGTGTCTGAT | ***A***GAGTGTCTGAT |
| AGAAAAGAGTGT | A***A***AAAAGAGTGT |
| TTAAGAAAAGAG | TTAA***A***AAAAGAG |
| GACACTCTTTTC | GACACTCTTTT***T*** |
| CTCTTTTTTCAA | CTCTTTTTT***T***AA |
| TCAGACACTCAT | TCAGACACTC***T***T |
| CACTCATTTTTT | CACTC***T***TTTTTT |
| GATTGAAAAAAG | GATT***A***AAAAAAG |
| CGGGCAATATGG | CGGGCAATATG***T*** |
| TTTTCTTAATCC | TTTT***T***TTAATCC |
| AAATGAGTGTCT | AAA***A***GAGTGTCT |
| AAGAAAAGAGTG | AA***A***AAAAGAGTG |
| ATCAGACACTCA | ATCAGACACTC***T*** |
| AAAAATGAGTGT | AAAAA***A***GAGTGT |
| TTGAAAAAAGAG | TT***A***AAAAAAGAG |
| CTTTTCTTAATC | CTTTT***T***TTAATC |
| ACACTCTTTTCT | ACACTCTTTT***T***T |
| TTTTTTCAATCC | TTTTTT***T***AATCC |
| GGGCAATATGGC | GGGCAATATG***T***C |

## Выводы по главе 1

1. Предложен алгоритм исправления ошибок в наборе чтений геномной последовательности, основанный на частотном анализе встречаемости *k*-меров.
2. Предложен подход для выбора параметров алгоритма: *k* – длины *k*-мера и величины *t* – порога, относящего *k*-мер к числу надежных или подозрительных.
3. Представлена оценка эффективности разработанного алгоритма по времени.
4. Приведено описание работы алгоритма исправления ошибок на тестовом примере.
5. Разработанный алгоритм может использоваться как часть процесса секвенирования генома.

# 2. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ИСПРАВЛЕНИЯ ОШИБОК В ДАННЫХ СЕКВЕНИРОВАНИЯ

## 2.1. Программная реализация алгоритма

В рамках данной работы была создана программная реализация разработанного алгоритма на языке программирования *Java*.

Ниже в листинге приведена реализация функции `findNFixes`, которая осуществляет поиск возможностей исправления ошибок в *k*-мере. Максимальное число возможных ошибок *n* передается в функцию как параметр. Предпочтение отдается меньшему числу ошибок – как только удалось исправить какое-то, не превышающее *n*, число ошибок так, что *k*-мер стал «надежным», функция прекращает свою работу.

Параметр *DEFINITELY_GOOD_THRESHOLD* – это упоминавшийся выше параметр *t*.

Листинг. Поиск возможностей исправления ошибок

```
    private int findNFixes(long str, int n, int begin)
                  {
        if (n == 0) {
            Info minfo = hm.get(str);
      if ((minfo != null) && (minfo.count.intValue()
                >= DEFINITELY_GOOD_THRESHOLD)) {
                newStr = str;
                return 1;
            } else {
                return 0;
            }
        }
        int k = 0;
    int maxPos = (begin == 0) ? (len - 1) : begin +
                 MAXIMAL_FIXES_DISTANCE;
        maxPos = Math.min(maxPos, len - n);
        for (int pos = begin; (k < 2) && (pos <=
                 maxPos); pos++) {
            for (long xor = 1; xor <= 3; xor++) {
                long nstr = str ^ (xor << (2 * pos));
                int r = findNFixes(nstr, n - 1, pos +
                  1);
                if (r == 2) {
                    k = 2;
                    break;
                }
                if (r == 0) {
                    continue;
                }
                k++;
                if (k == 2) {
```

```
                    break;
                }
            }
        }
        return k;
    }
```

## 2.2. Результаты работы алгоритма

Описанный в данной работе алгоритм исправления ошибок был реализован и применен в рамках проекта *dnGASP* [12], в котором участникам предлагалось восстановить синтетическую последовательность ДНК длиной около 1,8 миллиардов нуклеотидов, используя для этого любые уже имеющиеся или специально разработанные средства. При этом, хотя исходный геном был синтетическим, он представлял собой смесь геномов различных живых организмов – задача была близка к реальной.

В качестве исходных данных участникам была предоставлена библиотека парных чтений синтезированного генома. Эти чтения обеспечивали достаточно большое покрытие генома – 44. Также в них искусственным образом были внесены ошибки, для того чтобы сделать задачу максимально похожей на реальную.

Описываемый метод был использован в качестве одного из этапов алгоритма сборки генома. Алгоритм запускался на 24-ядерном компьютере с 24 ГБ оперативной памяти, что слишком мало для того, чтобы для исправления ошибок можно было использовать уже существующие методы.

Для оценки эффективности работы приведем некоторую статистику по *k*-мерам, содержащимся в чтениях, собранную два раза – до исправления ошибок и после него.

До запуска алгоритма в исходных данных было 6,5 миллиардов различных *k*-меров, из которых 3 миллиарда «надежных». Алгоритм работал около суток, после этого в данных стало 3,9 миллиардов (что на 40% меньше, чем в начале) различных *k*-меров, из которых 3,3 миллиарда «надежных». Таким образом, число «подозрительных» *k*-меров уменьшилось с 3,5 миллиардов до 0,6 миллиардов (примерно на 83%).

Число *k*-меров после исправления ошибок уменьшилось настолько, что граф де Брюина, используемый на последующих этапах сборки генома, сможет уместиться в оперативную память среднего по вычислительной мощности сервера (примерно 64 ГБ).

На рис. 4 и рис. 5 приведены графики частот *k*-меров в данных до исправления ошибок и после него. Данные, по которым построены графики, приведены в приложении 1.

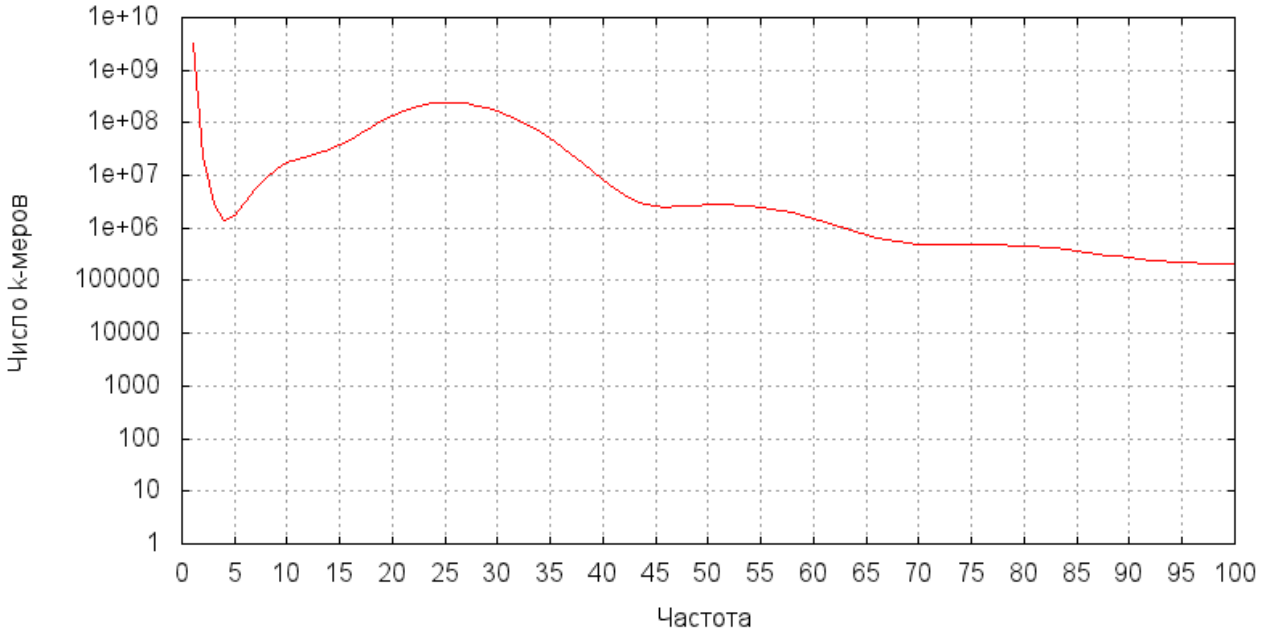



Рис. 4. Распределение частот *k*-меров в данных *dnGASP* до исправления ошибок

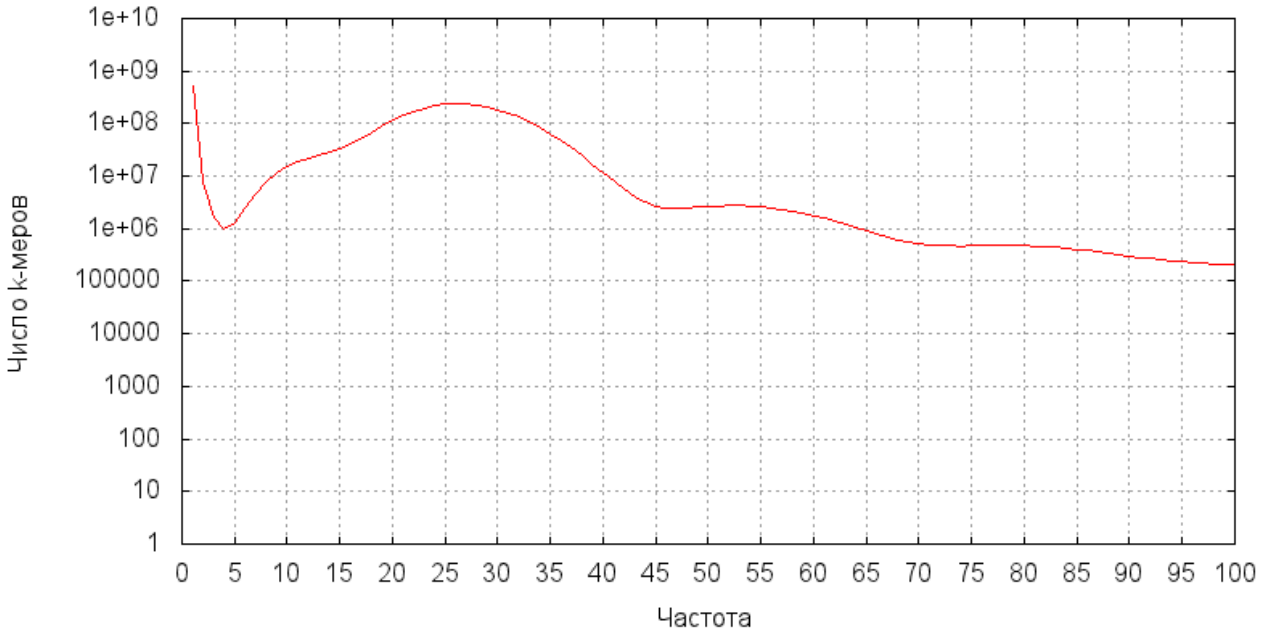



Рис. 5. Распределение частот *k*-меров в данных *dnGASP* после исправления ошибок

## Выводы по главе 2

1. Приведено описание программной реализации алгоритма исправления ошибок в наборе чтений геномной последовательности, основанном на частотном анализе встречаемости *k*-меров. Полная реализация алгоритма представлена в приложении 4.

2. Проведено экспериментальное исследование разработанного метода, показавшее работоспособность метода на данных, близких к реальным. Разработанный метод может эффективно использовать как небольшой объем ресурсов, так и большой.

## ЗАКЛЮЧЕНИЕ

В настоящей работе получены следующие результаты:

- разработан алгоритм исправления ошибок в данных секвенирования, превосходящий существующие методы по масштабируемости, а также гораздо менее требовательный к вычислительным ресурсам;
- проведен анализ данного алгоритма;
- осуществелна программная реализация алгоритма исправления ошибок в данных секвенирования на языке *Java*;
- разработанная программа успешно протестирована на синтетических данных, составленных из геномов живых организмов;

Кроме этого, по результатам НИР была опубликована статья в журнале «Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики» (входит в перечень ВАК) и получено свидетельство о регистрации программы для ЭВМ.

## СПИСОК ЛИТЕРАТУРЫ

1. *Zerbino D. R., Birney E.* Velvet: algorithms for de novo short read assembly using de Bruijn graphs // Genome Res. May 2008. Vol. 18. No. 5, pp. 821 – 829.
2. *Pevzner P. A., Tang H., Waterman M. S.* An eulerian path approach to DNA fragment assembly // Proc Nat Acad Sci USA. Aug 2001. Vol. 98. No. 17, pp. 9748 – 9753.
3. *Pevzner P. A., Tang H.* Fragment assembly with double-barreled data. Bioinformatics. 2001 Jun 17. Suppl 1: S225 – 233.
4. *Cock P., Fields C., Goto N., Heuer M., Rice P.* The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants // Nucleic Acids Research. 2010. Vol. 38. No. 6, 1767 – 1777.
5. *Phred -Quality Base Calling*. http://www.phrap.com/phred/.
6. *Butler J., MacCallum I., Kleber M., Shlyakhter I. A., Belmonte M. K., Lander E. S., Nusbaum C., Jaffe D. B.* Allpaths: de novo assembly of whole-genome shotgun microreads // Genome Research. 2008. Vol. 18. No. 5, pp. 810 – 820.
7. *Chaisson M. J., Brinza D., Pevzner P. A.* De novo fragment assembly with short mate-paired reads: Does the read length matter? // Genome Research. 2009. Vol. 19. No. 2, pp. 336 – 346.
8. *Александров А.В., Казаков С.В., Мельников С.В., Сергушичев А.А., Царев Ф.Н., Шалыто А.А.* Метод исправления ошибок в наборе чтений нуклеотидной последовательности // Научно-технический вестник СПбГУ ИТМО. 2011. № 5, с. 81 – 84.
9. *The SAM Format Specication*. http://samtools.sourceforge.net/SAM1.pdf.
10. *NCBI*: Formats: Documentation: Trace Archive v. 4.2: NCBI/NLM/NI. http://www.ncbi.nlm.nih.gov/Traces/trace.cgi?cmd=show&f=formats&m=doc&s=format.
11. *UCSC Genome Browser*: BAM Track Format. https://cgwb.nci.nih.gov/goldenPath/help/bam.html.
12. *dnGASP*. National Center for Genome Analysis. Barcelona. http://cnag.bsc.es/.

# ПРИЛОЖЕНИЕ 1. ДАННЫЕ О КОЛИЧЕСТВЕ *K*-МЕРОВ ДО И ПОСЛЕ ИСПРАВЛЕНИЯ ОШИБОК

| *n* | Число *k*-меров, содержащихся в чтениях *n* раз (до исправления ошибок) | Число *k*-меров, содержащихся в чтениях *n* раз (после исправления ошибок) |
|---|---|---|
| 1 | 3130509297 | 532770846 |
| 2 | 23978495 | 7370346 |
| 3 | 2833469 | 1644074 |
| 4 | 1405930 | 964250 |
| 5 | 1725467 | 1344162 |
| 6 | 3108862 | 2532697 |
| 7 | 5542599 | 4654354 |
| 8 | 8933360 | 7702092 |
| 9 | 12938954 | 11420948 |
| 10 | 17085135 | 15424645 |
| 11 | 20881921 | 19152192 |
| 12 | 24230586 | 22400507 |
| 13 | 27471307 | 25229360 |
| 14 | 31523923 | 28305739 |
| 15 | 37799720 | 32806030 |
| 16 | 47684652 | 40057072 |
| 17 | 62325861 | 51332438 |

| | | |
|---|---|---|
| 18 | 82298947 | 67490362 |
| 19 | 107159743 | 88729412 |
| 20 | 135555086 | 114335600 |
| 21 | 165252192 | 142632754 |
| 22 | 193402998 | 171280538 |
| 23 | 217247228 | 197635546 |
| 24 | 234200378 | 219115389 |
| 25 | 242664845 | 233477182 |
| 26 | 241828429 | 239406922 |
| 27 | 232140446 | 236370280 |
| 28 | 214891711 | 225145906 |
| 29 | 192086970 | 207101590 |
| 30 | 166024726 | 184147485 |
| 31 | 138955979 | 158526266 |
| 32 | 112622571 | 132191488 |
| 33 | 88545487 | 106922872 |
| 34 | 67641312 | 83997181 |
| 35 | 50267171 | 64138710 |
| 36 | 36386246 | 47678694 |
| 37 | 25729448 | 34547920 |
| 38 | 17850736 | 24481497 |

| | | |
|---|---|---|
| 39 | 12228555 | 17031140 |
| 40 | 8359047 | 11704969 |
| 41 | 5816821 | 8024776 |
| 42 | 4216106 | 5603302 |
| 43 | 3274746 | 4074792 |
| 44 | 2777564 | 3170799 |
| 45 | 2556617 | 2683700 |
| 46 | 2511250 | 2471540 |
| 47 | 2553259 | 2431984 |
| 48 | 2632677 | 2475747 |
| 49 | 2718603 | 2568770 |
| 50 | 2782930 | 2658262 |
| 51 | 2808657 | 2728760 |
| 52 | 2791588 | 2770382 |
| 53 | 2735871 | 2770092 |
| 54 | 2632558 | 2726030 |
| 55 | 2497079 | 2642596 |
| 56 | 2322248 | 2519509 |
| 57 | 2133322 | 2363632 |
| 58 | 1927375 | 2178675 |
| 59 | 1722393 | 1992026 |

| | | |
|---|---|---|
| 60 | 1520430 | 1784901 |
| 61 | 1325020 | 1583972 |
| 62 | 1153745 | 1389619 |
| 63 | 997751 | 1213840 |
| 64 | 861306 | 1049738 |
| 65 | 753160 | 909628 |
| 66 | 668082 | 790598 |
| 67 | 599797 | 695844 |
| 68 | 549866 | 620804 |
| 69 | 517298 | 564282 |
| 70 | 494095 | 524151 |
| 71 | 482642 | 497198 |
| 72 | 477257 | 479743 |
| 73 | 475630 | 474016 |
| 74 | 477263 | 470737 |
| 75 | 479429 | 470848 |
| 76 | 480998 | 475365 |
| 77 | 479848 | 477194 |
| 78 | 476366 | 480184 |
| 79 | 468307 | 479912 |
| 80 | 458905 | 471901 |

| | | |
|---|---|---|
| 81 | 445166 | 466868 |
| 82 | 429245 | 455716 |
| 83 | 409881 | 441110 |
| 84 | 390765 | 423482 |
| 85 | 367541 | 406266 |
| 86 | 346968 | 384481 |
| 87 | 326814 | 362576 |
| 88 | 307798 | 343473 |
| 89 | 289152 | 321062 |
| 90 | 272594 | 301967 |
| 91 | 258091 | 283574 |
| 92 | 246713 | 268498 |
| 93 | 236797 | 255458 |
| 94 | 227976 | 241154 |
| 95 | 222284 | 233104 |
| 96 | 219401 | 224752 |
| 97 | 216547 | 219596 |
| 98 | 215956 | 216093 |
| 99 | 214181 | 213106 |
| 100 | 212337 | 211109 |

## ПРИЛОЖЕНИЕ 2. КОПИЯ ТЕКСТА ОПУБЛИКОВАННОЙ СТАТЬИ

В данном приложении приведена копия статьи, опубликованной в рамках выполненных на втором этапе работ по контракту:

- Александров А.В., Казаков С.В., Мельников С.В., Сергушичев А.А., Царев Ф.Н., Шалыто А.А. Метод исправления ошибок в наборе чтений нуклеотидной последовательности // Научно-технический вестник СПбГУ ИТМО. 2011. № 5, с. 81 – 84.

В показанном примере используются два порта в общее адресное пространство. Для обеспечения бесконфликтной работы памяти достаточно применить двухпортовую память или регистровый файл. В случае, когда доступна только однопортовая память, акселератор будет проводить половину времени работы в ожидании данных от памяти. В более сложных случаях для обеспечения высокой производительности алгоритм должен быть переработан для использования нескольких независимых банков памяти.

В показанном примере нет межитерационных зависимостей, поэтому он может быть легко распараллелен путем дублирования аппаратуры. Единственная обратная связь проходит через функциональный блок (+1) в счетчике итераций. Если задержка через этот функциональный блок составляет 5 нс, то акселератор сможет работать на частоте до 200 МГц при соответствующей конвейеризации остальной части тракта данных. Чтобы поднять частоту акселератора, потребуется разбить операцию на обратной связи на несколько стадий, однако это не приведет к увеличению производительности, так как счетчик итерации будет производить действительные данные только 1 раз за $N$ тактов, где $N$ – число конвейерных стадий на обратной связи.

## Заключение

В работе представлены основные типы потоковых вычислительных моделей, пригодных для аппаратной реализации. Показан пример создания акселератора на основе расширенной BDF модели, являющейся одной из наиболее простых и эффективных при прямой реализации в аппаратуре. Выделены основные факторы, влияющие на производительность потоковых машин.

Потоковые модели обладают значительной выразительной мощью, что позволяет с их применением описывать широкий класс алгоритмов из различных предметных областей. При этом полученные модели пригодны как для прямой аппаратной реализации, так и для отображения на различные реконфигурируемые машины [5]. Наиболее перспективной областью применения потоковых моделей является их использование в виде промежуточного представления в системах высокоуровневого синтеза. С их помощью можно выразить все вычислительные и управляющие конструкции, используемые в современных языках высокого уровня, при этом они позволяют максимально использовать возможности распараллеливания вычислений в специализированной аппаратуре.

## Литература

1. Budiu M., Artigas P.V., Goldstein S.C. Dataflow: A Complement to Superscalar // Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software. – Washington, DC, USA, 2005. – P. 177–186.
2. Kahn G. The semantics of a simple language for parallel programming // Proc. of IFIP Congress. – Stockholm, Sweden, 1974. – P. 471–475.
3. Edward A. Lee, David G. Messerschmitt. Synchronous Data Flow // Proc. of the IEEE. – 1997. – V. 75. – № 9. – P. 1235–1245.
4. Buck J.T. A dynamic dataflow model suitable for efficient mixed hardware and software implementations of DSP applications // Proc. of the Third International Workshop on Hardware/Software Codesign. – France, 1994. – P. 165–172.
5. Swanson S., Michelson, K. WaveScalar // Proc. of the 36th annual IEEE/ACM International Symposium on Microarchitecture. – USA, 2003. – P. 291–302.

*Попов Роман Игоревич* – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, ripopov@gmail.com

УДК 004.021

# МЕТОД ИСПРАВЛЕНИЯ ОШИБОК В НАБОРЕ ЧТЕНИЙ НУКЛЕОТИДНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ

**А.В. Александров, С.В. Казаков, С.В. Мельников, А.А. Сергушичев, Ф.Н. Царев, А.А. Шалыто**

Задача секвенирования и сборки больших геномов является одной из актуальных задач современной биоинформатики. Исходными данными для сборки генома являются так называемые чтения, получаемые с помощью машин-секвенаторов. Эти чтения могут содержать ошибки, вызванные тем, что секвенирование основано на выполнении ряда химических реакций. Часто одним из этапов сборки генома является исправление ошибок в наборе чтений. Большинство методов исправления ошибок в чтениях строят граф де Брюина. Предлагаемый метод не использует граф де Брюина, что позволяет существенно сократить использование памяти. Метод реализован и опробован при сборке генома в рамках проекта «de novo Genome Assembly Project».

**Ключевые слова**: геном, секвенирование, сборка генома, исправление ошибок.

## Введение

Многие современные задачи биологии и медицины требуют знания генома живых организмов, который состоит из нескольких нуклеотидных последовательностей молекул дезоксирибонуклеиновой ки-

слоты (ДНК). В связи с этим возникает необходимость в дешевом и быстром методе секвенирования, т.е. определения последовательности нуклеотидов в образце ДНК.

Существующие технические средства (секвенаторы) не позволяют считать разом всю молекулу ДНК. Вместо этого они позволяют читать фрагменты генома небольшой длины. Длина фрагмента может варьироваться и является важным параметром секвенирования, так как от нее напрямую зависит стоимость секвенирования и время, затрачиваемое на чтение одного фрагмента: чем больше длина считываемого фрагмента, тем выше стоимость чтения и тем дольше это чтение происходит. В связи с этим сейчас получил распространение следующий дешевый и эффективный подход: сначала вычленяется случайно расположенный в геноме фрагмент длиной около 500 нуклеотидов, а затем считываются его префикс и суффикс (длиной около 80–120 нуклеотидов каждый). Эти префикс и суффикс называются *парными чтениями*. Этот процесс повторяется такое число раз, чтобы обеспечить достаточно большое покрытие генома чтениями.

Отметим, что описанные выше префикс и суффикс читаются с разных нитей ДНК: один – с прямой, другой – с обратно-комплементарной, причем неизвестно, который откуда. По этой причине удобно рассматривать геном и чтения, дополненные своими обратно-комплементарными копиями.

Описанный выше технологический процесс реализуется, например, секвенаторами компании Illumina [1]. При этом важной особенностью работы этих секвенаторов является возможность совершения ошибок при чтении. Это означает, что некоторые нуклеотиды секвенируются неверно (например, вместо нуклеотида A читается нуклеотид G). Ошибки замены – не единственный тип ошибок, допускаемый секвенаторами. Так, возможны также ошибки вставки и удаления. Однако эти ошибки встречаются в довольно специфических случаях и по сравнению с ошибками замены очень редки.

Кроме самой последовательности нуклеотидов, результатом работы секвенатора является информация о качестве прочтения каждого из нуклеотидов. По ней может быть вычислена вероятность того, что данный нуклеотид был прочитан неверно.

Для эффективной работы последующих стадий алгоритма очень важно исправить как можно больше ошибок в чтениях.

Существует большое число сборщиков, осуществляющих все или только некоторые из приведенных выше этапов. Все их можно разделить на две группы. Одни используют для исправления ошибок так называемый граф де Брюина [2], например ABySS [3], Velvet [4] или SOAPdenovo [5]. Другие проводят частотный анализ строк длины $k$ (их обычно называют $k$-мерами) – для каждого $k$-мера считают, сколько раз он встретился в чтениях, и на основе этих данных исправляют ошибки. Так работают, например, ALLPATHS [6] и EULER [7]. Все эти методы довольно требовательны к вычислительным ресурсам и не очень хорошо масштабируются.

Целью настоящей работы является разработка метода, лишенного указанных недостатков, который будет использоваться в качестве первого этапа сборки генома из набора парных чтений.

### Идея метода

Для эффективного исправления ошибок необходимо, чтобы каждая позиция генома была прочитана несколько раз, что, ввиду небольшой вероятности ошибки, дает право считать, что наибольшее число раз нуклеотид на каждой позиции был прочитан верно. На практике используются наборы чтений, покрывающие геном несколько десятков раз. Важно отметить, что не только отдельные позиции всего генома были прочитаны несколько десятков раз, но и небольшие его подстроки (не длиннее самих чтений) встречаются в чтениях несколько раз, причем чем длиннее подстрока, тем меньше шансов, что несколько различных чтений ее содержат. Последнее соображение вытекает не только из увеличения вероятности попадания чтения на конкретную подстроку при увеличении ее длины, но и из факта наличия в чтениях ошибок.

В работе рассматриваются строки одинаковой длины $k$ (так называемые $k$-меры). Для каждого $k$-мера, присутствующего в чтениях (прямых или обратно-комплементарных) хотя бы раз, вычислим, сколько раз он встречается в чтениях. На основании этой статистики все $k$-меры можно разделить на 2 группы – «надежные» $k$-меры и «подозрительные». «Надежные» $k$-меры – это те, которые встречаются в чтениях достаточно большое число раз – не меньше значения некоторого порога $t$, которое является параметром алгоритма и выбирается вручную. С «надежными» $k$-мерами делать ничего не нужно, предполагается, что в них ошибок нет. «Подозрительные» же $k$-меры вызваны или плохим покрытием тех частей генома, откуда они были прочитаны, либо, что гораздо более вероятно, наличием в них одной или нескольких ошибок, которые надлежит исправить.

После выделения «подозрительных» $k$-меров для каждого из них необходимо определить, в какой именно позиции могла быть совершена ошибка. Для этого предлагается перебрать все позиции $k$-мера (их ровно $k$) и все возможные нуклеотиды, попробовать заменить имеющийся нуклеотид на перебираемый и проанализировать получившийся $k$-мер. Если новый $k$-мер попадает в группу «надежных», значит, возможно, рассматриваемый $k$-мер является результатом его ошибочного прочтения. Если в течение пе-

ребора был найден только один «надежный» *k*-мер, получающийся из «подозрительного» путем замены одного нуклеотида на другой, полагается, что данный «подозрительный» *k*-мер исправлен, а соответствующее исправление запоминается. Если таких *k*-меров несколько, неясно, какое из исправлений запоминать, поэтому в таких случаях «подозрительные» *k*-меры не исправляются. И, наконец, если не было найдено ни одного способа, исправить «подозрительный» *k*-мер, полагается, что в нем было совершено больше одной ошибки, после чего запускается аналогичная процедура, но с исправлением не одного, а сразу двух нуклеотидов. Аналогичным образом можно пытаться изменить не только пары, но и тройки, а также кортежи из большего числа нуклеотидов, однако данное обобщение ощутимо сказывается на быстродействии алгоритма.

При выборе значения параметра *k* следует учитывать следующие соображения.

- Величина *k* должна быть значительно меньше длины чтений. Если длина *k*-мера будет сравнима с длиной чтения, то большинство *k*-меров будет встречаться в чтениях один раз, что не даст никакой информации для исправления ошибок.
- Величина *k* должна быть достаточно большой, чтобы вероятность того, что случайный *k*-мер заданной длины встречается в геноме, была ничтожно малой. В противном случае некоторые *k*-меры, содержащие ошибку, не будут исправлены только потому, что есть достаточно большая вероятность того, что эти *k*-меры на самом деле встречаются в геноме, а тогда они могут много раз встречаться в чтениях.

### Алгоритм исправления ошибок в чтениях

Для начала отметим, что качество нуклеотидов сильно ухудшается у конца чтения, поэтому для данного алгоритма имеет смысл обрезать куски чтений плохого качества, так как, с одной стороны, благодаря им образуется много «подозрительных» *k*-меров, с другой стороны, их невозможно исправить из-за большого числа ошибок.

Также важно заметить, что все *k*-меры можно разбить на группы на основании префикса небольшой длины, с которого они начинаются. Если при этом не исправлять ошибки в префиксе, то исправление не выводит *k*-меры из группы. Это не повлечет за собой неисправленные ошибки в префиксах, так как при обращении *k*-мера его префикс становится суффиксом.

Таким образом, алгоритм состоит из следующих шагов:

- обрезание *k*-меров на основании качества;
- разбиение *k*-меров на группы на основании префиксов;
- выполнение для каждой группы:
  - сбор статистики по *k*-мерам;
  - исправление ошибок.

Важно отметить, что алгоритм поиска ошибок в *k*-мерах легко распараллеливается, так как для обработки одного *k*-мера ему требуется только доступ на чтение к общей структуре данных, хранящей статистику по содержанию *k*-меров в чтениях, а также кратковременный доступ на запись для сохранения результата.

### Экспериментальные результаты

Описанный подход был разработан и применен в рамках проекта dnGASP [8]. В этом проекте участникам предлагалось восстановить синтетический геном, содержащий около 1,8 миллиардов нуклеотидов, который был покрыт чтениями 44 раза. Этот геном являлся смесью геномов различных живых организмов. При реализации алгоритма для хранения *k*-меров использовалась хэш-таблица с открытой адресацией [9]. Это позволило осуществлять добавление нового *k*-мера и проверку «надежности» нового за константное время (последнее особенно важно для распараллеливания алгоритма). Ввиду того, что в рамках этого конкурса чтения имели длину 114, было выбрано значение *k*, равное 30.

По частотному распределению *k*-меров (рисунок) было выбрано значение порога *t*, равное четырем. Алгоритм исправления ошибок работал на компьютере с четырьмя шетиядерными процессорами Intel® Xeon® E7450 с тактовой частотой 2,4 ГГц и с 24 ГБ оперативной памяти. До запуска алгоритма в исходных данных было 6,5 миллиардов различных *k*-меров, из которых 3 миллиарда «надежных». Алгоритм работал около 24 часов, после чего в данных стало 3,9 миллиардов (что на 40% меньше, чем в начале) различных *k*-меров, из которых 3,3 миллиарда «надежных».

### Заключение

Разработан метод исправления ошибок, основанный на частотном анализе *k*-меров. Проведено экспериментальное исследование метода, показавшее его работоспособность на данных, близких к реальным. Разработанный метод может эффективно использовать как достаточно малый, так и большой объем ресурсов. В настоящий момент исследуются возможные изменения, которые можно осуществить для улучшения эффективности и быстродействия предложенного метода. Например, планируется в

большей степени учитывать информацию о качестве прочитанных нуклеотидов, предоставляемую секвенаторами. Кроме того, рассматривается возможность учета взаимосвязи *k*-меров в чтениях.

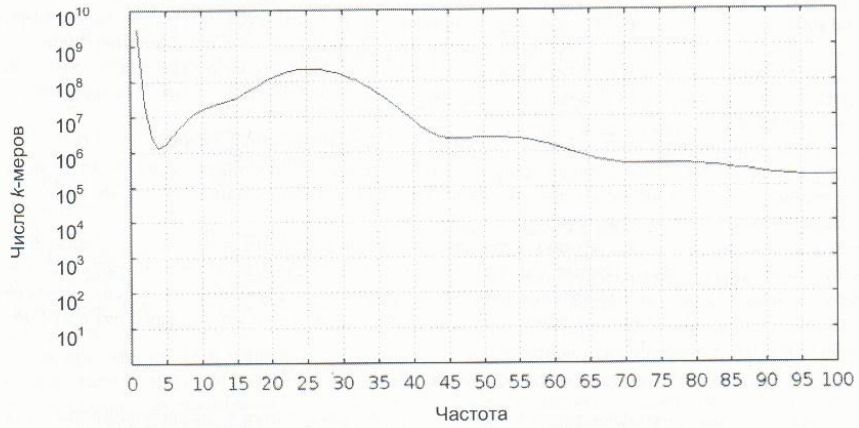



Рисунок. Зависимость числа *k*-меров от того, сколько раз они встречаются в чтениях

**Литература**

1. Illumina, Inc. [Электронный ресурс]. – Режим доступа: http://www.illumina.com/, свободный. Яз. англ. (дата обращения 28.05.2011).
2. Pevzner P. A 1-Tuple DNA sequencing: computer analysis // J. Biomol. Struct. Dyn. – 1989. – V. 7. – P. 63–73.
3. Simpson J.T., Wong K., Jackman S.D., Schein J.E., Jones S.J., Birol I. ABySS: A parallel assembler for short read sequence data // Genome Res. – 2009. – V. 19 – P. 1117–1123.
4. Zerbino D.R., Birney E. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs // Genome Res. – 2008. – V. 18 – P. 821–829.
5. Li R., Zhu H., Ruan J., Qian W., Fang X., Shi Z., Li Y., Li S., Shan G., Kristiansen K. et al. De novo assembly of human genomes with massively parallel short read sequencing // Genome Res. – 2010. – V. 20. – P. 265–272.
6. Butler J., MacCallum I., Kleber M., Shlyakhter I.A., Belmonte M.K., Lander E.S., Nusbaum C., Jaffe D.B. AllPaths: De novo assembly of wholegenome shotgun microreads // Genome Res. – 2008. – V. 18. – P. 810–820.
7. Pevzner P.A., Tang H., Waterman M.S. EULER: An Eulerian path approach to DNA fragment assembly // Proc. Natl. Acad. Sci. – 2001. – V. 98. – P. 9748–9753.
8. de novo Genome Assembly Project (dnGASP) [Электронный ресурс]. – Режим доступа: http://cnag.bsc.es/, свободный. Яз. англ. (дата обращения 28.05.2011).
9. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ: Пер. с англ. – 2-е изд. – М.: Вильямс, 2007. – 1296 с.

***Александров Антон Вячеславович*** – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, alexandrov@rain.ifmo.ru

***Казаков Сергей Владимирович*** – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, svkazakov@rain.ifmo.ru

***Мельников Сергей Вячеславович*** – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, melnikov@rain.ifmo.ru

***Сергушичев Алексей Александрович*** – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, студент, alserg@rain.ifmo.ru

***Царев Федор Николаевич*** – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, fedor.tsarev@gmail.com

***Шалыто Анатолий Абрамович*** – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

## ПРИЛОЖЕНИЕ 3. КОПИЯ СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММЫ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

# СВИДЕТЕЛЬСТВО

**о государственной регистрации программы для ЭВМ**

**№ 2011614454**

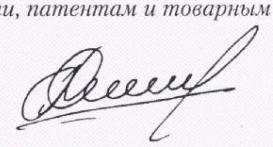
**Программное средство для удаления ошибок из набора чтений нуклеотидной последовательности**

Правообладатель(ли): ***Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» (RU)***

Автор(ы): ***Александров Антон Вячеславович, Исенбаев Владислав Вольдемарович, Казаков Сергей Владимирович, Мельников Сергей Вячеславович, Сергушичев Алексей Александрович, Царев Федор Николаевич (RU)***

Заявка № **2011612531**
Дата поступления **12 апреля 2011 г.**
Зарегистрировано в Реестре программ для ЭВМ
***6 июня 2011 г.***

*Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам*



*Б.П. Симонов*

# ПРИЛОЖЕНИЕ 4. ИСХОДНЫЙ КОД ПРОГРАММНОЙ РЕАЛИЗАЦИИ АЛГОРИТМА ИСПРАВЛЕНИЯ ОШИБОК

В данном приложении приведен исходный код программного средства для удаления ошибок из набора чтений нуклеотидной последовательности.

Cleaner.java

```
package ru.ifmo.genetics.tools.cleaner;

import java.io.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;
import java.util.TreeMap;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import ru.ifmo.genetics.Info;
import ru.ifmo.genetics.dna.DnaQ;
import ru.ifmo.genetics.framework.Dataset;
import ru.ifmo.genetics.framework.MultipleDataset;
import ru.ifmo.genetics.io.Source;
import ru.ifmo.genetics.io.formats.Illumina;
import ru.ifmo.genetics.io.formats.QualityFormat;
import ru.ifmo.genetics.statistics.Timer;
import ru.ifmo.genetics.tools.Util;
import ru.ifmo.genetics.tools.cleaner.fix.Fix;
import ru.ifmo.genetics.tools.cleaner.task.clean.CleanDispatcher;
import ru.ifmo.genetics.tools.cleaner.task.clean.CleanWorker;
import ru.ifmo.genetics.tools.cleaner.task.processHalfRead.GlobalContext;
import ru.ifmo.genetics.tools.cleaner.task.processHalfRead.LocalContext;
import ru.ifmo.genetics.tools.cleaner.task.processHalfRead.Task;

public class Cleaner {

    static int LEN = 30;
    static long MASK = (1L << (2 * LEN)) - 1;
    static int DISPATCH_WORK_RANGE_SIZE = 1 << 15;
    static int HALF_READ_TASK_SIZE = 1 << 11;
    static int STAT_WORK_THREADS_NUMBER = 16;

    static int CLEAN_WORK_THREADS_NUMBER = 20;

    static int CLEAN_WORK_QUEUE_CAPACITY = 2;

    static ArrayList<String> prefixes = new ArrayList<String>();
    // static int code = -1;
    // static String prefix;
    // static int prefixLen = -1;
    // static String prefixFile;

    static String DIR = null;
```

```
        static String BUCKETS = null;
        static String DATA = null;
        static File WORK_DIRECTORY = null;
        public final static QualityFormat ILUMINA = new Illumina();
        long time = System.currentTimeMillis();

        int countNoway;
        int countAmb;
        int countPolymorphism;
        int countFixes;
        Map<Integer, Integer> statAmb;
        Map<Integer, Integer> statPos;
        Map<Integer, Integer> statPosPair;
        int[] statNumb;

        ConcurrentHashMap<Long, Info> hm = new ConcurrentHashMap<Long, Info>((int)
1.5e8, 0.75f, STAT_WORK_THREADS_NUMBER + 1);

        void processSource(Source<DnaQ> source, boolean firstInPair, ExecutorService
pool, Collection<GlobalContext> envs) {
            int pairCounter = 0;
            LocalContext lastContext = new LocalContext(pairCounter, firstInPair,
                new ArrayList<DnaQ>(HALF_READ_TASK_SIZE));

            for (DnaQ dnaq : source) {
                pairCounter++;

                lastContext.dnaqs.add(dnaq);
                if (lastContext.dnaqs.size() == HALF_READ_TASK_SIZE) {
                    for (GlobalContext env: envs) {
                        pool.execute(new Task(env, lastContext));
                    }

                    lastContext = new LocalContext(pairCounter, firstInPair,
                        new ArrayList<DnaQ>(HALF_READ_TASK_SIZE));
                }

            }

            if (lastContext.dnaqs.size() != 0) {
                for (GlobalContext env: envs) {
                    pool.execute(new Task(env, lastContext));
                }
            }
        }

        Map<Long, Info> calcPrefix(List<String> prefixes, MultipleDataset
multipleDataSet) throws IOException, InterruptedException {
            System.err.println("start calcPrefix");
            Timer cpTimer = new Timer();

            int counter = 0;

            ExecutorService pool = Executors.newCachedThreadPool();

            int fileId = 0;
            for (Dataset dataset : multipleDataSet.datasets) {
                cpTimer.start();
                // int readCounter = 0;
```

```
                List<GlobalContext>             envs             =             new
ArrayList<GlobalContext>(prefixes.size());

                for (String prefix: prefixes) {
                    long prefixMask = Util.getPrefixMask(prefix, LEN);
                    long prefixCode = Util.getPrefixCode(prefix, LEN);
                    envs.add(new  GlobalContext(hm,  prefixMask,  prefixCode,  fileId,
LEN, MASK));
                }

                processSource(dataset.allFirsts(), true, pool, envs);
                processSource(dataset.allSeconds(), false, pool, envs);

                counter++;
                System.err.println("file = " + fileId + ", hm.size = " + hm.size() +
",  name  =  "  +  dataset.name()  +  ",  counter/dataset.size  =  "  +  counter  +  "/"  +
multipleDataSet.datasets.size());
                System.err.println("time = " + cpTimer.finish());
                ++fileId;
            }

            Util.shutdownAndAwaitTermination(pool);

            System.err.println("hm.size = " + hm.size());
            // System.err.println("sumTrustLen = " + sumTrustLength + ", sumLen = "
            // + sumLength + ", sumTrustLen/sumLen = " + ((double) sumTrustLength) /
            // sumLength);

            /*
             * PrintWriter out = new PrintWriter(BUCKETS + prefix); for
             * (Map.Entry<Long, Info> entry : hm.entrySet()) {
             * out.println(entry.getKey() + " " + entry.getValue()); } out.close();
             */

            System.err.println("end calcPrefix");

            return hm;
        }

        HashMap<Long, Info> readHashMapLongInfo(File file) throws IOException {
            BufferedReader br = new BufferedReader(new FileReader(file));
            HashMap<Long, Info> hm = new HashMap<Long, Info>();
            while (true) {
                String s = br.readLine();
                if (s == null)
                    break;
                StringTokenizer st = new StringTokenizer(s);
                hm.put(Long.parseLong(st.nextToken()), new Info(st));
            }
            return hm;
        }

        private void clean(Map<Long, Info> hm, Map<Fix, Fix> fixes) {
            System.err.println("start clean");
            CleanDispatcher      dispatcher      =      new      CleanDispatcher(hm,
DISPATCH_WORK_RANGE_SIZE);

            CleanWorker[] workersPool = new CleanWorker[CLEAN_WORK_THREADS_NUMBER];
```

```
                CountDownLatch latch = new CountDownLatch(workersPool.length);

                for (int i = 0; i < CLEAN_WORK_THREADS_NUMBER; ++i) {
                    workersPool[i] = new CleanWorker(dispatcher, hm, LEN, latch);
                    new Thread(workersPool[i]).start();
                }

                System.err.println("run");

                Timer t = new Timer();
                t.start();
                try {
                    latch.await();
                } catch (InterruptedException e) {
                    System.err.println("Main thread interrupted");
                    for (CleanWorker worker : workersPool) {
                        worker.interrupt();
                    }
                }
                System.err.println("Cleaning time = " + t.finish());

                for (CleanWorker worker : workersPool) {
                    for (Fix f: worker.getResults().values()) {
                        f.addToSet(fixes);
                    }
                    // fixes.addAll(worker.getResults());
                    countNoway += worker.countNoway;
                    countAmb += worker.countAmb;
                    countPolymorphism += worker.countPolymorphism;
                    countFixes += worker.countFixes;

                    for (Map.Entry<Integer, Integer> e : worker.statAmb.entrySet()) {
                        if (!statAmb.containsKey(e.getKey())) {
                            statAmb.put(e.getKey(), 0);
                        }
                        statAmb.put(e.getKey(), statAmb.get(e.getKey()) + e.getValue());
                    }

                    for (Map.Entry<Integer, Integer> e : worker.statPos.entrySet()) {
                        if (!statPos.containsKey(e.getKey())) {
                            statPos.put(e.getKey(), 0);
                        }
                        statPos.put(e.getKey(), statPos.get(e.getKey()) + e.getValue());
                    }

                    for (int i = 1; i <= worker.MAXIMAL_FIXES_NUMBER; i++) {
                        statNumb[i] += worker.statNumb[i];
                    }

                    /*
                    for (Map.Entry<Integer, Integer> e : worker.statPosPair.entrySet()) {
                        if (!statPosPair.containsKey(e.getKey())) {
                            statPosPair.put(e.getKey(), 0);
                        }
                        statPosPair.put(e.getKey(),     statPosPair.get(e.getKey())     +
e.getValue());
                    }
                    */
```

```
            }

            System.err.println("end clean");
        }

        void dumpGoodKMers(Map<Long, Info> hm, String filename) throws IOException {
            DataOutputStream out = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream(filename)));
            for (Map.Entry<Long, Info> e : hm.entrySet()) {
                if (e.getValue().count.intValue() >=
CleanWorker.DEFINITELY_GOOD_THRESHOLD) {
                    out.writeLong(e.getKey());
                }
            }
            out.close();
        }

        void stat(Map<Long, Info> hm) throws IOException {
            HashMap<Integer, Integer> stat = new HashMap<Integer, Integer>();
            for (Map.Entry<Long, Info> e : hm.entrySet()) {
                if (!stat.containsKey(e.getValue().count.intValue())) {
                    stat.put(e.getValue().count.intValue(), 0);
                }
                stat.put(e.getValue().count.intValue(),
stat.get(e.getValue().count.intValue()) + 1);
            }
            TreeMap<Integer, Integer> tm = new TreeMap<Integer, Integer>(stat);
            PrintWriter out = new PrintWriter(BUCKETS + File.separator +
Util.join(prefixes, "_") + ".stat");
            for (Map.Entry<Integer, Integer> e : tm.entrySet()) {
                out.println(e.getKey() + " " + e.getValue());
            }
            out.close();
        }

        private void run() throws IOException, ClassNotFoundException,
InterruptedException {
            countNoway = 0;
            countAmb = 0;
            countPolymorphism = 0;
            countFixes = 0;
            statAmb = new TreeMap<Integer, Integer>();
            statPos = new TreeMap<Integer, Integer>();
            statPosPair = new TreeMap<Integer, Integer>();
            statNumb = new int[CleanWorker.MAXIMAL_FIXES_NUMBER + 1];

            Map<Fix, Fix> allFixes = new HashMap<Fix, Fix>();
            MultipleDataset multipleDataSet = new MultipleDataset(new File(DATA),
true);
            /*
            int MAX_CODE = (1 << (2 * prefixLen));
            Timer bigTimer = new Timer();
            bigTimer.start();
            ArrayList<String> prefixes = new ArrayList<String>();
            BufferedReader br = new BufferedReader(new FileReader(prefixFile));
            while (true) {
                String prefix = br.readLine();
                if (prefix == null)
                    break;;
```

```
                prefixes.add(prefix);
            }
            */

            // double i = 0;
            // for (String prefix: prefixes)
            {
                // allFixes.clear();
                // ++i;
                Timer timer = new Timer();
                timer.start();
                System.err.println("prefixes = " + prefixes);
                Map<Long, Info> infos = calcPrefix(prefixes, multipleDataSet);
                System.err.println("reading took: " + timer.finish());
                stat(infos);
                clean(infos, allFixes);
                //   dumpGoodKMers(infos,   BUCKETS   +   Util.join(prefixes,   "_")   +
".kmers");
                System.err.println("current fixes = " + allFixes.size());
                System.err.println("noway = " + countNoway + ", amb = " + countAmb +
                        ",  fixes  =  "  +  countFixes  +  ",  polymorphism  =  "  +
countPolymorphism);

                System.err.println("StatAmb (numberOfFixes : count):");
                for (Map.Entry<Integer, Integer> e : statAmb.entrySet()) {
                    System.err.println(e.getKey() + " : " + e.getValue());
                }
                System.err.println();

                System.err.println("StatPos (numberOfPoses : count):");
                for (Map.Entry<Integer, Integer> e : statPos.entrySet()) {
                    System.err.println(e.getKey() + " : " + e.getValue());
                }
                System.err.println();

                System.err.println("StatPosPair (numberOfPoses : count):");
                for (Map.Entry<Integer, Integer> e : statPosPair.entrySet()) {
                    System.err.println(e.getKey() + " : " + e.getValue());
                }
                System.err.println("dumping prefixes");

                System.err.println("StatNumb:");
                for (int i = 1; i <= CleanWorker.MAXIMAL_FIXES_NUMBER; i++) {
                    System.err.println(i + ": " + statNumb[i]);
                }
                System.err.println();

                /*
                double done = i / prefixes.size();
                double total = bigTimer.finish() / done / 1000;
                double remained = total * (1 - done);
                */
                dumpFixes(allFixes.values(),  BUCKETS  +  Util.join(prefixes,  "_")  +
".fixes");
                // hm.clear();
                // System.gc();
                System.err.println("time for prefixes = " + timer.finish());
                /*
                System.err.println(100*done + "% done");
```

```
                    System.err.println("estimated  total time: " + total + "s, remained:
" + remained);
                    */
                    System.err.println();
                }
                // dumpFixes(allFixes);
                // ApplyFixes.applyFixes(DATA, multipleDataSet, allFixes.values(), true);
                // System.out.println("noway = " + countNoway + " amb = + " + countAmb);
            }

            public  void  dumpFixes(Collection<Fix>  fixes,  String  filename)  throws
IOException {
                // PrintWriter out = new PrintWriter(new FileOutputStream(DATA + "fixes",
true));
                PrintWriter out = new PrintWriter(filename);
                for (Fix f : fixes) {
                    out.println(f.dumpToString());
                }
                out.close();
            }

            public static void main(String[] args) throws Exception {
                if (args.length < 3) {
                    System.err.println("Usage: FastqCleaner <dir> <LEN> <prefix>+");
                    System.exit(666);
                }

            for (int i = 2; i < args.length; ++i) {
                    prefixes.add(args[i]);
                }
                // prefix = args[0];
                // prefixFile = args[0];

                DIR = args[0];
                DATA = DIR + File.separator;
                BUCKETS = DIR + File.separator + "buckets" + File.separator;
                WORK_DIRECTORY = new File(DIR, "work");

                if (args.length > 2) {
                    LEN = Integer.parseInt(args[1]);
                    MASK = (1L << (2 * LEN)) - 1;
                }

                long time = System.currentTimeMillis();
                new Cleaner().run();
                System.err.println("total  time  =  "  +  (System.currentTimeMillis()  -
time));
            }
```

ApplyFixes.java

```
package ru.ifmo.genetics.tools.cleaner.fix;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
```

```
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Map;
import java.util.HashMap;

import ru.ifmo.genetics.dna.DnaQ;
import ru.ifmo.genetics.dna.DnaTools;
import ru.ifmo.genetics.framework.Dataset;
import ru.ifmo.genetics.framework.MultipleDataset;
import ru.ifmo.genetics.io.Source;
import ru.ifmo.genetics.tools.Util;

public class ApplyFixes {
    public static void main(String[] args) throws IOException {
        if (args.length < 3) {
            System.err.println("Using: java ApplyFixes <work-dir> <is-binary>
<fixes-file-name>*");
            return;
        }

        String workDir = args[0];
            boolean isBinary = Boolean.parseBoolean(args[1]);

            String[] fixesFiles = Arrays.copyOfRange(args, 2, args.length);

        applyFixes(workDir, fixesFiles, isBinary);
        }

    static void applyFixes(String workDir, String[] fixesFiles, boolean isBinary)
throws IOException {
            String dataFile = workDir + File.separator;

        MultipleDataset multipleDataSet = new MultipleDataset(new File(dataFile),
isBinary);

        System.err.println("Loading fixes");
            Collection<Fix> allFixes = Fix.loadFixes(fixesFiles);

        applyFixes(workDir, multipleDataSet, allFixes, isBinary);
        }

    static void applyFixes(String workDir, MultipleDataset multipleDataSet,
Collection<Fix> allFixes, boolean isBinary) throws IOException {
            System.err.println("Dividing fixes by files");
            List<Fix>[][] byFiles = getByFiles(allFixes,
multipleDataSet.datasets.size());

        File fixedDir = new File(workDir, "fixedReads");
            fixedDir.mkdir();

        for (int i = 0; i < multipleDataSet.datasets.size(); i++) {
                Dataset dataset = multipleDataSet.datasets.get(i);
                for (int j = 0; j < 2; j++) {
```

```
                    System.err.println("fixing " + dataset.name() + "_" + (j + 1) +
                            ", " + byFiles[i][j].size() + " fixes" +
                            ",  total  "  +  (2  *  i  +  j)  +  "/"  +  2  *
multipleDataSet.datasets.size());

            File sourceFile = dataset.getFileByIndex(j);
                    File targetFile = new File(fixedDir, sourceFile.getName());

            if (isBinary) {
                        Source<DnaQ>  in  =  (j  ==  0)  ?  dataset.allFirsts()  :
dataset.allSeconds();
                        OutputStream    out    =    new    BufferedOutputStream(new
FileOutputStream(targetFile));
                        applyFixesBinary(in, out, byFiles[i][j]);
                    } else {
                        applyFixesNotBinary(sourceFile, targetFile, byFiles[i][j]);
                    }
                }
            }

          System.err.println("Done");
          }

        static List<Fix>[][] getByFiles(Collection<Fix> allFixes, int filesCount) {
            List<Fix>[][] byFiles = new ArrayList[filesCount][2];
            for (int i = 0; i < byFiles.length; i++) {
                for (int j = 0; j < 2; j++) {
                    byFiles[i][j] = new ArrayList<Fix>();
                }
            }
            for (Fix fix : allFixes) {
                byFiles[fix.file][fix.isFirst ? 0 : 1].add(fix);
            }
            for (int i = 0; i < byFiles.length; i++) {
                for (int j = 0; j < 2; j++) {
                    List<Fix> curFixes = byFiles[i][j];
                    Collections.sort(curFixes, new Comparator<Fix>() {
                        @Override
                        public int compare(Fix f1, Fix f2) {
                            if (f1.line != f2.line) {
                                return f1.line - f2.line;
                            }
                            return f1.shift - f2.shift;
                        }
                    });

                    List<Fix> newFixes = new ArrayList<Fix>();
                    int fixN = 0;
                    while (fixN < curFixes.size()) {
                        Fix fix = curFixes.get(fixN);
                        fixN++;
                        while ((fixN < curFixes.size()) &&
                                (curFixes.get(fixN).line == fix.line) &&
                                (curFixes.get(fixN).shift == fix.shift)) {

                Fix nFix = curFixes.get(fixN);
                            fixN++;
                            if (nFix.from != fix.from) {
                                System.err.println("Bad fixes:");
```

```
                                System.err.println(fix);
                                System.err.println(nFix);
                            } else if (nFix.to != fix.to) {
                                fix = fix.getBetter(nFix);
                            }
                        }

                newFixes.add(fix);
                    }

                    byFiles[i][j] = newFixes;
                }
            }

            return byFiles;
        }

        static void applyFixesNotBinary(File sourceFile, File targetFile, List<Fix>
fixes) throws IOException {
            BufferedReader in = new BufferedReader(new FileReader(sourceFile));
            PrintWriter out = new PrintWriter(targetFile);

            int line = 0;
            int fixN = 0;
            while (true) {
                String head = in.readLine();
                if (head == null) {
                    break;
                }
                out.println(head);

                char[] data = in.readLine().toCharArray();
                for (; fixN < fixes.size() && fixes.get(fixN).line == line; fixN++) {
                    Fix fix = fixes.get(fixN);
                    if (data[fix.shift] != fix.from) {
                        System.err.println("Incorrect fix: ");
                        System.err.println(fix);
                        System.err.println(new String(data));
                    } else {
                        data[fix.shift] = fix.to;
                        /*
                        System.err.println("Fix applied: ");
                        System.err.println(fix);
                        */
                    }
                }
                out.println(data);

                out.println(in.readLine());
                out.println(in.readLine());

                line++;
            }

            in.close();
            out.close();

        if (fixN < fixes.size()) {
```

```
                System.err.println("Too much fixes : " + (fixes.size() - fixN) + "
last fixes didn't apply");
            }
        }

       static void applyFixesBinary(Source<DnaQ> in, OutputStream out, List<Fix>
fixes) throws IOException {
            int line = 0;
            int fixN = 0;
            for (DnaQ dnaq : in) {
                for (; fixN < fixes.size() && fixes.get(fixN).line == line; fixN++) {
                    Fix fix = fixes.get(fixN);

                    byte b = dnaq.byteAt(fix.shift);

            if ((b & 3) != DnaTools.fromChar(fix.from)) {
                        System.err.println("Incorrect fix: ");
                        System.err.println(fix);
                    } else {
                        b = (byte) (b ^ (b & 3) ^ DnaTools.fromChar(fix.to));
                        dnaq.setByte(fix.shift, b);
                        /*
                        System.err.println("Fix applied: ");
                        System.err.println(fix);
                        */
                    }
                }

          Util.putByteArray(dnaq.toByteArray(), out);

          line++;
            }

        out.close();

        if (fixN < fixes.size()) {
                System.err.println("Too much fixes : " + (fixes.size() - fixN) + "
last fixes didn't apply");
            }
        }
```

CompareFixes.java

```
package ru.ifmo.genetics.tools.cleaner.fix;
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Collection;

import ru.ifmo.genetics.statistics.QuantitativeStatistics;

public class CompareFixes {
    static int fileCount = 100;

  public static void main(String[] args) throws IOException {
        String file1 = args[0];
```

```
            String[] files2 = Arrays.copyOfRange(args, 1, args.length);

            System.err.println("Loading fixes");
            Collection<Fix> lf1 = Fix.loadFixes(file1);
            Collection<Fix> lf2 = Fix.loadFixes(files2);

        System.err.println("Dividing fixes by files");
            List<Fix>[][] bf1 = ApplyFixes.getByFiles(lf1, fileCount);
            List<Fix>[][] bf2 = ApplyFixes.getByFiles(lf2, fileCount);

        System.err.println("Comparing");
            QuantitativeStatistics<Integer>           stat           =           new
QuantitativeStatistics<Integer>(true);

        for (int i = 0; i < fileCount; i++) {
                for (int j = 0; j < 2; j++) {
                    if (bf1[i][0].size() != 0) {
    //                    System.err.println();
                        System.err.println("File " + i + "_" + (j + 1));

                HashMap<Fix, Fix> fs = new HashMap<Fix, Fix>();
                        for (Fix f : bf1[i][j]) {
                            fs.put(f, f);
                        }

                int good = 0;
                        int wrong = 0;
                        int newFixes = 0;

                for (Fix f : bf2[i][j]) {
                            Fix rf = fs.get(f);
                            if (rf == null) {
    //                            stat.add(f.weight);
                                newFixes++;
                            } else {
                                if (rf.to == f.to) {
    //                                stat.add(f.weight);
                                    good++;
                                } else {
    //                                stat.add(f.weight);
                                    wrong++;
                                }
                                fs.remove(f);
                            }
                        }

                System.err.println("Good = " + good + ", wrong = " + wrong + ", new
fixes = " + newFixes

                                    + ", not found = " + fs.size());
                    }
                }
            }

            stat.print();
        }
```

Fix.java

```
package ru.ifmo.genetics.tools.cleaner.fix;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.FileReader;
import java.util.Map;
import java.util.StringTokenizer;
import java.util.Collection;
import java.util.HashMap;

public class Fix {
    public int file;
    public int line;
    public int shift;
    public boolean isFirst;
    public char from;
    public char to;
    public int weight;

    public Fix(int file, int line, int shift, boolean isFirst, char from, char
to, int weight) {
        this.file = file;
        this.line = line;
        this.shift = shift;
        this.isFirst = isFirst;
        this.from = from;
        this.to = to;
        this.weight = weight;
    }

    public Fix(String s) {
        StringTokenizer st = new StringTokenizer(s);
        file = Integer.parseInt(st.nextToken());
        line = Integer.parseInt(st.nextToken());
        shift = Integer.parseInt(st.nextToken());
        isFirst = Boolean.parseBoolean(st.nextToken());
        from = st.nextToken().charAt(0);
        to = st.nextToken().charAt(0);
        weight = Integer.parseInt(st.nextToken());
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + file;
        result = prime * result + line;
        result = prime * result + shift;
        result = prime * result + (isFirst ? 1231 : 1237);
        result = prime * result + from;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
```

```
                return false;
            Fix other = (Fix) obj;
            if (file != other.file)
                return false;
            if (line != other.line)
                return false;
            if (shift != other.shift)
                return false;
            if (isFirst != other.isFirst)
                return false;
            if (from != other.from)
                return false;
            return true;
        }


        public String dumpToString() {
            return file + " " + line + " " + shift + " " + isFirst + " " + from + " "
+ to + " " + weight;
        }

        @Override
        public String toString() {
            return "file: " + file + "; line: " + line + "; shift: " + shift + ";
isFirst: " +
                isFirst + "; from: " + from + "; to: " + to + "; weight: " + weight;
        }

      public Fix getBetter(Fix other) {
            if ((other == null) || (other.weight < weight)) {
                return this;
            }
            return other;
        }

        public boolean addToSet(Map<Fix, Fix> fixes) {
            Fix old = fixes.get(this);
            if (old == null || old.weight < weight) {
                fixes.put(this, this);
                return true;
            }
            return false;
        }

      public static Collection<Fix> loadFixes(String... files) throws IOException {
            Map<Fix, Fix> fixes = new HashMap<Fix, Fix>();

            for (String file : files) {
                BufferedReader in = new BufferedReader(new FileReader(file));
                while (in.ready()) {
                    new Fix(in.readLine()).addToSet(fixes);
                }
                System.err.println("loading from " + file + " done, fixes.size =  " +
fixes.size());
                in.close();
            }

            return fixes.values();
        }
```

fix/MergeFixes.java

```
package ru.ifmo.genetics.tools.cleaner.fix;
import java.util.Collection;
import java.io.IOException;

public class MergeFixes {
    public static void main(String[] args) throws IOException {
        Collection<Fix> fixes = Fix.loadFixes(args);
        long size = fixes.size();
        System.err.println("fixes.size = " + size);
        long done = 0;
        for (Fix f : fixes) {
            System.out.println(f.dumpToString());
            ++done;
            if ((done & 0xfffff) == 0) {
                System.err.print("\r" + done + "/" + size + " done");
            }
        }
        System.err.println();
    }
```

KmersDumper.java

```
package ru.ifmo.genetics.tools.cleaner;

import java.io.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.StringTokenizer;
import java.util.TreeMap;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import ru.ifmo.genetics.Info;
import ru.ifmo.genetics.dna.DnaQ;
import ru.ifmo.genetics.framework.Dataset;
import ru.ifmo.genetics.framework.MultipleDataset;
import ru.ifmo.genetics.io.Source;
import ru.ifmo.genetics.io.formats.Illumina;
import ru.ifmo.genetics.io.formats.QualityFormat;
import ru.ifmo.genetics.statistics.Timer;
import ru.ifmo.genetics.tools.Util;
import ru.ifmo.genetics.tools.cleaner.task.clean.CleanWorker;
import ru.ifmo.genetics.tools.cleaner.task.processHalfRead.GlobalContext;
import ru.ifmo.genetics.tools.cleaner.task.processHalfRead.LocalContext;
import ru.ifmo.genetics.tools.cleaner.task.processHalfRead.Task;

public class KmersDumper {

    static int LEN = 30;
    static long MASK = (1L << (2 * LEN)) - 1;
```

```
        static int DISPATCH_WORK_RANGE_SIZE = 1 << 15;
        static int HALF_READ_TASK_SIZE = 1 << 11;
        static int STAT_WORK_THREADS_NUMBER = 16;

        static ArrayList<String> prefixes = new ArrayList<String>();
        // static int code = -1;
        // static String prefix;
        // static int prefixLen = -1;
        // static String prefixFile;

        static String DIR = null;
        static String BUCKETS = null;
        static String DATA = null;
        static File WORK_DIRECTORY = null;
        public final static QualityFormat ILUMINA = new Illumina();
        long time = System.currentTimeMillis();

        int countNoway;
        int countAmb;
        int countPolymorphism;
        int countFixes;
        Map<Integer, Integer> statAmb;
        Map<Integer, Integer> statPos;
        Map<Integer, Integer> statPosPair;
        int[] statNumb;

        ConcurrentHashMap<Long, Info> hm = new ConcurrentHashMap<Long, Info>((int)
1.5e8, 0.75f, STAT_WORK_THREADS_NUMBER + 1);

        void processSource(Source<DnaQ> source, boolean firstInPair, ExecutorService
pool, Collection<GlobalContext> envs) {
            int pairCounter = 0;
            LocalContext lastContext = new LocalContext(pairCounter, firstInPair,
                new ArrayList<DnaQ>(HALF_READ_TASK_SIZE));

            for (DnaQ dnaq : source) {
                pairCounter++;

                lastContext.dnaqs.add(dnaq);
                if (lastContext.dnaqs.size() == HALF_READ_TASK_SIZE) {
                    for (GlobalContext env: envs) {
                        pool.execute(new Task(env, lastContext));
                    }

                    lastContext = new LocalContext(pairCounter, firstInPair,
                        new ArrayList<DnaQ>(HALF_READ_TASK_SIZE));
                }

            }

            if (lastContext.dnaqs.size() != 0) {
                for (GlobalContext env: envs) {
                    pool.execute(new Task(env, lastContext));
                }
            }
        }

        Map<Long, Info> calcPrefix(List<String> prefixes, MultipleDataset
multipleDataSet) throws IOException, InterruptedException {
```

```
            System.err.println("start calcPrefix");
            Timer cpTimer = new Timer();

            int counter = 0;

            ExecutorService pool = Executors.newCachedThreadPool();

            int fileId = 0;
            for (Dataset dataset : multipleDataSet.datasets) {
                cpTimer.start();
                // int readCounter = 0;

                List<GlobalContext> envs = new
ArrayList<GlobalContext>(prefixes.size());

                for (String prefix: prefixes) {
                    long prefixMask = Util.getPrefixMask(prefix, LEN);
                    long prefixCode = Util.getPrefixCode(prefix, LEN);
                    envs.add(new GlobalContext(hm, prefixMask, prefixCode, fileId,
LEN, MASK));
                }

                processSource(dataset.allFirsts(), true, pool, envs);
                processSource(dataset.allSeconds(), false, pool, envs);

                counter++;
                System.err.println("file = " + fileId + ", hm.size = " + hm.size() +
", name = " + dataset.name() + ", counter/dataset.size = " + counter + "/" +
multipleDataSet.datasets.size());
                System.err.println("time = " + cpTimer.finish());
                ++fileId;
            }

            Util.shutdownAndAwaitTermination(pool);

            System.err.println("hm.size = " + hm.size());
            // System.err.println("sumTrustLen = " + sumTrustLength + ", sumLen = "
            // + sumLength + ", sumTrustLen/sumLen = " + ((double) sumTrustLength) /
            // sumLength);

            /*
             * PrintWriter out = new PrintWriter(BUCKETS + prefix); for
             * (Map.Entry<Long, Info> entry : hm.entrySet()) {
             * out.println(entry.getKey() + " " + entry.getValue()); } out.close();
             */

            System.err.println("end calcPrefix");

            return hm;
        }

        HashMap<Long, Info> readHashMapLongInfo(File file) throws IOException {
            BufferedReader br = new BufferedReader(new FileReader(file));
            HashMap<Long, Info> hm = new HashMap<Long, Info>();
            while (true) {
                String s = br.readLine();
                if (s == null)
                    break;
                StringTokenizer st = new StringTokenizer(s);
```

```
                hm.put(Long.parseLong(st.nextToken()), new Info(st));
            }
            return hm;
        }

        void dumpGoodKMers(Map<Long, Info> hm, String filename) throws IOException {
            DataOutputStream out = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream(filename)));
            for (Map.Entry<Long, Info> e : hm.entrySet()) {
                if (e.getValue().count.intValue() >=
CleanWorker.DEFINITELY_GOOD_THRESHOLD) {
                    out.writeLong(e.getKey());
                }
            }
            out.close();
        }

        void stat(Map<Long, Info> hm) throws IOException {
            HashMap<Integer, Integer> stat = new HashMap<Integer, Integer>();
            for (Map.Entry<Long, Info> e : hm.entrySet()) {
                if (!stat.containsKey(e.getValue().count.intValue())) {
                    stat.put(e.getValue().count.intValue(), 0);
                }
                stat.put(e.getValue().count.intValue(),
stat.get(e.getValue().count.intValue()) + 1);
            }
            TreeMap<Integer, Integer> tm = new TreeMap<Integer, Integer>(stat);
            PrintWriter out = new PrintWriter(BUCKETS + File.separator +
Util.join(prefixes, "_") + ".stat");
            for (Map.Entry<Integer, Integer> e : tm.entrySet()) {
                out.println(e.getKey() + " " + e.getValue());
            }
            out.close();
        }

        private void run() throws IOException, ClassNotFoundException,
InterruptedException {
            countNoway = 0;
            countAmb = 0;
            countPolymorphism = 0;
            countFixes = 0;
            statAmb = new TreeMap<Integer, Integer>();
            statPos = new TreeMap<Integer, Integer>();
            statPosPair = new TreeMap<Integer, Integer>();
            statNumb = new int[CleanWorker.MAXIMAL_FIXES_NUMBER + 1];

            MultipleDataset multipleDataSet = new MultipleDataset(new File(DATA),
true);

            {
                Timer timer = new Timer();
                timer.start();
                System.err.println("prefixes = " + prefixes);
                Map<Long, Info> infos = calcPrefix(prefixes, multipleDataSet);
                System.err.println("reading took: " + timer.finish());
                stat(infos);
                dumpGoodKMers(infos, BUCKETS + Util.join(prefixes, "_") + ".kmers");
                System.err.println("noway = " + countNoway + ", amb = " + countAmb +
```

```
                        ",  fixes  =  "  +  countFixes  +  ",  polymorphism  =  "  +
countPolymorphism);

                System.err.println("StatAmb (numberOfFixes : count):");
                for (Map.Entry<Integer, Integer> e : statAmb.entrySet()) {
                    System.err.println(e.getKey() + " : " + e.getValue());
                }
                System.err.println();

                System.err.println("StatPos (numberOfPoses : count):");
                for (Map.Entry<Integer, Integer> e : statPos.entrySet()) {
                    System.err.println(e.getKey() + " : " + e.getValue());
                }
                System.err.println();

                System.err.println("StatPosPair (numberOfPoses : count):");
                for (Map.Entry<Integer, Integer> e : statPosPair.entrySet()) {
                    System.err.println(e.getKey() + " : " + e.getValue());
                }
                System.err.println("dumping prefixes");

                System.err.println("StatNumb:");
                for (int i = 1; i <= CleanWorker.MAXIMAL_FIXES_NUMBER; i++) {
                    System.err.println(i + ": " + statNumb[i]);
                }
                System.err.println();

                System.err.println("time for prefixes = " + timer.finish());
                System.err.println();
            }
        }

        public static void main(String[] args) throws Exception {
            if (args.length < 3) {
                System.err.println("Usage: KmersDumper <dir> <LEN> <prefix>+");
                System.exit(666);
            }

        for (int i = 2; i < args.length; ++i) {
                prefixes.add(args[i]);
            }
            // prefix = args[0];
            // prefixFile = args[0];

            DIR = args[0];
            DATA = DIR + File.separator;
            BUCKETS = DIR + File.separator + "buckets" + File.separator;
            WORK_DIRECTORY = new File(DIR, "work");

            if (args.length > 2) {
                LEN = Integer.parseInt(args[1]);
                MASK = (1L << (2 * LEN)) - 1;
            }

            long time = System.currentTimeMillis();
            new KmersDumper().run();
            System.err.println("total  time  =  "  +  (System.currentTimeMillis()  -
time));
        }
```

task/clean/CleanDispatcher.java

```
package ru.ifmo.genetics.tools.cleaner.task.clean;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.atomic.AtomicInteger;

import ru.ifmo.genetics.Info;

public class CleanDispatcher {
    public static int BAD_SUBSTRING_THRESHOLD = 3;

    public CleanDispatcher(Map<Long, Info> hm, int workRangeSize) {
        tasks = new ArrayList<List<Long>>();

        List<Long> task = new ArrayList<Long>(workRangeSize);
        for (Map.Entry<Long, Info> e : hm.entrySet()) {
            long str = e.getKey();
            Info info = e.getValue();
            if (info.count.intValue() <= BAD_SUBSTRING_THRESHOLD) {
                task.add(str);
                if (task.size() == workRangeSize) {
                    tasks.add(task);
                    task = new ArrayList<Long>(workRangeSize);
                }
            }
        }

        if (task.size() != 0) {
            tasks.add(task);
        }

        System.err.println("clean tasks size = " + tasks.size());

        index = new AtomicInteger();
    }

    List<List<Long>> tasks;
    AtomicInteger index;

    public List<Long> getWorkRange() {
        int curIndex = index.getAndIncrement();
        if (curIndex < tasks.size()) {
            return tasks.get(curIndex);
        } else {
            return null;
        }
    }
```

CleanWorker.java

```
package ru.ifmo.genetics.tools.cleaner.task.clean;
import java.util.Collection;
import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.List;
import java.util.Map;
import java.util.concurrent.CountDownLatch;

import ru.ifmo.genetics.Info;
import ru.ifmo.genetics.dna.DnaTools;
import ru.ifmo.genetics.tools.Util;
import ru.ifmo.genetics.tools.cleaner.fix.Fix;

public class CleanWorker implements Runnable {

    public final static int DEFINITELY_GOOD_THRESHOLD = 4;

    public final static int MAXIMAL_FIXES_NUMBER = 3;
    public final static int MAXIMAL_FIXES_DISTANCE = 1;


    private CleanDispatcher d;
    private Map<Fix, Fix> fixes = new HashMap<Fix, Fix>();
    private Map<Long, Info> hm;
    private final int len;
    private long newStr;

    public int countFixes = 0;
    public int countNoway = 0;
    public int countAmb = 0;
    public int countPolymorphism = 0;
    public Map<Integer, Integer> statAmb;
    public Map<Integer, Integer> statPos;
    public int[] statNumb;
    // public Map<Integer, Integer> statPosPair;
    CountDownLatch latch;
    boolean interrupted = false;

    public CleanWorker(CleanDispatcher d, Map<Long, Info> hm, int len,
CountDownLatch latch) {
        this.d = d;
        this.hm = hm;
        this.len = len;
        this.latch = latch;
        statAmb = new HashMap<Integer, Integer>();
        statPos = new HashMap<Integer, Integer>();
        statNumb = new int[MAXIMAL_FIXES_NUMBER + 1];
        // statPosPair = new HashMap<Integer, Integer>();
    }

    public Map<Fix, Fix> getResults() {
        return fixes;
    }

    public int getCountNoway() {
        return countNoway;
    }

    public int getCountAmb() {
        return countAmb;
    }

    public void interrupt() {
```

```
            interrupted = true;
        }

        int numberOfFixes;
        int numberOfPoses;
        // int numberOfPosesPairs;

        private Fix createFix(long str, Info info, int pos, long xor, int weight) {
            long nstr = str ^ (xor << (2 * pos));
            byte fromNuc = (byte) ((str >> (2 * pos)) & 3);
            byte toNuc = (byte) ((nstr >> (2 * pos)) & 3);
            int shift;
            if (!info.reversedComplemented) {
                shift = (info.shift + len - 1 - pos);
            } else {
                shift = (info.shift - len + 1 + pos);
                fromNuc ^= 3;
                toNuc ^= 3;
            }
            char from = DnaTools.toChar(fromNuc);
            char to = DnaTools.toChar(toNuc);
            return new Fix(info.file, info.line, shift, info.firstInPair, from, to,
weight);
        }

        private Collection<Fix> generateFixes(long oldStr, long newStr) {
            Collection<Fix> ans = new ArrayList<Fix>();
            Info info = hm.get(oldStr);
            Info ninfo = hm.get(newStr);
            for (int pos = 0; pos < len; pos++) {
                long xor = ((oldStr ^ newStr) >> (2 * pos)) & 3;
                if (xor != 0) {
                    ans.add(createFix(oldStr,         info,         pos,         xor,
ninfo.count.intValue()));
                }
            }
            return ans;
        }

        private Long findFixes(long str) {
            for (int n = 1; n <= MAXIMAL_FIXES_NUMBER; n++) {
                int r = findNFixes(str, n, 0);
                if (r == 1) {
                    statNumb[n]++;
                    return newStr;
                }
            }
            return null;
        }

        private int findNFixes(long str, int n, int begin) {
            if (n == 0) {
                Info minfo = hm.get(str);
                if    ((minfo    !=    null)    &&    (minfo.count.intValue()    >=
DEFINITELY_GOOD_THRESHOLD)) {
                    newStr = str;
                    return 1;
                } else {
                    return 0;
```

```
                }
            }

            int k = 0;
            int maxPos = (begin == 0) ? (len - 1) : begin + MAXIMAL_FIXES_DISTANCE;
            maxPos = Math.min(maxPos, len - n);
            for (int pos = begin; (k < 2) && (pos <= maxPos); pos++) {
                for (long xor = 1; xor <= 3; xor++) {
                    long nstr = str ^ (xor << (2 * pos));
                    int r = findNFixes(nstr, n - 1, pos + 1);
                    if (r == 2) {
                        k = 2;
                        break;
                    }
                    if (r == 0) {
                        continue;
                    }
                    k++;
                    if (k == 2) {
                        break;
                    }
                }
            }

            return k;

        }

        @Override
        public void run() {
            while (!interrupted) {
                List<Long> p = d.getWorkRange();
                if (p == null)
                    break;
                for (long str : p) {
                    numberOfFixes = 0;
                    numberOfPoses = 0;

                    Long nstr = findFixes(str);
                    if (nstr == null) {
                        countNoway++;
                        continue;
                    }
                    Collection<Fix> c = generateFixes(str, nstr);
                    for (Fix f : c) {
                        f.addToSet(fixes);
                        countFixes++;
                    }

                    // Util.incrementInt(statPos, numberOfPoses);
                    // Util.incrementInt(statPosPair, numberOfPosesPairs);


            /*
                    if (numberOfPoses == 1) {
                        f.addToSet(fixes);
                        countFixes++;
                        if (numberOfFixes > 1) {
```

```
                            countPolymorphism++;
                        }
                    } else if (numberOfPoses == 0) {
                        countNoway++;
                    } else if (numberOfPoses > 1) {
                        countAmb++;

                        Util.incrementInt(statAmb, numberOfPoses);
                    }
                    */
                }
            }
            latch.countDown();
        }
```

GlobalContext.java

```
package ru.ifmo.genetics.tools.cleaner.task.processHalfRead;
import java.util.concurrent.*;

import ru.ifmo.genetics.Info;

public class GlobalContext {
      public ConcurrentHashMap<Long, Info> hm;
    public long prefixMask;
    public long prefixCode;
    public int fileId;
    public int LEN;
    public long MASK;

      public  GlobalContext(ConcurrentHashMap<Long,  Info>  hm,  long  prefixMask,
long prefixCode, int fileId,
                    int LEN, long MASK) {
        this.hm = hm;
        this.prefixMask = prefixMask;
        this.prefixCode = prefixCode;
        this.fileId = fileId;
        this.LEN = LEN;
        this.MASK = MASK;
    }
```

task/processHalfRead/LocalContext.java

```
package ru.ifmo.genetics.tools.cleaner.task.processHalfRead;

import java.util.List;

import ru.ifmo.genetics.dna.DnaQ;

public class LocalContext {
    public int it0;
    public boolean firstInPair;
    public List<DnaQ> dnaqs;

    public LocalContext(int it0, boolean firstInPair, List<DnaQ> dnaqs) {
```

```
            this.it0 = it0;
            this.firstInPair = firstInPair;
            this.dnaqs = dnaqs;
        }


    task/processHalfRead/Task.java

    package ru.ifmo.genetics.tools.cleaner.task.processHalfRead;
    import ru.ifmo.genetics.Info;
    import ru.ifmo.genetics.dna.DnaQ;

    public class Task implements Runnable {
        GlobalContext env;
        LocalContext local;

        public Task(GlobalContext env, LocalContext local) {
            this.env = env;
            this.local = local;
        }

        private void addDnaQToStat(DnaQ dnaq, int it, boolean reversedComplemented) {
            long cur = 0;
            for (int i = 0; i < env.LEN - 1; i++) {
                cur = cur << 2 | dnaq.nucAt(i);
            }
            // String s = dnaq.toString();
            for (int i = env.LEN - 1; i < dnaq.length(); i++) {
                cur = cur & (env.MASK >> 2);
                cur = (cur << 2) | dnaq.nucAt(i);
                // String t = s.substring(i - LEN + 1, i + 1);
                // assert cur == getCode(t) : t;
                // assert t.startsWith(prefix) == ((prefixMask & cur) == prefixCode)
                // : t + " " + (prefixCode & cur) + " "
                // + prefixMask;
                if    (((env.prefixMask    &    cur)    ==    env.prefixCode))    {    //
t.startsWith(prefix)
                    if (!env.hm.containsKey(cur)) {
                        int index;
                        if (!reversedComplemented) {
                            index = i - env.LEN + 1;
                        } else {
                            index = dnaq.length() - 1 - (i - env.LEN + 1);
                        }
                        env.hm.putIfAbsent(cur,   new   Info(env.fileId,   it,   index,
local.firstInPair, reversedComplemented));
                    }
                    env.hm.get(cur).count.incrementAndGet();
                }
            }
        }

        void process(DnaQ dnaq, int it) {
            if (dnaq.length() < env.LEN) {
                return;
            }
            // System.out.println("trustLength = " + trustLength);

            addDnaQToStat(dnaq, it, false);
```

```
            addDnaQToStat(dnaq.reverseComplement(), it, true);
        }

        public void run() {
            int it = local.it0;
            for (DnaQ dnaq : local.dnaqs) {
                process(dnaq, it++);
            }
        }
```