Министерство образования и науки Российской Федерации

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

УТВЕРЖДЕНО:
Исполнитель: **федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Санкт- Петербургский национальный исследовательский университет информационных технологий, механики и оптики"**
Руководитель организации **_____/В. Н. Васильев/** **М.П.**

НАУЧНО-ТЕХНИЧЕСКИЙ ОТЧЕТ

о выполнении третьего этапа Государственного контракта
№ 16.740.11.0495 от 16 мая 2011 г. и Дополнению от 25 октября 2011 г. № 1

Исполнитель: федеральное государственное бюджетное образовательное 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: **_____/Шалыто Анатолий Абрамович** (подпись)

Санкт-Петербург
2012 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.
подпись, дата

Исполнители темы:

кандидат технических наук, без ученого звания _____ Корнеев Г. А.
подпись, дата

кандидат технических наук, без ученого звания _____ Станкевич А. С.
подпись, дата

без ученой степени, без ученого звания _____ Царев Ф. Н.
подпись, дата

без ученой степени, без ученого звания _____ Федотов П. В.
подпись, дата

без ученой степени, без ученого звания _____ Буздалов М. В.
подпись, дата

без ученой степени, без ученого звания _____ Александров А. В.
подпись, дата

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

без ученой степени, без ученого звания _____ Мельников С. В.
подпись, дата

без ученой степени, без ученого звания _____ Сергушичев А. А.
подпись, дата

РЕФЕРАТ

Отчет 89 с., 7 рис., 0 табл., 2 прил., 1 часть, 14 источников.

Геном, сборка генома, контиг, секвенирование, граф де Брюина, граф перекрытий

В отчете представлены результаты исследований, выполненных по 3 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 третьего этапа являются:

1. Разработка алгоритма восстановления фрагментов геномной последовательности по парным чтениям.

2. Программная реализация алгоритма восстановления фрагментов геномной последовательности по парным чтениям.

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 третьего этапа были получены следующие результаты:

1. Разработан алгоритм восстановления фрагментов геномной последовательности по парным чтениям;

2. Осуществлена программная реализация алгоритма восстановления фрагментов геномной последовательности по парным чтениям.

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 – определения последовательности нуклеотидов в образце ДНК.

В работе представлен алгоритм восстановления фрагментов геномной последовательности по парным чтениям. Алгоритм состоит из нескольких этапов и включает в себя процедуру исправления ошибок в чтениях, построение достаточно длинных фрагментов геномной последовательности с помощью графа де Брюина и, наконец, построение с помощью графа перекрытий максимально длинных непрерывных фрагментов геномной последовательности, которые далее не могут быть расширены однозначным образом.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ

В настоящем отчете о НИР применяются следующие термины с соответствующими определениями.

De novo сборка генома – определение геномной последовательности живого существа, геном которого неизвестен.

K-мер – строка длиной *K* символов над алфавитом {A, G, C, T}.

Бинарный поиск – алгоритм, позволяющий осуществлять поиск элемента в отсортированном массиве за время, пропорциональное двоичному логарифму длины массива.

Взвешенный граф – граф, в котором каждое ребро имеет вес – вещественное число, сопоставленное с ним.

Геном – совокупность генов организма. Представляет собой одну или несколько последовательностей нуклеотидов.

Граф – объект, состоящий из двух множеств. Первое множество – множество вершин графа. Второе множество – множество ребер – является подмножеством множества всех пар вершин. В ориентированных графах ребро соответствует неупорядоченной паре вершин, тогда как в неориентированных – упорядоченной.

Граф де Брюина – граф, вершинами которого являются k-меры, при этом из одной вершины есть ребро в другую, если существует такой (k+1)-мер, что k-мер, соответствующий первой вершине, является его префиксом, а k-мер, соответствующий второй, – суффиксом.

Граф перекрытий – граф, вершинами которого являются последовательности, причем ребро из одной вершины в другую существует в том случае, если суффикс последовательности, соответствующей первой вершине, совпадает с префиксом последовательности, соответствующей второй.

Квазиконтиг – протяженный непрерывный фрагмент геномной последовательности, префикс которого совпадает с одним парным чтением, а суффикс – со вторым.

Контиг – протяженный непрерывный фрагмент геномной последовательности, который не может быть расширен однозначным образом ни в одну из сторон.

Метрика N 50 – максимальная длина такого контига, что суммарная длина контигов с длиной не меньше этой составляет не меньше 50% длины генома.

Нуклеотид – химическое соединение, являющееся частью ДНК.

Обход в ширину – алгоритм, осуществляющий обход всех вершин, достижимых из заданной, в порядке увеличения расстояния.

Префикс строки – подстрока строки, начинающаяся с первого символа этой строки.

Секвенирование – определение последовательности нуклеотидов в образце ДНК.

Скэффолд – набор контигов, для которого с большой степенью уверенности определено его взаимное расположение в геномной последовательности.

Суффикс строки – подстрока строки, кончающаяся последним символом этой строки.

Хеш-таблица – структура данных, позволяющая хранить пары вида (ключ, значение). Для каждого ключа может храниться не более одного значения.

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».

Задачами этапа являются:

1. Разработка алгоритма восстановления фрагментов геномной последовательности по парным чтениям.
2. Программная реализация алгоритма восстановления фрагментов геномной последовательности по парным чтениям.

В настоящее время исследования в области геномики ведутся в таких университетах и лабораториях мира, как, например, *Cold Spring Harbor*

Laboratory (штат Нью-Йорк, США), *Университет Мериленда* (США), *Национальный центр геномного анализа* (Барселона, Испания). При изучении генома живого существа обычно выделяют три основных этапа:

а) секвенирование молекул ДНК, содержащих информацию о геноме (выполняется с использованием специальных устройств-секвенаторов);

б) сборка геномной последовательности (коротко – сборка генома, выполняется с использованием компьютеров);

в) анализ и сравнение геномов (выполняется с использованием компьютеров).

Изучение генома человека и других живых существ имеет важное прикладное значение. На основании результатов сборки генома конкретного человека возможна реализация персонифицированной медицины – определения предрасположенности человека 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, в определенном смысле, центральной среди всех задач биоинформатики. Это объясняется тем, что без ее решения нельзя приступить к детальному изучению генома живого существа и его анализу с применением других алгоритмов биоинформатики.

В середине первого десятилетия XXI века широкое распространение получили так называемые технологии *next generation sequencing* (технологии секвенирования нового поколения). По оценкам экспертов (Зубов В. В. Приборы для чтения ДНК // Химия и жизнь. 2010, № 7, с. 4 – 7. www.dubna-oez.ru/images/data/gallery/10_2948_.pps) эти технологии в настоящее время развиваются существенно быстрее, чем компьютерные технологии и алгоритмы сборки геномных последовательностей – производительность компьютеров удваивается каждые два года, а производительность геномных секвенаторов за тот же самый период увеличивается в 10 раз.

Использование существующих в настоящее время алгоритмов на персональных компьютерах приведет к тому, что сборка одного генома займет месяцы. Таким образом, актуальной является задача разработки новых алгоритмов сборки геномных последовательностей, соответствующих по своим параметрам существующим методам секвенирования. В рамках настоящей НИР будет рассматриваться так называемая задача *de novo* сборки генома – сборки генома живого существа, для которого геном еще не известен.

Сложность задачи сборки геномной последовательности обусловлена следующими факторами:

а) большой объем входных данных;

б) сложность структуры генома – наличие в нем повторов и полиморфизмов;

в) наличие ошибок в исходных данных, полученных с устройств-секвенаторов.

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

а) исправление ошибок в исходных данных – чтениях геномной последовательности;

б) сборка контигов – достаточно длинных непрерывных фрагментов искомой геномной последовательности;

в) сборка скэффолдов – наборов контигов, для которых с большой степенью уверенности определено их взаимное расположение в геномной последовательности.

В рамках третьего этапа НИР выполняется разработка и программная реализация алгоритма, позволяющего осуществить выполнение второго из указанных этапов – сборку контигов. Разрабатываемый алгоритм восстановления фрагментов геномной последовательности по парным чтениям основан на использовании графа де Брюина и графа перекрытий. Помимо этого, разрабатываемый алгоритм использует алгоритм исправления ошибок в исходных данных, который был ранее представлен в рамках данной НИР.

1 . РАЗРАБОТКА АЛГОРИТМА ВОССТАНОВЛЕНИЯ ФРАГМЕНТОВ ГЕНОМНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ ПО ПАРНЫМ ЧТЕНИЯМ

В настоящем разделе описывается алгоритм восстановления фрагментов геномной последовательности по парным чтениям.

Алгоритм сборки геномных последовательностей состоит из трех этапов, для каждого из которых разработан отдельный алгоритм:

1. Исправление ошибок в исходных чтениях на основе частотного анализа.
2. Сборка с помощью графа де Брюина квазиконтигов – последовательностей, по длине больших чтений, но не являющихся контигами в смысле невозможности наращивания.
3. Сборка квазиконтигов в контиги с помощью метода Overlap – Layout – Consensus.

Каждый следующий этап получает на вход результаты работы предыдущего.

1.1 . Алгоритм исправления ошибок в чтениях

Алгоритм исправления ошибок в данных секвенирования (чтениях геномной последовательности) [1] основан на частотном анализе содержащихся в чтениях *k*-меров и не использует графа де Брюина [4 – 6]. 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 [7, 9], дает основания полагать, что на каждой позиции нуклеотид, считанный наибольшее число раз, является верным. На практике используются наборы чтений, покрывающие геном несколько десятков раз [10].

Алгоритм и программная реализация для исправления ошибок в исходных чтениях геномной последовательности были описаны в отчете за второй этап работ настоящего контракта [3].

1.2 . Алгоритм сборки квазиконтигов

В настоящем разделе приведено описание алгоритма сборки квазиконтигов, который является модификацией алгоритма из работы [11].

1.2.1 . Основная идея метода

В предлагаемом методе используется подграф графа де Брюина, в котором множество ребер состоит только из «надежных» *(k+1)*-меров – тех, которые встречаются в чтениях достаточно большое число раз, не меньшее некоторого порогового значения, для того чтобы их можно было с очень большой вероятностью считать входящими в геном. Множество вершин состоит из тех вершин графа де Брюина, которым инцидентно хотя бы одно из выбранных ребер. Если участок нуклеотидной последовательности покрылся достаточно хорошо – все входящие в него *(k+1)*-меры по много раз входят в исходные данные, то в этом подграфе существует путь между первым и последним *k*-мерами участка.

Предлагаемый метод основан на поиске такого пути для фрагмента, соответствующего парным чтениям. Из всех путей интересны только те, которые укладываются в априорные границы длин фрагментов, поэтому слишком короткие и слишком длинные пути можно отбросить. Из оставшихся путей следует выбрать те, из которых действительно могли получиться имеющиеся парные чтения. Такие пути – хорошие кандидаты на роль пути, соответствующего фрагменту в действительности. Если нашелся единственный такой путь, то можно с очень большой уверенностью сказать, что он соответствует реальной подстроке геномной последовательности, поэтому этот фрагмент считается восстановленным, а найденный путь выводится.

1.2.2 . Оценка «правдоподопности» пути

Пусть s — строка из нуклеотидов. Будем обозначать строку, обратно-комплементарную s как s^{rc}.

Рассмотрим парные чтения r_1 и r_2 длины l_r, s_1 и s_2 – строки их нуклеотидов ($s_{i,j}$ – j-й нуклеотид i-го чтения), q_1 и q_2 – векторы вероятностей ошибок ($q_{i,j}$ – вероятность неправильного прочтения в j-й позиции i-го чтения. Пусть a_1 и a_2 – их

первые *k*-меры, а *p* – путь длины (*l-k)*, соединяющий в рассматриваемом графе a_1 и a_2^{rc}. Этому пути естественным образом соответствует строка *s* длины *l*, полученная конкатенацией a_1 и (*l-k)* символов, последовательно стоящих на ребрах этого пути. В дальнейшем, будем это преобразование пути в строку будет обозначать *s(p)*.

Необходимо оценить «правдоподобность» утверждения, что *s* является последовательностью нуклеотидов в том фрагменте, который породил чтения r_1 и r_2. Для этого сэмулируем безошибочные чтения $\tilde{s}_1$ и $\tilde{s}_2$ – в качестве $\tilde{s}_1$ возьмем первые l_r символов *s*, а в качестве $\tilde{s}_2$ – обратно-комплементированные последние l_r символов. Посчитаем вероятность того, что при чтении ошибки были допущены именно в тех позициях, где s_1 не совпадает с $\tilde{s}_1$ и s_2 не совпадает с $\tilde{s}_2$, обозначим это событие как $R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}$. Событие, произошедшее в отдельной позиции *j* чтения *i*, обозначим $R'_{i,j}$. Если исходить из того, что ошибки в каждой позиции возникают независимо (что в действительности не совсем так, но для оценки этого достаточно), то вероятность $P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2})$ будем определять по формуле:

$$P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}) = \prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} P(R'_{i,j}), \tag{1.1}$$

где

$$P(R'_{i,j}) = \begin{cases} q_{i,j}, & \text{если } \tilde{s}_{i,j} \neq s_{i,j}, \\ 1 - q_{i,j}, & \text{иначе}. \end{cases} \tag{1.2}$$

Теперь рассмотрим математическое ожидание этой вероятности и ее дисперсию, для того чтобы оценить, насколько произошедшее событие обычно. Для начала заметим, что из (1.2) следуют равенства:

$$E(P(R'_{i,j})) = q_{i,j}^2 + (1 - q_{i,j})^2 \tag{1.3}$$

и

$$E(P(R'_{i,j})^2) = q_{i,j}^3 + (1 - q_{i,j})^3. \tag{1.4}$$

Из того, что в нашей модели происхождение ошибок в разных позициях независимо, и из формул (1.1) и (1.3) для математического ожидания $E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \leftarrow s_2}))$ можно получить следующую формулу:

$$E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2})) = E(\prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} P(R'_{i,j})) = \prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} E(P(R'_{i,j})) = \prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} E(q_{i,j}^2 + (1 - q_{i,j})^2). \quad (1.5)$$

Формула для вычисления дисперсии $D(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \leftarrow s_2}))$ получается из формул (1.4) и (1.5):

$$D(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}) = E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2})^2) - E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}))^2 =$$
$$= E(\prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} P(R'_{i,j})^2) - E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}))^2 = \prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} (q_{i,j}^3 + (1 - q_{i,j})^3) - \prod_{i=1}^{2} \prod_{j=1}^{|\tilde{s}_1|} E(q_{i,j}^2 + (1 - q_{i,j})^2). \quad (1.6)$$

Для оценки «правдоподобности» можно использовать отношение:

$$L_{s_1, s_2}(\tilde{s}_1, \tilde{s}_2) = \frac{\left| P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}) - E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2})) \right|}{\sqrt{D(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2}))}}. \quad (1.7)$$

Если для пути p величина $L_{s_1, s_2}(\tilde{s}_1, \tilde{s}_2)$ меньше некоторого заданного порога L_{max}:

$$L_{s_1, s_2}(\tilde{s}_1, \tilde{s}_2) < L_{\max}, \quad (1.8)$$

то будем считать его «правдоподобным». В частности, из неравенства Чебышева следует, что если реальными последовательностями нуклеотидов являются $\tilde{s}_1$ и $\tilde{s}_2$, то вероятность того, что будут прочитаны последовательности $\bar{s}_1$ и $\bar{s}_2$, для которых $L_{\bar{s}_1, \bar{s}_2}(\tilde{s}_1, \tilde{s}_2)$ не меньше чем L_{max}, не больше чем $\frac{1}{L_{\max}^2}$.

Отметим, что соотношение (1.7) определено и для строк, которые короче, чем s_1 и s_2.

1.2.3 . Поиск путей

Для начала введем некоторые обозначения. Пусть путь p_1 длины l_1 ведет из вершины v_1 в вершину v_2, а путь p_2 длины l_2 ведет из вершины v_2 в вершину v_3. Будем обозначать конкатенацию этих путей – путь длины $l_1 + l_2$, соединяющий вершины v_1 и v_3, проходящий сначала по пути p_1, а затем – по p_2, как $p_1 \cdot p_2$. Рассмотрим множества путей P_1 и P_2. С помощью $P_1 \cdot P_2$ будем обозначать все пути, которые можно получить конкатенацией путей $p1$ и p_2 из P_1 и P_2 соответственно – $P_1 \cdot P_2 = \{p_1 \cdot p_2 \,|\, p_1 \in P_1, p_2 \in P_2\}$. Заметим, что конкатенировать можно только те пути p_1 и p_2, у которых последняя вершина p_1 совпадает с первой вершиной p_2. Если P_1

состоит из одного пути $v_1 \to v_2$, а множество P_2 состоит из путей $v_2 \to v_3$ и $v_4 \to v_5$, то множество $P_1 \cdot P_2$ будет состоять из одного пути $v_1 \to v_3$.

Теперь рассмотрим задачу поиска путей, соединяющих две заданные вершины v_1 и v_2, длины которых лежат в промежутке $[l_{\min}; l_{\max}]$. Будем обозначать множество всех путей из v_1 в v_2 длины l как P^l, тогда искомое множество всех путей из v_1 в v_2 будет получаться объединением множеств P^l:

$$P = \bigcup_{l=l_{\min}}^{l_{\max}} P^l .$$

Для поиска путей будем применять двунаправленный поиск [12], в котором происходит одновременный поиск путей, ведущих из первой вершины, и путей, ведущих во вторую вершину. Это позволяет сократить время работы с $O(d^{l_{\max}})$ до $O(d^{l_{\max}/2})$, где d – средняя степень вершины графа. Обозначим множество всех путей длины l_1, ведущих из первой вершины, за $P_1^{l_1}$, а множество всех путей длины l_2, ведущих во вторую вершину за $P_2^{l_2}$. Применение этого приема к решаемой задаче обусловлено тем, что верно равенство $P^l = P_1^{l_1} \cdot P_2^{l_2}$ для всех $l_1 \in \{0, 1, \ldots, l\}$ и $l_2 = l - L_1$. Действительно, любой путь p длины l из v_1 в v_2 можно разбить на два более коротких пути p_1 и p_2 длиной l_1 и l_2 соответственно, $p = p_1 \cdot p_2$, $l = l_1 + l_2$, верно и обратное – любые два пути p_1 из $P_1^{l_1}$ и p_2 из $P_2^{l_2}$, если их можно конкатенировать, после конкатенации образуют путь из v_1 в v_2 длины $l = l_1 + l_2$.

Для реализации такого подхода удобно запустить одновременно два обхода в ширину: из первой вершины по прямым ребрам и из второй – по обратным. Тогда на каждом шаге l можно поддерживать следующий инвариант: для первой вершины будем хранить множество $P_1^{l_1}$ всех исходящих из нее путей длины l_1, а для второй – множество $P_2^{l_2}$ всех входящих путей длины l_2, причем $l_1 + l_2 = l$. Таким образом, на l-ом шаге можно получить все пути длины l из v_1 в v_2 путем конкатенацией путей из множеств $P_1^{l_1}$ и $P_2^{l_2}$:

$$P^l = P_1^{l_1} \cdot P_2^{l_2} .$$

На начальном шаге этого алгоритма l, l_1 и l_2 равны нулю, а P_1^0 и P_2^0 содержат по одному пути нулевой длины (эту пути состоят, соответственно, из вершин v_1 и v_2).

Если E – это множество всех ребер графа, то шаг в первом обходе осуществляется по формуле:

$$P_1^{l_1+1} = P_1^{l_1} \cdot E ,$$

а шаг во втором обходе по формуле:

$$P_2^{l_2+1} = E \cdot P_2^{l_2} .$$

Для перехода к следующей итерации необходимо выбрать, в каком из обходов делать шаг. Самым простым является поочередной переход, например, когда номер итерации l четный, делать шаг в первом обходе, а когда нечетный – во втором. Для более эффективного использования памяти и времени лучше производить увеличение в том обходе, в котором на данный момент $P_i^{l_i}$ в каком-то смысле меньше. Сравнение может происходить, например, по числу путей или, если использовать для хранения путей структуру, описанную в разд. 1.2.7, по числу различных концевых вершин.

1.2.4 . Обработка путей

В конце работы алгоритма требуется оставить только «правдоподобные» пути – те пути, для которых выполняется неравенство (1.8). Заметим, что «неправдоподобные» пути можно отсекать уже на ранней стадии. Для этого будем оценивать «правдоподобность» путей из $P_1^{l_1}$ и $P_2^{l_2}$, подставив в неравенство (1.8) для пути p_1 из $P_1^{l_1}$ значения $\tilde{s}_1 = s(p_1)$ и $\tilde{s}_2 = \varepsilon$, где ε – это пустая строка, и для пути p_2 из $P_2^{l_2}$ значения $\tilde{s}_1 = \varepsilon$ и $\tilde{s}_2 = s(p_2)$.

Важно отметить, что значение (1.7) для путей из $P_1^{l_1}$ и $P_2^{l_2}$ может быть пересчитано за $O(1)$ из значений для $P_1^{l_1-1}$ и $P_2^{l_2-1}$, так как это верно для входящих в него значений $P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2})$, $E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \leftarrow s_2}))$ и $D(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \leftarrow s_2}))$. Это следует из того, что выражения (1.1) и (1.5) для вычисления значения величин $P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \to s_2})$ и $E(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \leftarrow s_2}))$ являются произведениями значений функций, зависящих только от

одной позиции, а выражение (1.6) для вычисления $D(P(R_{\tilde{s}_1 \to s_1, \tilde{s}_2 \leftarrow s_2}))$ является разностью таких произведений.

Из оставшихся «правдоподобных» путей требуется выбрать один, который будет выведен как восстановленный фрагмент. Вообще говоря, если нашлось несколько таких путей (рис. 1), то это сделать сложно, но заметим, что в частном случае, когда нашлось небольшое число близких путей, можно взять любой из них. Это обусловлено тем, что алгоритм *overlap-layout-consensus* позволяет проверять и неточные совпадения. Необходимость обработки этого случая следует из того, что в геноме встречаются однонуклеотидные полиморфизмы (*single nucleotide polymorphism, SNP*), из-за которых соответствующий путь раздваивается. Поэтому, если на каком-то шаге нашлось несколько путей разной длины или несколько сильно различающихся путей одинаковой длины, то алгоритм можно прервать, не дожидаясь итерации с номером $l_{\max}$.

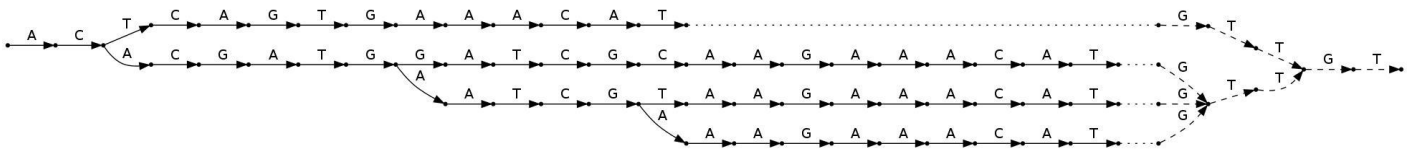

Рис. 1. Пример фрагмента графа, получившегося при поиске путей. Сплошные и штриховые пунктирные ребра – ребра, добавленные в ходе левого и правого обходов соответственно. Точечные пунктирные ребра соединяют вершины, соответствующие двум одинаковым k-мерам.

1.2.5 Общий алгоритм

Обобщим вышесказанное в один алгоритм.

В начале инициализируем l_1 и l_2 нулем, а $P_1^{l_1}$ и $P_2^{l_2}$ множествами состоящими из одного пути нулевой длины.

Затем для каждого l из промежутка $1, 2, \ldots, l_{max}$ выполним следующее:

- Если $P_1^{l_1}$ меньше чем $P_2^{l_2}$, то построим $P_1^{l_1+1} = P_1^{l_1} \cdot E$ и увеличим l_1 на единицу, иначе построим $P_2^{l_2+1} = E \cdot P_2^{l_2}$ и увеличим l_2 на единицу.
- Если требуется, пересчитаем «правдоподобность» путей из $P_1^{l_1}$ и $P_2^{l_2}$ и отбросим «неправдоподобные» пути.

- Если l лежит в промежутке $[l_{min}, l_{max}]$, то построим множество $P^l = P_1^{l_1} \cdot P_2^{l_2}$. Если оно пусто, перейдем к следующей итерации. Если до этого уже был найден путь-кандидат, то завершаем алгоритм – этот фрагмент восстановить не удалось. Если в множестве P^l есть сильно различающиеся пути, то, опять же, выходим. В множестве P^l один или несколько похожих путей – выбираем любой из них и запоминаем как путь-кандидат.

Если после всех итераций был найден только один путь-кандидат, то выводим соответствующую строку как восстановленный фрагмент.

Так как в некоторых местах графа де Брюина в связи со сложностью генома может быть достаточно высокая средняя степень вершин, то к алгоритму добавляется еще отсечение по размеру множеств $P_1^{l_1}$ и $P_2^{l_2}$ – если хотя бы в одном из них число конечных вершин больше некоторого порога, то алгоритм прерывается.

1.2.6 . Хранение графа

Для того, чтобы потребление памяти предлагаемого метода было не очень большим, необходимо иметь компактное представление используемого подграфа графа де Брюина. Для его хранения достаточно хранить только множество его ребер, что можно эффективно делать, используя, например, хеш-таблицу с открытой адресацией. Преимуществами такого подхода хранения перед другими являются его простота, быстродействие и возможность балансировки между используемой памятью и скоростью. Более эффективными с точки зрения потребляемой памяти являются *rank/select* словари [14], которые позволяют сделать ее использование близким к энтропии, но из-за этого увеличивается время доступа.

Важным является и то, что каждый *(k+1)*-мер входит в граф вместе с обратно-комплементарным. Тогда вместо пары *(k+1)*-меров s и s^{rc} можно хранить только один, определяемый по некоторому правилу – например, таким правилом может быть выбор лексикографически минимального *(k+1)*-мера. В этом случаем необходимый объем памяти уменьшается примерно в два раза для четных k и ровно

в два раза – для нечетных (только для четных k бывают обратно комплементарные себе $(k+1)$-меры).

1.2.7 . Хранение путей

Самым простым способом хранения путей является множество, состоящего из строки $s(p)$ для каждого пути p. К сожалению, часто у одного пути возникает несколько продолжений, из-за чего строки приходится копировать, что требует $O(l)$ времени, где l – это длина строки (если бы продолжений было бы не больше одного, то при соответствующей реализации строк на добавление одного символа требовалось бы амортизировано $O(1)$ времени). Этого можно избежать, если для хранения путей использовать граф.

Рассмотрим множество $P_1^{l_1}$ всех путей из v_1 длиной l_1. Построим граф, в котором существуют вершины вида (v, l) тогда и только тогда, когда в исходном графе де Брюина существует путь из v_1 в v длиной l. Направленным ребром в этом графе будет соединены вершины (u, l) и $(v, l + 1)$ тогда и только тогда, когда в исходном графе де Брюина есть ребро $u \to v$. Будем называть множество всех вершин вида (v, l) слоем с номером l. Отметим, что для любого пути из v_1 в v длины l_1 существует соответствующий путь из $(v_1, 0)$ в (v, l_1). Верно и обратное, любому пути из $(v_1, 0)$ в (v, l_1) естественным образом соответствует путь в исходном графе из v_1 в v длины l_1. Обновление этого графа до графа путей длины l_1+1 требует $O(n)$ времени, где n – число конечных вершин – вершин в l_1-ом слое. Для учета «правдоподобности» будем для каждой вершины (v, l) хранить информацию о значении $L_{s_1,s_2}(s(p), \varepsilon)$ и информацию, необходимую для пересчета этого значения для пути на единицу большей длины, только для одного пути p, который определяется по индукции из предыдущих вершин. Для вершины $(v_1, 0)$ – это путь состоящий из одной вершины v_1. Для вершины (v, l) рассмотрим все пути, заканчивающиеся в этой вершине. Из них выберем только те, префикс которых хранится в одной из вершин слоя $l - 1$. Из оставшихся путей выберем тот, значение $L_{s_1,s_2}(s(p), \varepsilon)$ для которого ближе к нулю, будем его обозначать за $L(v, l)$. При образовании нового слоя, если в какой-то вершине значение $L(v, l_1 + 1)$ будет не

меньше чем порог L_{max}, то удалим эту вершину. Заметим, что такое отбрасывание не является эквивалентным отбрасыванию только «неправдоподобных» путей, а является только приближением.

Для множества $P_2^{l_2}$ построение аналогично. В качестве вершин используются пары вида *(v, l)* тогда и только тогда, когда есть путь длины *l* из *v* в v_2. Вершины *(u, l + 1)* и *(v, l)* соединены ребром, если в графе де Брюина есть ребро $u \to v$. В качестве *L(v, l)* используется значение $L_{s_1,s_2}(\varepsilon, s(p))$ для некоторого пути *p* определяемого из индукции, аналогичной предыдущему случаю.

1.3 . Алгоритм сборки контигов

Сборка контигов из квазиконтигов основана на подходе *overlap-layout-consensus* [13] и состоит из нескольких этапов:

1. Поиск перекрытий.
2. Поиск ненайденных перекрытий с помощью найденных.
3. Удаление неверных перекрытий.
4. Поиск развилок.
5. Удаление транзитивных ребер.
6. Вывод контигов.
7. Вывод консенсуса для контигов.

Такой процесс повторяется, при этом на каждом следующем этапе контиги, построенные на предыдущем этапе, считаются входными данными (рис. 2).

Рис. 2. Высокоуровневая блок-схема алгоритма сборки контигов

Далее подробнее рассмотрим каждый этап.

1.3.1 . Поиск перекрытий

На данном этапе необходимо найти перекрытия между всеми квазиконтигами. Для начала приведем определение перекрытия. Пусть имеется квазиконтиг *A* и квазиконтиг *B*. Перекрытием называется ситуация, когда некоторый суффикс квазиконтига *A* совпадает с некоторым префиксом квазиконтига *B* (рис. 3). При этом строка, по которой перекрываются квазиконтиги, не должна быть маленькой. Минимальный порог длины такой строки записывается в файл конфигурации и является параметром алгоритма.

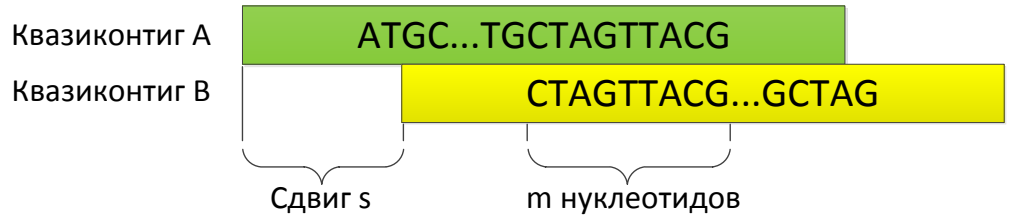

Рис. 3. Перекрытие двух квазиконтигов

При нахождении перекрытия строится ребро из квазиконтига *A* в квазиконтиг *B*. Ребро задается номерами квазиконтигов *A* и *B* и сдвигом *s* (рис. 3). Из определения перекрытия следует, что сдвиг s является неотрицательно величиной.

Отметим, что из-за неотрицательности сдвига любое перекрытие (кроме случая полного совпадения квазиконтигов) однозначно задается ребром. Если квазиконтиги совпадают, то можно добавить ребро как из A в B, так и из B в A.

Теперь рассмотрим подробнее процесс поиска перекрытий. Для начала построим строку $C_1\$C_2\$C_3\$...C_n\$$, где C_i – *i*-ый квазиконтиг. После этого необходимо построить суффиксный массив для этой строки – отсортированный лексикографически массив всех суффиксов строки. С его помощью можно найти все квазиконтиги, в которых встречается заданная подстрока. Это можно сделать, например, с помощью бинарного поиска в суффиксном массиве суффиксов, которые начинаются с заданной подстроки. Они будут располагаться рядом за счет сортировки.

Зафиксируем некоторый квазиконтиг B. Необходимо найти все ребра, ведущие в него. Будем рассматривать его префиксы в порядке увеличения длины, начиная с минимального порога. Зафиксируем некоторый префикс. Для того, чтобы некоторый квазиконтиг A перекрывался с квазиконтигом B, необходимо, чтобы зафиксированный префикс строки B был суффиксом строки A. Следовательно, необходимо найти все квазиконтиги, которые заканчиваются зафиксированным префиксом.

Для поиска необходимо воспользоваться построенным суффиксным массивом. Однако для того, чтобы искать не все строки, которые содержат выбранный префикс, а только те, которые оканчиваются на него, необходимо дописать в конец выбранного префикса символ $. Благодаря этому и будет выполняться то, что требуется.

При таком подходе будут добавляться ребра только для точных перекрытий – для случаев, когда суффикс строки A полностью совпадает с префиксом строки B. Однако в квазиконтигах довольно много ошибок и имеет смысл добавлять ребра и для неточных перекрытий. Для этого необходимо после фиксирования префикса, по которому будут перекрываться квазиконтиги, изменить некоторое небольшое число его символов и добавить ребра из всех квазиконтигов, которые оканчиваются уже на измененный префикс квазиконтига B.

Понятно, что имеет смысл изменять только небольшое число символов префикса. Поэтому одним из параметров алгоритма данного этапа является максимальное число ошибок в перекрывающейся строке.

Приведем оценки сложности предложенного алгоритма. Он состоит из двух частей.

Первая часть – построение суффиксного массива. Разбиение неотсортированного массива на части происходит за $O(nk)$, где n – число квазиконтигов, k – длина префикса, по которому разбиваем (является константой). Сортировка некоторой части осуществляется за $O(n\log nL)$, где L – средняя длина квазиконтига.

Вторая часть с исправлением двух ошибок выполняется в худшем случае за $O(nC_к^2 \log n)$ для небольших L. Для больших же L эта оценка не верна из-за того, что в суффиксом массиве остается мало префиксов и для каждого из них можно просто посчитать число ошибок. Также применяется еще несколько оптимизаций, что делает процесс поиска перекрытий более быстрым.

1.3.2 . Поиск ненайденных перекрытий с помощью найденных

После выполнения предыдущего этапа возможны ситуации, когда некоторые перекрытия не будут найдены. Это может случиться, например, если число ошибок в перекрывающейся строке немного больше максимально разрешенного числа ошибок. Поэтому необходимо запустить поиск ненайденных перекрытий с помощью найденных.

Для этого берется некоторый квазиконтиг, выписываются все перекрывающиеся с ним квазиконтиги. После этого среди них производится поиск новых перекрытий. Ввиду того, что квазиконтигов, между которыми производится поиск перекрытий, теперь намного меньше, чем в предыдущем этапе, используется другой алгоритм поиска.

Рассмотрим этот алгоритм. Предположим, что некоторый квазиконтиг A выбран. Выпишем все квазиконтиги, из которых существует ребро в него. Тогда возможно несколько ситуаций, некоторые из которых показаны на рис. 4, 5.

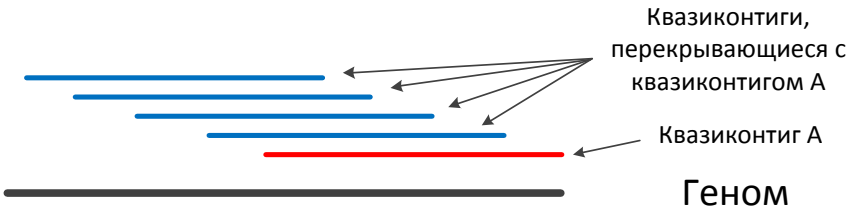

Рис. 4. Схема перекрытия при геноме без вилок

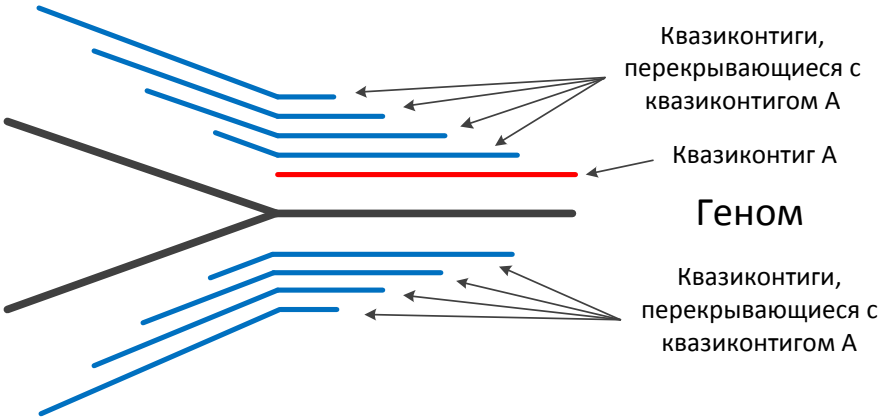

Рис. 5. Схема перекрытия при геноме с вилками

Алгоритм делает следующее. После выбора всех квазиконтигов, из которых существует ребро в выбранный квазиконтиг А, и производится их сортировка по сдвигу относительно квазиконтига А. После этого выполняется последовательная обработка квазиконтигов, начиная с самого левого и кончая самым правым. По мере обработки сохраняются последние квазиконтиги из разных веток – квазиконтиги, которые не перекрываются между собой. Обработка состоит в том, что обрабатываемый квазиконтиг проверяется на возможность перекрытия с последними квазиконтигами из разных веток. Если такое перекрытие найдено, то считается, что обрабатываемый квазиконтиг должен находиться в той же ветке, что и квазиконтиг, с которым он перекрывается. Добавляется новое ребро между этими квазиконтигами, если оно еще не существовало. Если же не получается найти перекрытие с существующими квазиконтигами разных веток, то добавляется новая ветка, а обрабатываемый квазиконтиг сохраняется как последний квазиконтиг новой ветки.

В этом алгоритме не производится поиск перекрытий среди квазиконтигов, которые перекрываются с А и которые расположены правее квазиконтига А (т.е. квазиконтигов, в которые есть ребра из А) из-за того, что в большинстве случаев существует другой квазиконтиг, благодаря которому ненайденные перекрытия между ними будут найдены. Поиск же перекрытий между квазиконтигами, которые левее А и квазиконтигами, которые правее А, не производится из-за того, что в транзитивных перекрытиях нет необходимости.

Приведем оценки сложности алгоритма данного этапа. Алгоритм выполняется за $O(n(h\log h+hL)+t)$, где n – число квазиконтигов, h – среднее число перекрытий с квазиконтигом (для реальных геномов обычно невелико и можно считать константой), L – средняя длина квазиконтига, t – число всех найденных перекрытий.

1.3.3 . Удаление неверных перекрытий

Из-за того, что большинство строк небольшого размера не однозначно задают место в геноме, возможны ситуации, когда появляются ребра между квазиконтигами, которые на самом деле не должны существовать. Такие ребра можно опознать благодаря тому, что число других ребер, которые бы подтверждали реальное существование такого перекрытия, обычно невелико. Для хороших же ребер такое число является большим из-за большого покрытия генома квазиконтигами (маленькие части покрываются квазиконтигами несколько десятков раз). На следующих этапах такие ребра будут сильно мешать, поэтому выполняется их поиск и удаление.

Для поиска таких ребер выполняется следующее. Сначала фиксируется некоторый квазиконтиг A. После этого выписываются все квазиконтиги, которые перекрываются с ним. После этого для всей охваченной части генома строится консенсус – для каждой позиции выбирается нуклеотид, который встречается на этой позиции среди выписанных квазиконтигов наибольшее число раз. При этом возможно, что консенсуса для каких-то позиций просто нет из-за того, что реально существует несколько веток (рис. 5).

Далее рассматривается каждый квазиконтиг, который перекрывается с A, и сравнивается с консенсусом. Если число несовпадающих с консенсусом нуклеотидов велико, то считается, что рассматриваемый квазиконтиг не должен перекрываться с квазиконтигом A, и ребро между ними нужно удалить. Позиций, для которых нет консенсуса, исключаются из рассмотрения при сравнении нуклеотидов.

Приведем оценки сложности алгоритма. Поиск неверных перекрытий происходит за $O(nhL)$, где n – число квазиконтигов, h – число перекрывающихся

квазиконтигов (константа), L – средняя длина квазиконтига. Непосредственно удаление ребер выполняется за $O(t)$, где t – число найденных плохих ребер.

1.3.4 . Поиск развилок

Как можно понять из рис. 5, геном состоит не только из простых последовательностей нуклеотидов, но там также возможны развилки и циклические повторы некоторых его частей.

На рис. 5 показана входящая развилка – развилка, при которой несколько веток сходятся в одну. Также существуют исходящие развилки – такие развилки, при которых одна ветка расходится на две или более веток.

Для корректного вывода контигов необходимо знать положение развилок. Алгоритм поиска развилок отличается от алгоритма поиска неверных перекрытий только в последней стадии – после построения консенсуса нам не надо его сравнивать с квазиконтигами, достаточно посмотреть на него и найти места, где нет консенсуса.

Заметим, во-первых, что на позициях квазиконтига A консенсус всегда должен быть (его может не быть только на небольшом множестве позиций из всех позиций квазиконтига A). Во-вторых, входящая развилка возможна только в области до квазиконтига A, а исходящая – в области после. Поэтому для поиска развилок достаточно найти самую правую позицию, до которой нет консенсуса, и самую левую, после которой тоже нет консенсуса. Эти позиции и будут позициями ближайших к квазиконтигу A развилок.

Приведем оценку сложности алгоритма. Выполняется $O(nhL)$ действий, где n – число квазиконтигов, h – число перекрывающихся квазиконтигов (константа), L – средняя длина квазиконтига.

1.3.5 . Удаление транзитивных ребер

Как уже отмечалось выше, в транзитивных ребрах нет никакой необходимости, и, более того, они увеличивают сложность алгоритмов на следующих шагах. Для избежания этого необходимо удалить все такие ребра.

Дадим определение транзитивному ребру между квазиконтигами. Ребро AC между квазиконтигами A и C считается транзитивным, если существует квазиконтиг B и существуют ребра AB и BC, каждое из которых имеет меньший сдвиг (рис. 6). Тогда ребро AC необходимо удалить.

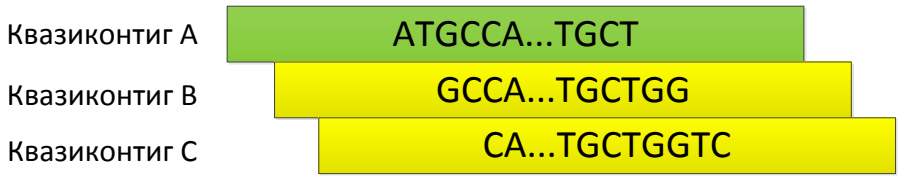

Рис. 6. Перекрывающиеся квазиконтиги

Поиск осуществляется следующим образом. После фиксирования некоторого квазиконтига A загружаются исходящие из него ребра, и они сортируются по сдвигу относительно квазиконтига A. После этого находятся квазиконтиги, которые перекрываются с A, и загружаются их ребра. Далее перебираются два соседних квазиконтига B и C в одной ветке, между которыми сауществует ребро, и удаляется более длинное из ребер AB и AC.

Приведем оценку сложности алгоритма. Поиск выполняется за $O(n(h\log h+h))$, где n – число квазиконтигов, h – среднее число перекрытий с квазиконтигом (константа), L – средняя длина квазиконтига. Непосредственно удаление ребер – за $O(t)$, где t – число найденных транзитивных ребер.

1.3.6 . Вывод контигов

На этом этапе выводятся контиги – непересекающиеся пути с вершинами, имеющими равные единице входящую и исходящую степени. Любой контиг выводится в виде последовательности номеров квазиконтигов и смещений между двумя соседними квазиконтигами.

Этот этап выполняется в два подэтапа. На первом подэтапе строятся отображения m_i, на втором происходит полное восстановление путей контигов и их вывод.

Рассмотрим первый подэтап. На нем последовательно строятся отображения m_1, m_2, m_4, … , заканчивая первым пустым отображением. Отображение m_i ставит в

соответствие вершине некоторую другую вершину, которая отстоит от первой ровно на i ребер. Если для некоторой вершины нет другой вершины, которая отстоит ровно на i ребер, то первая вершина не участвует в этом и следующих отображениях.

Отметим, что вершины, имеющие входящую или исходящую степень больше единицы, участвуют только в качестве начальных или конечных вершин пути. Поэтому для вершин, имеющих исходящую степень большее единицы, нет следующей вершины, отстоящей на одно ребро.

Также отметим, что строятся только m_{i}, где i является степенью двойки. Благодаря этому достигается экономия используемой памяти, а также не теряется возможность восстановления любого пути.

Первое отображение m_1 строится следующим образом. Рассматривается множество квазиконтигов и положения исходящих вилок для каждого используемого квазиконтига, а также ребра, идущие из квазиконтигов из этого множества, после этого выбираются квазиконтиги, в которые идут ребра. Затем для каждого квазиконтига можно узнать исходящую степень, и, при необходимости, найти для каждого квазиконтига следующий квазиконтиг. Таким образом получаем часть отображения и объединяем эти части, получая все отображение m_1, которое записывается в файл.

Каждое следующее отображение строится с использованием только предыдущего. Для этого рассматриваются все квазиконтиги A, которым сопоставлены некоторые квазиконтиги B в предыдущем отображении. Перебираются все квазиконтиги из предыдущего файла отображения и ищется отображение для квазиконтигов B, которые сами являются отображением для квазиконтигов A из переданного на вход множества квазиконтигов (рис. 7). Если отображение найдено, то добавляется новая запись в новое отображение. За счет этого происходит удвоение числа ребер между квазиконтигами в новом отображении.

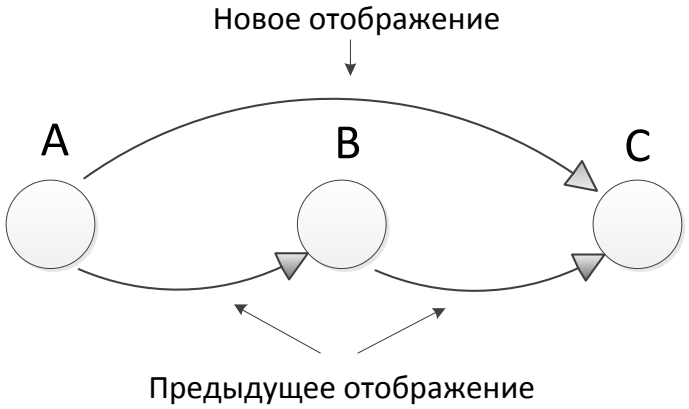

Рис. 7. Построение нового отображения

Рассмотрим второй подэтап. Вначале выбираются все квазиконтиги и положения исходящих вилок. По этим данным можно определить какие из квазиконтигов являются начальными вершинами путей. После этого необходимо пройтись в обратном порядке по отображениям m_i и восстановить пути из найденных начальных вершин.

Восстановление путей происходит следующим образом. Предположим, что зафиксирована некоторая начальная вершина. Сначала используется отображение с максимальным числом ребер между вершинами. Если выбранная начальная вершина встретилась в качестве первой вершины отображения, то можно запомнить вторую вершину, которая тоже будет на пути, который необходимо восстановить. После этого находятся все используемые в пути вершины в качестве первых вершин. При нахождении отображения добавляется еще одна вершина в путь и т. д. Таким образом, после использования отображения m_1 весь путь будет восстановлен.

Приведем оценки сложности алгоритмов данного этапа.

Создание отображение m_1 в первом этапе выполняется за $O(nh)$, где n – число квазиконтигов, h – число перекрывающихся квазиконтигов (константа). Каждое последующее создание нового отображения требует $O(n)$ действий. Всего непустых отображений будет $\lfloor \log L_{\max} \rfloor$, где $L_{\max}$ – длина наибольшего контига.

Суммарная оценка всех выполненных действий при восстановлении всех путей – $O(n(\log L_{\max} + L_i))$ действий, где L_i – длина контига i, n – число контигов.

Можно считать, что $L_i > \log L_{max}$, поэтому оценку для поиска всех путей можно упростить – $O(L_{sum})$, где L_{sum} – суммарная длина всех восстановленных контигов.

1.3.7 . Вывод консенсуса для контигов

На этом этапе рассматриваются контиги, которые записаны в виде последовательности номеров квазиконтигов, и выводится одна последовательность нуклеотидов – консенсус всех перечисленных квазиконтигов.

Суммарная оценка сложности данного алгоритма – $O(L_{sum})$, где L_{sum} – суммарная длина всех восстановленных контигов.

1.4 . Теоретическая оценка сложности алгоритмов сборки геномных последовательностей

В настоящем разделе описана теоретическая оценка ввычислительной сложности описанных выше алгоритмов сборки геномных последовательностей.

1.4.1 . Теоретическая оценка сложности алгоритма исправления ошибок в чтениях геномной последовательности

Обозначим длину геномной последовательности, из которой произодились чтения, как L, а коэффициент покрытия геномной последовательности чтениями – как C. Размер k-мера обозначим как k, а общее число различных k-меров, которые присутствуют в чтениях геномной последовательности – как N. Это число зависит от того, сколько ошибок было допущено при чтении – его типичное значение для покрытия в десятки и сотни раз составляет от $2L$ до $3L$.

Время работы первого этапа алгоритма (вычисления частот k-меров) составляет $O(LCk)$. Информация о частотах вхождения k-меров, получающаяся в результате выполнения первого этапа алгоритма, имеет размер $O(LCk)$. Эта информация хранится в хэш-таблице [2].

Время работы второго этапа алгоритма (поиск исправлений, позволяющих из «подозрительных» k-меров получить «надежные») составляет $O(Lk + M\log M)$ при условии, что размер хэш-таблицы выбран достаточно большим, для чтобы среднее время операции с ней составляло $O(1)$. Здесь как M обозначено число найденных

исправлений, а слагаемое $M\log M$ отвечает за сортировку исправлений по их позициям в исходных данных. Значение M в худшем случае составляет порядка LC (если ошибка чтения допущена в каждом нуклеотиде), а типичное значение – порядка L.

Время работы третьего этапа алгоритма (внесение исправлений в чтения) составляет $O(LC + M)$ при условии, что исправления упорядочены по позиции, в которой их необходимо произвести.

Таким образом, общее время работы алгоритма составляет $O(LCk + L\log L)$, а объем требуемой памяти – $O(LCk)$.

1.4.2 . Теоретическая оценка сложности алгоритма сборки квазиконтигов

Для оценки сложности алгоритма сборки квазиконтигов обозначим среднюю степень вершины графа де Брюина как d. Тогда время работы алгоритма можно оценить как $O(d^{l_{\max}/2})$.

1.4.3 . Теоретическая оценка сложности алгоритма сборки контигов

Для оценки сложности алгоритма сборки контигов из квазиконтигов обозначим:

- число квазиконтигов за N_{qc};
- среднюю длину квазиконтига за L_{qc} (логично считать, что $N_{qc}L_{qc} \approx L$, но в реальности из-за того, что геном покрыт чтениями несколько десятков раз, $N_{qc}L_{qc}$ больше L, но порядок остается тем же. При первой сборке контигов $L_{qc} \approx 500$).

Суммарная оценка сложности алгоритма поиска перекрытий составляет $O(N_{qc} log N_{qc} L_{qc})$. Оценка алгоритмов этапов поиска ненайденных перекрытий, удаления неверных перекрытий, поиска развилок и удаления транзитивных ребер – $O(N_{qc}L_{qc})$. Оценка же алгоритмов вывода контигов и вывода консенсуса для контигов – $O(N_{qc}L_{qc})$.

Таким образом, весь алгоритм сборки контигов из квазиконтигов осуществляется за $O(N_{qc} log N_{qc} L_{qc})$.

Выводы по главе 1

1. Описан алгоритм сборки геномных последовательностей, состоящий из трех алгоритмов.
2. Описан алгоритм сборки квазиконтигов из исправленных чтений геномной последовательности.
3. Описан алгоритм сборки контигов из квазиконтигов.
4. Проведена теоретическая оценка вычислительной сложности указанных алгоритмов.

2 . ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ВОССТАНОВЛЕНИЯ ФРАГМЕНТОВ ГЕНОМНОЙ ПОСЛЕДОВАТЕЛЬНОСТИ ПО ПАРНЫМ ЧТЕНИЯМ

В настоящем разделе описывается программная реализация алгоритма восстановления фрагментов по парным чтениям. В рамках данной работы было создано программное средство, написанное на языке программирования *Java*, включающее в себя множество инструментов для сборки генома. В частности, программное средство включает в себя реализацию алгоритмов исправления ошибок в чтениях геномной последовательности, алгоритм сборки квазиконтигов и алгоритм сборки контигов. Полностью исходные коды разработанного программного средства весьма объемны и занимают более 500 КБ, поэтому в данном отчете будут приведены лишь основные фрагменты реализации. Программное средство поддерживает многопоточность и при работе на многоядерном компьютере имеет существенное ускорение.

2.1 . Хранение данных о геномной последовательности

Для представления в памяти нуклеотидной последовательности используются классы `PairedDnaQ`, `SingleDnaQ` и `DnaWritable`. Опишем подробнее структуру класса `DnaWritable`.

Поля класса

- `array: byte[]` – внутренний массив для хранения данных о последовательности.

Конструкторы

- `DnaWritable(IDna dna)` – делает копию переданной последовательности ДНК.
- `DnaWritable()` – создает пустую последовательность.
- `DnaWritable(int length)` – создает последовательность длины `length`.

Методы

- `set(IDna dna):void` – выставляет значение последовательности, равное предложенному.
- `write(DataOutput dataOutput):void` – записывает последовательность в поток данных.
- `readFields(DataInput dataInput):void` – читает последовательности из потока данных.
- `nucAt(int pos):byte` – возвращает значение нуклеотида в запрашиваемой позиции..
- `reverse():IDna` – обращает последовательность.
- `complement():IDna` – комплементирует последовательность.
- `substring(int beginIndex, int endIndex):IDna` – возвращает подпоследовательность.
- `setNuc(int pos, byte value):IDna` – выставляет значение нуклеотида в данной позиции равное предложенному.

2.2 . Программная реализация алгоритма исправления ошибок в чтениях

Входными данными алгоритма исправления ошибок и всего программного средства восстановления фрагментов геномной последовательности по парным чтениям являются чтения геномной последовательности в формате *FASTQ* [7].

Программная реализация алгоритма для исправления ошибок в исходных чтениях геномной последовательности была описана в отчете за второй этап работ настоящего контракта [3]. Авторами настоящей работы получено свидетельство о регистрации программы для ЭВМ на «Программное средство для удаления ошибок из набора чтений нуклеотидной последовательности».

2.3 . Программная реализация алгоритма сборки квазиконтигов

Сборка квазиконтигов основана на использовании подграфа графа де Брюина. Граф представляется множеством своих ребер. Объявления и описания членов

класса для хранения графа де Брюина находятся в файле: `/ru/ifmo/genetics/tools/CompactDeBruijnGraph.java`.

Поля класса

- `edges:LongsHashSet`
- `k: int`

Конструкторы

- `CompactDeBruijnGraph(int k, long memSize)` – создает граф для данного `k` с использование `memSize` байт памяти.

Открытые методы

- `addEdges(IDna dna): void` – добавляет все ребра, содержащиеся в данной последовательности;
- `containsEdges(IDna dna):` boolean – проверяет, что все ребра, содержащиеся в данной последовательности имеются в графе;
- `addEdge(long e):` boolean – добаляет ребро в граф;
- `outcomeEdges(long v): long[]` – возвращает список исходящих ребер из данной вершины;
- `incomeEdges(long v): long[]` – возвращает список входящих ребер в данную вершину.

Ниже в листинге 1 приведена реализация класса `CompactDeBruijnGraph`, отвечающего за компактное хранение графа де Брюина.

Листинг 1. Реализация класса для хранения графа де Брюина

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


import ru.ifmo.genetics.dna.LightDna;
import ru.ifmo.genetics.tools.set.BigLongsHashSet;
import ru.ifmo.genetics.tools.set.LongsHashSet;


import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.Arrays;
```

```
public class CompactDeBruijnGraph implements Writable {
    private static int MAX_K = 30;
    private LongsHashSet edges;

    public int k;
    public int k2; // k * 2
    public long incomeEdgeIncrement;
    public long vertexMask;

    private int unusedBits;

    public CompactDeBruijnGraph() {
        edges = new BigLongsHashSet();
    }

    public CompactDeBruijnGraph(int k, long memSize) {
        if ((k > MAX_K) || (k <= 0)) {
            throw new IllegalArgumentException("k should be
        in range 1.." + MAX_K);
        }
        edges = new BigLongsHashSet(memSize);
        this.k = k;
        k2 = k * 2;
        incomeEdgeIncrement = 1L << k2;
        vertexMask = incomeEdgeIncrement - 1;
        unusedBits = 64 - k2 - 2;
    }

    public void reset() {
        edges.reset();
    }

    public void addEdges(LightDna dna) {
        if (dna.length() <= k)
```

```
            return;
        long cur = 0;
        for (int i = 0; i < k; i++) {
            cur = (cur << 2) | dna.nucAt(i);
        }
        // String s = dnaq.toString();
        for (int i = k; i < dna.length(); i++) {
            cur = cur & vertexMask;
            cur = (cur << 2) | dna.nucAt(i);
            addEdge(cur);
        }
    }

    public boolean containsEdges(LightDna dna) {
        if (dna.length() <= k)
            return true;
        long cur = 0;
        for (int i = 0; i < k; i++) {
            cur = (cur << 2) | dna.nucAt(i);
        }
        // String s = dnaq.toString();
        for (int i = k; i < dna.length(); i++) {
            cur = cur & vertexMask;
            cur = (cur << 2) | dna.nucAt(i);
            if (!containsEdge(cur)) {
                return false;
            }
        }
        return true;

    }

    private long reverseComplementEdge(long e) {
        e = ((e & 0x3333333333333333L) << 2) | ((e &
        0xccccccccccccccccL) >>> 2);
```

```
        e = ((e & 0x0f0f0f0f0f0f0f0fL) << 4) | ((e &
        0xf0f0f0f0f0f0f0f0L) >>> 4);
        e = ((e & 0x00ff00ff00ff00ffL) << 8) | ((e &
        0xff00ff00ff00ff00L) >>> 8);
        e = ((e & 0x0000ffff0000ffffL) << 16) | ((e &
        0xffff0000ffff0000L) >>> 16);
        e = ((e & 0x00000000ffffffffL) << 32) | ((e &
        0xffffffff00000000L) >>> 32);

        e = ~e;

        return e >>> unusedBits;
    }

    private long getEdgesKey(long e, long rcE) {
        return Math.min(e, rcE);
    }

    private long getEdgeKey(long e) {
        return getEdgesKey(e, reverseComplementEdge(e));
    }

    public boolean addEdge(long e) {
        return edges.put(getEdgeKey(e));
    }

    public boolean containsEdge(long e) {
        return edges.contains(getEdgeKey(e));
    }

    public long[] outcomeEdges(long v) {
        long e = v << 2;
        long rcE = reverseComplementEdge(e);
        long[] t = new long[4];
        int tl = 0;
```

```
        for (int i = 0; i < 4; i++, e++, rcE -=
        incomeEdgeIncrement) {
            assert reverseComplementEdge(e) == rcE;
            if (edges.contains(getEdgesKey(e, rcE))) {
                t[tl++] = e;
            }
        }

        long[] res = new long[tl];
        System.arraycopy(t, 0, res, 0, tl);
        return res;
    }

    public long[] incomeEdges(long v) {
        long e = v;
        long rcE = reverseComplementEdge(e);
        long[] t = new long[4];
        int tl = 0;
        for (int i = 0; i < 4; i++, e += incomeEdgeIncrement,
        rcE--) {
            assert reverseComplementEdge(e) == rcE;
            if (edges.contains(getEdgesKey(e, rcE))) {
                t[tl++] = e;
            }
        }

        long[] res = new long[tl];
        System.arraycopy(t, 0, res, 0, tl);
        return res;
    }

    public long edgesSize() {
        return edges.size();
    }
```

```
    @Override
    public void write(DataOutput out) throws IOException {
        edges.write(out);
        out.writeInt(k);
        out.writeInt(k2);
        out.writeLong(incomeEdgeIncrement);
        out.writeLong(vertexMask);
        out.writeInt(unusedBits);
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        edges.readFields(in);
        k = in.readInt();
        k2 = in.readInt();
        incomeEdgeIncrement = in.readLong();
        vertexMask = in.readLong();
        unusedBits = in.readInt();
    }
}
```

Классы, отвечающие непосредственно за сборку квазиконтигов находятся в пакете `ru.ifmo.genetics.tools.assembling`:

- `BuildAndDumpGraph` – класс, отвечающий за построение графа де Брюина из множества «надежных» *k*-меров;
- `ConnectedComponentsFinder` – класс, отвечающий за поиск компонент связности в графе де Брюина;
- `ReadsFiller` – класс, отвечающий за распределение (*k*+1)-меров.

В пакете `ru.ifmo.genetics.tools.assembling.task` находится класс `FillingTask`, который отвечает за обходы и поиск путей в графе де Брюина. Данный класс реализует преобразование парных чтений в квазиконтиги, путем заполнения промежутков между ними.

Опишем подробнее методы класса `FillingTask`, так как в нем сосредоточена значительная часть логики сборки квазиконтигов:

- `buildReversePath(DnaQBuilder builder, Set<BfsTreeNode> layer): boolean` – строит путь от начального *k*-мера до слоя `layer`;
- `buildPath(DnaQBuilder builder, Set<BfsTreeNode> layer): boolean` – строит путь от слоя `layer` до конечного *k*-мера;
- `leftBfsTurn(Map<Long, BfsTreeNode> currentQueue, Map<Long, BfsTreeNode> nextQueue, byte nuc, double rightnessFactor): void` – делает шаг в левом обходе в ширину;
- `rightBfsTurn(Map<Long, BfsTreeNode> currentQueue, Map<Long, BfsTreeNode> nextQueue, byte nuc, double rightnessFactor): void` – делает шаг в правом обходе в ширину;
- `removeIncorrectNodes(Map<Long, BfsTreeNode> queue): void` – удаляет вершины, пути от/до которых сильно отличаются от имеющихся данных;
- `doubleBfs(DnaQBuilder leftDnaBuilder, long leftVertex, long rightVertex, long minPathLength, long maxPathLength, DnaQ leftHint, DnaQ rightHint): DnaQ` – запускает параллельно два обхода в ширину;
- `run(): void` – начинает выполнение задания;
- `fillRead(DnaQ left, DnaQ right): DnaQ` – получает квазиконтиг из парных чтений.

В приложении Б приведена полная программная реализация алгоритма сборки квазиконтигов.

2.4 . Программная реализация алгоритма сборки контигов

Сборка контигов из квазиконтигов основана на подходе *overlap-layout-consensus* [13] и, как было отмечено выше, состоит из нескольких основных этапов:

1. Поиск перекрытий.
2. Поиск ненайденных перекрытий с помощью найденных.
3. Удаление неверных перекрытий.
4. Поиск развилок.
5. Удаление транзитивных ребер.
6. Вывод контигов.
7. Вывод консенсуса для контигов.

Программный модуль сборки контигов из квазиконтигов находится в пакете `ru.ifmo.genetics.genetics.tools.longReadsAssembler` и может быть запущен по стадиям следующими командами с добавлением необходимых параметров:

- поиск перекрытий: `java ru.ifmo.genetics.tools.longReadsAssembler.overlaps.overlapper.Overlapper`;
- поиск ненайденных перекрытий с помощью найденных: `java ru.ifmo.genetics.tools.longReadsAssembler.overlaps.saturator.OverlapsSaturator`;
- удаление неверных перекрытий: `java ru.ifmo.genetics.tools.longReadsAssembler.overlaps.weeder.OverlapsWeeder`;
- вывод контигов и построение консенсуса для них: `java ru.ifmo.genetics.tools.longReadsAssembler.layouter.Layouter`.

Алгоритм сборки контигов принимает на вход квазиконтиги и файл конфигурации. Выходными данными является набор контигов, которые были собраны из квазиконтигов с заданными в файле конфигурации параметрами. Они выдаются в формате *FASTA* [8].

Модуль сборки контигов содержит более 50 классов. В данном отчете не приводится описание и программная реализация всех классов. Тем не менее, опишем основные части программной реализации алгоритма сборки контигов.

2.4.1 . Суффиксный массив

Класс `SuffixArray` находится в пакете `ru.ifmo.genetics.tools.longReadsAssembler.suffixArray` и отвечает за хранение суффиксного массива.

Поля класса

- `text:LargeByteArray` – строка, представленная в виде массива символов, для которой хранится суффиксный массив;
- `array:FiveByteArray` – суффиксный массив;
- `length:int` – длина суффиксного массива.

Конструкторы

- `SuffixArray(LargeByteArray, File)` – принимает на вход строку, для которой хранится суффикный массив, и файл, из которого нужно загрузить сам суффиксный массив.
- `SuffixArray(LargeByteArray, String)` – принимает на вход строку, для которой хранится суффикный массив, и имя файла, из которого нужно загрузить сам суффиксный массив.

Методы

- `save(File):void` – сохраняет массив в файл;
- `save(String):void` – сохраняет массив в файл;
- `get(int):long` – возвращает заданный элемент суффиксного массива;
- `set(int, long):void` – устанавливает элемент суффиксного массива;
- `getChar(int, int):byte` – возвращает символ строки.

- `getCharWithLastZeros(int,int):byte` – возвращает символ строки, если позиция символа выходит за пределы строки, возвращает нулевой символ.

2.4.2 Поиск перекрытий

Класс `Overlapper` находится в пакете `ru.ifmo.genetics.tools.longReadsAssembler.overlaps.overlapper`. Этот класс управляет процессом поиска перекрытий.

Поля класса

- `config:Configuration` – конфигурация всей программы;
- `bucketsDir:File` – папка для файлов промежуточной стадии;
- `numberOfErrors:int` – максимальное число возможных ошибок в перекрывающейся строке;
- `saBuilder:SuffixArrayBuilder` – класс, ответственный за построение суффиксного массива;
- `totalLength:long[]` – длина строки, начиная с начала и кончая i-тым по порядку квазиконтигом;
- `readCount:int` – число чтений;
- `tlen:long[]` – длина i-того по порядку квазиконтига (включая символ $ в начале строки);
- `fullString:LargeByteArray` – вся строка $C_1\$C_2\$C_3\$...C_n\$$, где C_i – i-ый квазиконтиг;
- `loadArrayFromFile:boolean` – загружать ли суффиксный массив из файла (уже построенный)?
- `OUTPUT_OVERLAPS:boolean` – выводить найденные перекрытия (необходимо для дебага);
- `THE_MAP:byte[]` – отображение символов исходного файла;
- `BACKWARD_MAP:char[]` – обратное отображение символов исходного файла;

- `$:byte` – номер отображенного символа ‘$’;
- `DNA:byte[]` – номера отображенных символов нуклеотидов ‘A’,‘T’,‘G’,‘C’.

Конструкторы

- `Overlapper(Configuration, boolean)` – создает класс с заданной конфигурацией.

Методы

- `main(String[]):void` – метод, используемый для запуска алгоритма;
- `run():void` – запускает алгоритм;
- `getAllApproximateOverlapByBuckets():void` – ищет перекрытия при уже построенном суффиксном массиве;
- `before(int, long, int, long, long, IntComparator):boolean` – сравнивает два рида.

Класс `OverlapsList` находится в пакете `ru.ifmo.genetics.tools.longReadsAssembler.overlaps.overlapper`. Этот класс хранит список перекрытий и выполняет действия над ним.

Поля класса

- `size:int` – размер списка;
- `DEFAULT_SIZE:int` – начальный размер списка;
- `tos: int[]` – номер контигов, в которые ведут ребра;
- `shifts: int[]` – сдвиги ребер.

Конструкторы

- `OverlapsList()` – создает класс с начальным размером списка;
- `OverlapsList(Collection<Edge>)` – создает класс с начальным размером списка и добавляет в него все ребра из переданной коллекции.

Методы

- `get(int): Edge` – получает ребро по позиции в списке;

- `setTo(int, int): void` – установливает номер контига, в который ведет ребро;
- `setShift(int, int): void` – установливает сдвиг ребра в заданной позиции;
- `getTo(int): int` – получает номер контига, в который ведет ребро;
- `getShift(int): int` – получает сдвиг ребра в заданной позиции;
- `isEmpty():boolean` – список пуст;
- `size():int` – размер списка;
- `contains(Edge): boolean` – содержит ли список заданное ребро?
- `contains(int, int): boolean` – содержит ли список заданное ребро?
- `capacity(): int` – текущая вместимость списка без расширения;
- `find(int, int): int` – найти ребро;
- `find(Edge): int` – найти ребро;
- `add(Edge): void` – добавить ребро;
- `add(int, int): void` – добавить ребро;
- `ensureCapacity(int): void` – гарантировать заданную вместимость;
- `remove(int): void` – удалить ребро на заданной позиции;
- `removeLast(): void` – удалить последнее ребро;
- `clear():void` – очистить список;
- `sort(OverlapsSortTraits): void` – отсортировать списк;
- `addToSet(HashSet<Edge>): HashSet<Edge>` – добавить в множество все ребра из данного списка;
- `toSet(HashSet<Edge>): HashSet<Edge>` – преобразовать данный список в множество;
- `toSet(): HashSet<Edge>` – преобразовать данный список в множество;

- `unite(OverlapsList): OverlapsList` – объединить списки;
- `toString(): String` – вывести список в строку.

2.4.3 . Поиск ненайденных перекрытий с помощью найденных

Класс `OverlapsSaturator` находится в пакете `ru.ifmo.genetics.tools.longReadsAssembler.overlaps.saturator`. Этот класс выполняет поиск ненайденных перекрытий с помощью найденных.

Поля класса

- `readsFile:String` – файл чтений;
- `overlapsFile:String` – файл перекрытий, полученный на предыдущем этапе;
- `outputFile:String` – файл для вывода новых перекрытий;
- `overlapsSortTraits:OverlapsSortTraits` – класс, содержащий функции для сортировки;
- `overlaps:OverlapsList[]` – прямые ребра;
- `backOverlaps:OverlapsList[]` – обратные ребра;
- `readsCount:int` – число чтений;
- `reads:ArrayList<Dna>` – загруженные чтения.

Конструкторы

- `OverlapsSaturator(Configuration)` – создает класс с заданной конфигурацией.

Методы

- `numberOfErrors(Dna, int, Dna):int` – вычисляет число ошибок между двумя чтениями;
- `numberOfErrors(int, int, int):int` – вычисляет число ошибок между двумя чтениями;
- `addNonDetectedOverlaps():void` – реализует алгоритм нахождения ненайденных перекрытий с помощью найденных;
- `run():void` – запускает алгоритм;

- `main(String[] args):void` – метод, используемый для запуска алгоритма.

2.4.4 . Удаление неверных перекрытий

Класс `OverlapsWeeder` находится в пакете `ru.ifmo.genetics.tools.longReadsAssembler.overlaps.weeder`. Этот класс выполняет выполняет поиск и удаление неверных перекрытий.

Поля класса

- `readsFile:String` – файл чтений;
- `overlapsFile:String` – файл перекрытий, полученный на предыдущем этапе;
- `outputFile:String` – файл для вывода новых перекрытий;
- `percentForElect:double` – минимальный процент одинаковых нуклеотидов при консенсусе;
- `overlapsSortTraits:OverlapsSortTraits` – класс, содержащий функции для сортировки;
- `overlaps:OverlapsList[]` – прямые ребра;
- `backOverlaps:OverlapsList[]` – обратные ребра;
- `readsCount:int` – число чтений;
- `reads:ArrayList<Dna>` – загруженные чтения.

Конструкторы

- `OverlapsWeeder(Configuration)` – создает класс с заданной конфигурацией.

Методы

- `getLocalConensus(int, int, Consensus):Consensus` – вычисляет консенсус чтений;
- `removeFalseOverlaps():void` – реализует алгоритм нахождения неверных перекрытий;
- `run():void` – запускает алгоритм;

- `main(String[]):void` – метод, используемый для запуска алгоритма.

2.4.5 . Вывод контигов

Класс `Layouter` находится в пакете `ru.ifmo.genetics.tools.longReadsAssembler.layouter`. Этот класс ответственен за вывод контигов.

Поля класса

- `maxNumberOfErrors:int` – максимальное число ошибок;
- `percentForElect:double` – минимальный процент одинаковых нуклеотидов при консенсусе;
- `readsFile:String` – файл чтений;
- `contigsFile:String` – файл контигов;
- `overlapsFile:String` – файл перекрытий;
- `inForksFile:String` – файл входящих вилок;
- `overlapsSortTraits:OverlapsSortTraits` – класс, содержащий функции для сортировки;
- `overlaps:OverlapsList[]` – прямые ребра;
- `backOverlaps:OverlapsList[]` – обратные ребра;
- `readsCount:int` – число чтений;
- `inForks:int[]` – положения входящих вилок;
- `inDegree:int[]` – входящая степень контига;
- `vertexQueue:Queue<Integer>` – очередь контигов;
- `was:boolean[]` – контиг просмотрен;
- `reads:ArrayList<Dna>` – массив чтений;
- `contigsMade:int` – число сделанных контигов;
- `forkFail:int` – число контигов, у которых нет консенсуса;
- `inForkFail:int` – число неправильных веток;

- `out:PrintWriter` – выходной поток для записи контигов.

Конструкторы

- `Layouter(Configuration)` – создает класс с заданной конфигурацией.

Методы

- `makeLayout(int, int[], int)` – выводит контиг;
- `dfs(int):int` – запускает рекурсивный процесс добавления квазиконтигов;
- `run():void` – запускает алгоритм;
- `main(String[]):void` – метод, используемый для запуска алгоритма.

2.4.6 . Поиск консенсуса

Класс `NucleotideConsensus` находится в пакете `ru.ifmo.genetics.tools.longReadsAssembler.layouter`. Этот класс находит консенсус нуклеотидов, находящихся на одной позиции. Реализация класса `NucleotideConsensus` достаточно коротка и полностью приведена в листинге 2.

Листинг 2. Реализация класса поиска консенсуса нуклеотидов, находящихся на одной позиции

```
package ru.ifmo.genetics.tools.longReadsAssembler.layouter;

import java.util.Arrays;

public class NucleotideConsensus {
    final int NUCS_N = 4;
    int[] counts = new int[NUCS_N];
    int size = 0;
    final double percentForElect;
    byte maxNuc = -1;
    boolean changed = false;
```

```
    public NucleotideConsensus(double percentForElect) {
        this.percentForElect = percentForElect;
    }


    public void put(byte nuc) {
        counts[nuc]++;
        ++size;
        changed = true;
    }

    public byte get() {
        if (!changed) {
            return maxNuc;
        }
        maxNuc = -1;
        for (byte nuc = 0; nuc < NUCS_N; ++nuc) {
            if (maxNuc == -1 || counts[nuc] > counts[maxNuc])
        {
                maxNuc = nuc;
            }
        }
        double percent = (counts[maxNuc] + 0.0) / size;
        if (size > 4 && percent < percentForElect) {
            return (byte)(-maxNuc - 1);
        }
        changed = false;
        return maxNuc;
    }

    public int size() {
        return size;
    }

    public void reset() {
```

```
        size = 0;
        Arrays.fill(counts, 0);
    }
}
```

Класс Consensus находит консенсус нуклеотидов, находящихся на некоторой части генома.

Поля класса

- percentForElect:double – минимальный процент одинаковых нуклеотидов при консенсусе;
- reads:ArrayList<Dna> – массив чтений;
- ar:ArrayList<NucleotideConsensus> – массив консенсусов нуклеотидов для каждой используемой позиции.

Конструкторы

- Consensus(double, ArrayList<Dna>) – создает класс с выбранным минимальным порогом консенсуса.

Методы

- get(int):byte – получает консенсус для заданной позиции;
- size():int – возвращает число добавленных чтений;
- reset():void – забывает все предыдущие чтения;
- getNucleotideConsensus(int):NucleotideConsensus – получает класс консенсуса для заданной позиции;
- addLayoutPart(LayoutPart):void – добавляет чтение в консенсус.

Выводы по главе 2

1. Описано представление данных о геномной последовательности в программе.
2. Приведено описание программной реализации сборки квазиконтигов.
3. Приведено описание программной реализации сборки контигов из квазиконтигов.

ЗАКЛЮЧЕНИЕ

В результате исследований, выполненных на третьем этапе работ по контракту, были разработаны:

а) алгоритм восстановления фрагментов геномной последовательности по парным чтениям, в том числе:
   1) алгоритм сборки квазиконтигов из исправленных парных чтений геномной последовательности;
   2) алгоритм сборки контигов из квазиконтигов;
   3) теоретическая оценка вычислительной сложности указанных алгоритмов.

б) программная реализация алгоритма восстановления фрагментов геномной последовательности по парным чтениям.

Потенциальными потребителями разработанных алгоритмов и программного комплекса являются организации и предприятия, ведущие разработки и исследования, а также оказывающие услуги в области биотехнологий, фармацевтики и персонифицированной медицины.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *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 Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2011. № 5, с. 81 – 84.

2 *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ. М.: Вильямс. 2011.

3 *Разработка и программная реализация алгоритма исправления ошибок в данных секвенирования*. Промежуточный отчет по этапу II «Разработка метода сборки геномных последовательностей на основе восстановления фрагментов по парным чтениям». НИУ ИТМО. 2011.

4 *Zerbino D. R., Birney E.* Velvet: algorithms for de novo short read assembly using de Bruijn graphs // Genome Res. 2008. Vol. 18. No. 5, pp. 821 – 829.

5 *Pevzner P. A., Tang H., Waterman M. S.* An eulerian path approach to DNA fragment assembly // Proc. Nat. Acad. Sci USA. 2001. Vol. 98. No. 17, pp. 9748 – 9753.

6 *Pevzner P. A., Tang H.* Fragment assembly with double-barreled data. Bioinformatics. 2001 June; 17 Suppl 1:S225 – 233.

7 *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, pp. 1767 – 1777.

8 *FASTA format*. [Электронный ресурс]. – Режим доступа: http://zhanglab.ccmb.med.umich.edu/FASTA/, свободный. Яз. англ. (дата обращения 05.06.2012).

9 *Phred-Quality Base Calling*. [Электронный ресурс]. – Режим доступа: http://www.phrap.com/phred/, свободный. Яз. англ. (дата обращения 05.06.2012).

10 *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 Res. 2008. Vol. 18. No. 5, pp. 810 – 820.

11 *Исенбаев В. В.* Разработка системы секвенирования ДНК с использованием paired-end данных. Бакалаврская работа. СПбГУ ИТМО. 2010. [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/genom/_isenbaev_thesis.pdf, свободный. Яз. русск. (дата обращения 05.06.2012).

12 *Рассел С., Норвиг П.* Исскуственный интеллект: современный подход. М.: Вильямс, 2006.

13 International Human Genome Sequencing Consortium. 2001. Initial sequencing and analysis of the human genome // Nature. 409: 860 – 921.

14 *Okanohara D., Sadakane K.* Practical entropy-compressed rank/select dictionary // Computing Research Repository. arXiv:cs/0610001v1. 2006.

ПРИЛОЖЕНИЕ А

КОПИЯ ПИСЬМА О ПОДАЧЕ ЗАЯВКИ НА РЕГИСТРАЦИЮ ПРОГРАММЫ ДЛЯ ЭВМ

В настоящем приложении приводится копия письма о подаче заявки на регистрацию программы для ЭВМ «Программное средство для сборки квазиконтигов из парных чтений».

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики» (НИУ ИТМО)

Кронверкский пр., д. 49,
г. Санкт-Петербург, 197101
Тел. (812) 232-97-04. Факс (812) 232-23-07
e-mail: od@mail.ifmo.ru
http://www.ifmo.ru

18.05.2012 № 33-05-2/96

В отдел регистрации программ для ЭВМ,
баз данных, топологий ИМС и передачи прав на них
Федерального государственного учреждения «Федеральный институт промышленной собственности Федеральной службы по интеллектуальной собственности, патентам и товарным знакам (ФГУ ФИПС)
Бережковская наб., 30, корп. 1, Москва,
Г-59, ГСП-5, 123995

Направляю Вам на регистрацию программу для ЭВМ **Программное средство для сборки квазиконтигов из парных чтений** правообладателем исключительного права н которую является федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Комплектность заявки указана в приложении.

Приложение:

1.	Заявление (форма РП)	на	1	л.	в	1	экз.
2.	Дополнение к заявлению (форма РП/Доп)	на	2	л.	в	1	экз.
3.	Распечатка исходного текста программы	на	32	л.	в	1	экз.
4.	Реферат	на	1	л.	в	2	экз.
5.	Платежный документ об уплате госпошлины	на	1	л.	в	1	экз.

Всего на 37 листах + 1 платежный документ
Свидетельство прошу выслать по почте

Проректор Никифоров В.О.

«16» 05 2012 г.

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММНОЙ РЕАЛИЗАЦИИ АЛГОРИТМА СБОРКИ КВАЗИКОНТИГОВ

ПАКЕТ RU.IFMO.GENETICS.TOOLS.ASSEMBLING

Класс PairWriter

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

import ru.ifmo.genetics.dna.DnaQ;
import ru.ifmo.genetics.io.Pair;
import ru.ifmo.genetics.tools.Util;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.List;
import java.util.Queue;

public class PairWriter implements Runnable {
    static final int CHARS_IN_LINE = 70;

    Queue<List<Pair<DnaQ>>> queue;
    String filePrefix;

    public PairWriter(Queue<List<Pair<DnaQ>>> queue, String filePrefix) {
        this.queue = queue;
        this.filePrefix = filePrefix;
    }

    @Override
    public void run() {
        boolean finished = false;
        long c = 0;

        long tasksWritten = 0;
        boolean firstTime = true;
        while (!finished) {
            List<Pair<DnaQ>> dnas;
            dnas = queue.poll();
            if (dnas == null) {
                try {
                    Thread.sleep(123);
                } catch (InterruptedException e) {
                    System.err.println("writing thread interrupted");
                    break;
                }
                continue;
            }

            if ((dnas.size() == 1) && (dnas.get(0) == null)) {
                break;
            }

            try {
                Util.dnaQs2DoubleFastaFile(Util.extractFirsts(dnas),   filePrefix   +
          "_1", !firstTime, c);
```

```
                Util.dnaQs2DoubleFastaFile(Util.extractSeconds(dnas),   filePrefix   +
            "_2", !firstTime, c);
             } catch (IOException e) {
                System.err.println("Error while writing long reads");
                e.printStackTrace(System.err);
             }
             c += dnas.size();
             firstTime = false;

        }
    }
}
```

Класс ReadsFiller

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

import org.apache.commons.cli.*;
import org.apache.commons.configuration.*;

import java.util.*;
import java.io.*;
import java.util.concurrent.*;

import ru.ifmo.genetics.framework.*;
import ru.ifmo.genetics.dna.*;
import ru.ifmo.genetics.statistics.Timer;
import ru.ifmo.genetics.io.*;
import ru.ifmo.genetics.tools.assembling.task.*;
import ru.ifmo.genetics.tools.Util;
import ru.ifmo.genetics.tools.CompactDeBruijnGraph;

public class ReadsFiller {
    static String DIR = null;
    static String BUCKETS = null;
    static String DATA = null;

    private final static int TASK_SIZE = 1 << 12;

    private static int WRITE_QUEUE_CAPACITY = 1 << 12;
    private int k;
    private int p; // = k + 1

    private int maxFragmentSize;
    private int minFragmentSize;

    private int availableProcessors;

    private String graphFile;
    private String[] filenames;
    private CompactDeBruijnGraph graph;

    public ReadsFiller(Configuration config, String[] filenames) {
        availableProcessors = config.getInt("available_processors");
        minFragmentSize = config.getInt("min_length");
        maxFragmentSize = config.getInt("max_length");
        k = config.getInt("k");
        p = k + 1;
        graphFile = config.getString("graph");
        this.filenames = filenames;

    }
```

```
private void processPairSource(Source<Pair<DnaQ>> pairSource, ExecutorService
      pool, GlobalContext env) {
    int pairCounter = 0;
    List<Pair<DnaQ>> task = new ArrayList<Pair<DnaQ>>(TASK_SIZE);

    for (Pair<DnaQ> pair : pairSource) {
        pairCounter++;

        task.add(pair);
        if (task.size() == TASK_SIZE) {
            pool.execute(new FillingTask(env, task));

            task = new ArrayList<Pair<DnaQ>>(TASK_SIZE);
        }

    }

    if (task.size() != 0) {
        pool.execute(new FillingTask(env, task));
    }

}

private void fillReadsInMulttipleDataset(MultipleDataset multipleDataset) throws
      InterruptedException, IOException {
    Timer timer = new Timer();
    timer.start();

    Timer oneDatasetTimer = new Timer();

    int fileId = 0;

    Queue<List<DnaQ>> writeQueue = new ConcurrentLinkedQueue<List<DnaQ>>();
    Queue<List<Pair<DnaQ>>>     writeFailedQueue     =     new     Concurrent-
      LinkedQueue<List<Pair<DnaQ>>>();
    GlobalContext env = new GlobalContext(writeQueue, writeFailedQueue, k,
            minFragmentSize, maxFragmentSize, graph);

    int n = multipleDataset.datasets.size();

    for (Dataset dataset : multipleDataset.datasets) {
        System.err.println("Processing dataset " + dataset.name());
        ExecutorService pool = Executors.newFixedThreadPool(availableProcessors);

        Thread writingThread = new Thread(new Writer(writeQueue, DATA + da-
      taset.name()));
        Thread writingFailedThread =
                new Thread(new PairWriter(writeFailedQueue, DATA + dataset.name()
      + "_failed"));
        //        Thread        writingThread        =        new        Thread(new
      ru.ifmo.genetics.tools.assembling.Writer(writeQueue, dataset.name()));
        writingThread.start();
        writingFailedThread.start();
        oneDatasetTimer.start();

        processPairSource(dataset.allPairs(), pool, env);

        System.err.println("Dataset read, waiting for termination");

        Util.shutdownAndAwaitTermination(pool);
        List<DnaQ> endList = new ArrayList<DnaQ>(1);
        endList.add(null);

        writeQueue.add(endList);
```

```
            List<Pair<DnaQ>> endFailedList = new ArrayList<Pair<DnaQ>>(1);
            endFailedList.add(null);

            writeFailedQueue.add(endFailedList);

            writingThread.join();
            writingFailedThread.join();

            ++fileId;
            System.err.println("name = " + dataset.name() + ", fileId/dataset.size =
           " + fileId + "/" + n);
            System.err.println("time = " + oneDatasetTimer.finish());

            double done = ((double) fileId) / n;
            double total = timer.finish() / done / 1000;
            double remained = total * (1 - done);
            double elapsed = total * done;

            System.err.println(100 * done + "% done");
            System.err.println("estimated  total time: " + total + ", remaining: " +
          remained + ", elapsed: "
                     + elapsed);

        }

    }

    private void run() throws IOException, ClassNotFoundException, InterruptedExcep-
          tion {
        Timer timer = new Timer();
        timer.start();

        System.err.println("Loading graph...");
        FileInputStream fis = new FileInputStream(graphFile);
        ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(fis));
        graph = (CompactDeBruijnGraph)ois.readObject();
        ois.close();
        System.err.println("Loading graph done, it took " + timer.finish()/1000. + "
          seconds");

        MultipleDataset multipleDataset = new MultipleDataset(filenames, true);

        fillReadsInMulttipleDataset(multipleDataset);
        System.err.println("total time = " + timer.finish());
    }

    public static void main(String args[]) {
        Options options = new Options();

        options.addOption("h", "help", false, "prints this message");
        options.addOption("c", "config", true,  "sets the config file name, default
          to config.properties");
        options.addOption("o", "output-dir", true,  "sets the ouput file name");
        options.addOption("k", true,  "sets the size of k-mer");
        options.addOption("g", "graph", true,  "sets the graph file name");
        options.addOption("l", "min-length", true,  "sets the minimum insert size");
        options.addOption("L", "max-length", true,  "sets the maximum insert size");

        CommandLineParser parser = new PosixParser();
        CommandLine cmd;
        try {
            cmd = parser.parse(options, args);
        } catch (ParseException e) {
```

```
            e.printStackTrace(System.err);
            return;
        }

        if (cmd.hasOption("help")) {
            new HelpFormatter().printHelp("fill_reads <options> <file>+", options);
            return;
        }

        String configFileName = cmd.getOptionValue("config", "config.properties");
        Configuration config;
        try {
            config = new PropertiesConfiguration(configFileName);
        } catch (ConfigurationException e) {
            e.printStackTrace(System.err);
            return;
        }

        Util.addOptionToConfig(cmd, config, "output-dir", "output_dir");
        Util.addOptionToConfig(cmd, config, "k");
        Util.addOptionToConfig(cmd, config, "graph");
        Util.addOptionToConfig(cmd, config, "min-length", "min_length");
        Util.addOptionToConfig(cmd, config, "max-length", "max_length");

        if (args.length < 4) {
            System.err.println("Usage: ReadsFiller <output dir> <k> <graph-file>
           <file>+");
            System.exit(666);
        }

        // prefix = args[0];
        // prefixFile = args[0];

        DIR = config.getString("output_dir");
        (new File(DIR)).mkdir();
        DATA = DIR + File.separator;
        BUCKETS = DIR + File.separator + "buckets" + File.separator;


        String[] filenames = cmd.getArgs();

        Timer t = new Timer();
        try {
            new ReadsFiller(config, filenames).run();
        } catch (Exception e) {
            System.err.println(e);
            e.printStackTrace(System.err);
            System.exit(42);
        }
        System.err.println("total time = " + t);

    }
}
```

Класс Writer

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

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.*;
```

```
import java.util.concurrent.BlockingQueue;

import ru.ifmo.genetics.dna.DnaQ;
import ru.ifmo.genetics.dna.IDna;
import ru.ifmo.genetics.tools.Util;

public class Writer implements Runnable {
    static final int CHARS_IN_LINE = 70;

    String filePrefix;
    Queue<List<DnaQ>> queue;

    public Writer(Queue<List<DnaQ>> queue, String filePrefix) throws FileNotFoundEx-
           ception {
        this.queue = queue;
        this.filePrefix = filePrefix;
    }

    @Override
    public void run() {
        boolean finished = false;
        long c = 0;

        long tasksWritten = 0;
        boolean firstTime = true;
        while (!finished) {
            List<DnaQ> dnas;
            dnas = queue.poll();
            if (dnas == null) {
                try {
                    Thread.sleep(123);
                } catch (InterruptedException e) {
                    System.err.println("writing thread interrupted");
                    break;
                }
                continue;
            }

            if ((dnas.size() == 1) && (dnas.get(0) == null)) {
                break;
            }

            try {
                Util.dnaQs2DoubleFastaFile(dnas, filePrefix, !firstTime, c);
            } catch (IOException e) {
                System.err.println("Error while writing long reads");
                e.printStackTrace(System.err);
            }
            c += dnas.size();

            firstTime = false;
            ++tasksWritten;
            if ((tasksWritten & 15) == 0) {
                System.err.println((new Date()) + ": " + c + " DNAs written from " +
          tasksWritten + " tasks");
            }

        }
    }
}
```

Класс FillingTask

```
package ru.ifmo.genetics.tools.assembling.task;

import java.io.PrintWriter;
import java.util.*;

import ru.ifmo.genetics.dna.*;
import ru.ifmo.genetics.io.Pair;
import ru.ifmo.genetics.io.formats.Illumina;
import ru.ifmo.genetics.tools.Util;
import ru.ifmo.genetics.tools.CompactDeBruijnGraph;
import ru.ifmo.genetics.statistics.Timer;

public class FillingTask implements Runnable {
    private static byte DEFINETELY_QUALITY = 62;
    private static byte POLYMORPHISM_QUALITY = 2;
    private static int MAX_POLYMORPHISM_NUMBER = 3;

    private static int MAX_VISITED_SIZE = (int) 1e3;
    private final static boolean EDGE_SKIPPING_ENABLED = false;

    private static Illumina illumina = new Illumina();

    private static final double PHRED_THRESHOLD = 0.90;
    private static final double RIGHTNESS_METRIC_THRESHOLD = 1e-4;

    GlobalContext context;
    List<ru.ifmo.genetics.io.Pair<DnaQ>> task;

    int ok = 0;
    int notFound = 0;
          int noAnchor = 0;
    int ambiguous = 0;
    int tooBig = 0;
    int tooShort = 0;
    int lowBranching = 0;
          int dropped = 0;
    int tooPolymorphic = 0;
    int processed = 0;

    PrintWriter graphOut = null;

    public  FillingTask(GlobalContext  context,  List<ru.ifmo.genetics.io.Pair<DnaQ>>
          task) {
        this.context = context;
        this.task = task;
    }

    @Override
    public void run() {
        Timer t = new Timer();
        t.start();

        List<DnaQ> result = new ArrayList<DnaQ>(task.size());
        List<Pair<DnaQ>> failed = new ArrayList<Pair<DnaQ>>(task.size());


        processed = 0;
        for (ru.ifmo.genetics.io.Pair<DnaQ> p : task) {
```

```
            //System.err.println(p.first);
            DnaQ dna = fillRead(p.first, p.second);
            if (dna != null) {
                result.add(dna);
            } else {
                failed.add(p);
            }
            // System.err.print("\r" + dna);
            // System.err.println(i + ": ok = " + ok + ", notfound = " + notFound +
               // ", ambiguous = " + ambiguous + ", tooBig = " + tooBig + ",
          task.size = " + task.size());
        }
        //
        //
        System.err.println("took " + t.finish());
        printStat();
        context.queue.add(result);
        context.failedQueue.add(failed);
    }

    public void printStat() {
        System.err.println("ok = " + ok + ", notFound = " + notFound + ", noAnchor =
          " + noAnchor +    ", ambiguous = " + ambiguous + ", tooBig = " + tooBig +
          ", tooPolymorphic = " + tooPolymorphic + ", lowBranching = " + lowBranch-
          ing +  ", processed = " + processed + ", dropped = " + dropped);
    }
    public DnaQ fillRead(DnaQ left, DnaQ right) {
        ++processed;
        if ((left.length() < context.k + 1) || (right.length() < context.k + 1)) {
            tooShort++;
            return null;
        }
        right = right.complement();
        DnaQ leftEnd = left.substring(0, context.k);
        DnaQ rightEnd = right.substring(0, context.k).reverse();
        long leftVertex = leftEnd.toLong();
        long rightVertex = rightEnd.toLong();

        DnaQ leftHint;
        DnaQ rightHint;

        DnaQBuilder leftDnaBuilder = new DnaQBuilder(context.dnaBuilderCapacity);

        long localMinFragmentSize = context.minFragmentSize;
        long localMaxFragmentSize = context.maxFragmentSize;

        {
            boolean goodFound = false;
            int i;
            for (i = context.k; (!goodFound) && (i < left.length()); ++i) {
                long le = (leftVertex << 2) + left.nucAt(i);
                if (context.graph.containsEdge(le)) {
                    for (int j = context.k - 1; j >= 0; --j) {
                        leftDnaBuilder.append((byte)((leftVertex  >>>  (2*j))  &  3),
          DEFINETELY_QUALITY);
                    }

                    goodFound = true;

                    leftDnaBuilder.append(left.nucAt(i), DEFINETELY_QUALITY);
                }
                leftVertex = le & context.toVertexMask;
                localMinFragmentSize--;
                localMaxFragmentSize--;
```

```
    }
    if (!goodFound) {

        noAnchor++;
        return null;
    }
    leftHint = left.substring(i, left.length());
}

DnaQ rightAppendix;
{
    DnaQBuilder rightAppendixBuilder = new DnaQBuilder(right.length());
    boolean goodFound = false;
    int i;
    for (i = context.k; (!goodFound) && (i < right.length()); ++i) {
        long re = rightVertex + ((long)(right.nucAt(i)) << (2*context.k));
        if (context.graph.containsEdge(re)) {
            goodFound = true;

            rightAppendixBuilder.append(right.nucAt(i      -      context.k),
  DEFINETELY_QUALITY);
        }
        rightVertex = re >>> 2;
        localMinFragmentSize--;
        localMaxFragmentSize--;

    }

    if (!goodFound) {
        noAnchor++;
        return null;
    }
    rightHint = right.substring(i, right.length());
    rightAppendix = rightAppendixBuilder.build().reverse();
}

/*
Map<Long, IDna> leftMap = new HashMap<Long, IDna>();

Set<LIPair> lv = new HashSet<LIPair>();

if (!leftDfs(leftDnaBuilder, leftVertex, context.minHalfPathLength,
        context.maxHalfPathLength, leftMap, lv)) {
    return null;
}

IDna res = rightBfs(rightVertex, context.minHalfPathLength,
        context.maxHalfPathLength, leftMap);

*/
DnaQ res = doubleBfs(leftDnaBuilder, leftVertex, rightVertex,
        localMinFragmentSize - context.k,
        localMaxFragmentSize - context.k,
        leftHint, rightHint);

if (res != null) {
    res = new DnaQ(res, rightAppendix);
    int polymorphismNumber = 0;
    for (int i = 0; i < res.length(); ++i) {
        if (res.phredAt(i) == POLYMORPHISM_QUALITY) {
            ++polymorphismNumber;
        }
    }
    if (polymorphismNumber > MAX_POLYMORPHISM_NUMBER) {
```

```
            tooPolymorphic++;
            return null;
        }
    }

    if (res != null)
        ok++;
    return res;
}

private boolean buildReversePath(DnaQBuilder builder, Set<BfsTreeNode> layer) {
    boolean[] possibleBases = new boolean[4];
    for (BfsTreeNode n: layer) {
        possibleBases[(int)(n.v & 3)] = true;
    }

    int basesCount = 0;
    byte base = -1;

    for (byte i = 0; i < 4; ++i) {
        if (possibleBases[i]) {
            ++basesCount;
            if (base == -1) {
                base = i;
            }
        }
    }

    if (basesCount > 2) {
        ++ambiguous;
        return false;
    }

    Set<BfsTreeNode> nextLayer = new HashSet<BfsTreeNode>(layer.size());

    for (BfsTreeNode n: layer) {
        for (int i = 0; i < 4; ++i) {
            if (n.prev[i] != null) {
                nextLayer.add(n.prev[i]);
            }
        }
    }

    if (nextLayer.isEmpty()) {
        return true;
    }

    if (!buildReversePath(builder, nextLayer)) {
        return false;
    }

    builder.append(base,    (basesCount    ==    1)    ?    DEFINETELY_QUALITY    :
       POLYMORPHISM_QUALITY);
    return true;
}

private boolean buildPath(DnaQBuilder builder, Set<BfsTreeNode> layer) {
    Set<BfsTreeNode> currentLayer = new HashSet<BfsTreeNode>();
    Set<BfsTreeNode> nextLayer = new HashSet<BfsTreeNode>();

    currentLayer.addAll(layer);

    boolean[] possibleBases = new boolean[4];
```

```
    while (true) {
        Arrays.fill(possibleBases, false);

        for (BfsTreeNode n: currentLayer) {
            for (int i = 0; i < 4; ++i) {
                possibleBases[i] |= (n.prev[i] != null);
            }
        }

        int basesCount = 0;
        byte base = -1;

        for (byte i = 0; i < 4; ++i) {
            if (possibleBases[i]) {
                ++basesCount;
                if (base == -1) {
                    base = i;
                }
            }
        }

        if (basesCount > 2) {
            ++ambiguous;
            return false;
        }

        if (basesCount == 0) {
            break;
        }

        builder.append(base,   (basesCount   ==   1)   ?   DEFINETELY_QUALITY   :
      POLYMORPHISM_QUALITY);

        for (BfsTreeNode n: currentLayer) {
            for (int i = 0; i < 4; ++i) {
                if (n.prev[i] != null) {
                    nextLayer.add(n.prev[i]);
                }
            }
        }

        Set<BfsTreeNode> temp = currentLayer;
        currentLayer = nextLayer;
        nextLayer = temp;
        nextLayer.clear();
    }
    return true;
}


private void leftBfsTurn(Map<Long, BfsTreeNode> currentQueue,
        Map<Long, BfsTreeNode> nextQueue, byte nuc, double rightnessFactor) {
    boolean useHint = (rightnessFactor > PHRED_THRESHOLD);
    for (Map.Entry<Long, BfsTreeNode> entry: currentQueue.entrySet()) {
        long v = entry.getKey();
        long e = v << 2;
        for (int i = 0; i < 4; ++i, ++e) {
            double localRightnessFactor = 1;
            if (useHint) {
                if (i == nuc) {
                    localRightnessFactor = rightnessFactor;
                }  else {
                    localRightnessFactor = (1 - rightnessFactor) / 3;
                }
```

```
            }
            boolean shouldBeAdded = context.graph.containsEdge(e);
            if (EDGE_SKIPPING_ENABLED && !shouldBeAdded) {
                long e2 = (e & context.toVertexMask) << 2;
                for (int j2 = 0; j2 < 4; ++j2, ++e2) {
                    if (context.graph.containsEdge(e2)) {
                        shouldBeAdded = true;
                        break;
                    }
                }
            }

            if (shouldBeAdded) {
                long newVertex = e & context.toVertexMask;

                BfsTreeNode node = nextQueue.get(newVertex);
                if (node == null) {
                    node = new BfsTreeNode(newVertex);
                    nextQueue.put(newVertex, node);
                }
                node.prev[(int)(e >>> (2*context.k))] = entry.getValue();
                node.rightnessMetric += entry.getValue().rightnessMetric * lo-
      calRightnessFactor;
            }
        }
    }
}

private void rightBfsTurn(Map<Long, BfsTreeNode> currentQueue,
        Map<Long, BfsTreeNode> nextQueue, byte nuc, double rightnessFactor) {
    boolean useHint = (rightnessFactor > PHRED_THRESHOLD);
    for (Map.Entry<Long, BfsTreeNode> entry: currentQueue.entrySet()) {
        long v = entry.getKey();
        long e = v;
        for (int i = 0; i < 4; ++i, e += context.toVertexMask + 1) {
            double localRightnessFactor = 1;
            if (useHint) {
                if (i == nuc) {
                    localRightnessFactor = rightnessFactor;
                } else {
                    localRightnessFactor = (1 - rightnessFactor) / 3;
                }
            }
            boolean shouldBeAdded = context.graph.containsEdge(e);
            if (EDGE_SKIPPING_ENABLED && !shouldBeAdded) {
                long e2 = e >>> 2;
                for (int j2 = 0; j2 < 4; ++j2, e2 += context.toVertexMask + 1) {
                    if (context.graph.containsEdge(e2)) {
                        shouldBeAdded = true;
                        break;
                    }
                }
            }

            if (shouldBeAdded) {
                long newVertex = e >>> 2;

                BfsTreeNode node = nextQueue.get(newVertex);
                if (node == null) {
                    node = new BfsTreeNode(newVertex);
                    nextQueue.put(newVertex, node);
                }
                node.prev[(int)(v & 3)] = entry.getValue();
```

```
                node.rightnessMetric += entry.getValue().rightnessMetric * lo-
      calRightnessFactor;
            }
        }
    }
}

private void removeIncorrectNodes(Map<Long, BfsTreeNode> queue) {
    Iterator<BfsTreeNode> it = queue.values().iterator();
    while (it.hasNext()) {
        BfsTreeNode node = it.next();
        if (node.rightnessMetric < RIGHTNESS_METRIC_THRESHOLD) {
                dropped++;
            it.remove();
        }
    }
}

private DnaQ doubleBfs(DnaQBuilder leftDnaBuilder, long leftVertex,
        long rightVertex, long minPathLength, long maxPathLength,
        DnaQ leftHint, DnaQ rightHint) {
    Map<Long, BfsTreeNode> leftQueue = new HashMap<Long, BfsTreeNode>();
    Map<Long, BfsTreeNode> leftNextQueue = new HashMap<Long, BfsTreeNode>();
    Map<Long, BfsTreeNode> rightQueue = new HashMap<Long, BfsTreeNode>();
    Map<Long, BfsTreeNode> rightNextQueue = new HashMap<Long, BfsTreeNode>();

    leftQueue.put(leftVertex, new BfsTreeNode(leftVertex, 1));
    rightQueue.put(rightVertex, new BfsTreeNode(rightVertex, 1));

    /*
    // if graphs output is needed - uncomment
    try {
        graphOut = new PrintWriter("graph" + hashCode() + "_" + processed);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    */
    DnaQ res = null;

    int leftLength = 0;
    int rightLength = 0;

    int maxSize = 2;

    boolean isAmbiguous = false;

    if (graphOut != null) {
        graphOut.println("digraph G" + hashCode() + "_" + processed + "{");
        graphOut.println("center=true; rankdir=LR;");
    }
    printLeftLayer(leftQueue.values(), 0);
    printRightLayer(rightQueue.values(), (int) (maxPathLength + 1));
    for (int length = 1; length <= maxPathLength; ++length) {
        byte nuc = 0;
        byte phred = 2;
        if (leftQueue.size() <= rightQueue.size()) { // left
            // System.err.println("left");
            if (leftLength < leftHint.length()) {
                nuc = leftHint.nucAt(leftLength);
                phred = (byte)leftHint.phredAt(leftLength);
            }
            leftBfsTurn(leftQueue,    leftNextQueue,    nuc,    1    -    illumi-
      na.getProbability(phred));
```

```
        Map<Long, BfsTreeNode> oldQueue = leftQueue;
        leftQueue = leftNextQueue;
        leftNextQueue = oldQueue;

        leftNextQueue.clear();

        /*
        for (Long l : leftQueue.keySet()) {
            for (int i = 0; i < leftLength + leftDnaBuilder.length() ; ++i)
                System.err.print(' ');
            System.err.println(Util.long2DnaQString(l, context.k));
        }
        */

        leftLength++;

        if (leftQueue.isEmpty()) {
            break;
        }

        printLeftLayer(leftQueue.values(), leftLength);

        if (leftLength < leftHint.length()) {
            removeIncorrectNodes(leftQueue);
        }

    } else { // right
        // System.err.println("right");
        if (rightLength < rightHint.length()) {
            nuc = rightHint.nucAt(rightLength);
            phred = (byte)rightHint.phredAt(rightLength);
        }
        rightBfsTurn(rightQueue,   rightNextQueue,   nuc,   1   -   illumi-
na.getProbability(phred));

        Map<Long, BfsTreeNode> oldQueue = rightQueue;
        rightQueue = rightNextQueue;
        rightNextQueue = oldQueue;

        rightNextQueue.clear();

        rightLength++;

        if (rightQueue.isEmpty()) {
            break;
        }
        printRightLayer(rightQueue.values(),   (int)(maxPathLength   +   1   -
rightLength));
        if (rightLength < rightHint.length()) {
            removeIncorrectNodes(rightQueue);
        }
    }

    //  System.err.println(length  +  ":  "  +  leftQueue.size()  +  "  +  "  +
rightQueue.size());

    if (length >= minPathLength) {
        Set<BfsTreeNode> leftHalfOfIntersection = new HashSet<BfsTreeNode>();
        Set<BfsTreeNode>         rightHalfOfIntersection         =         new
HashSet<BfsTreeNode>();
        for (Map.Entry<Long, BfsTreeNode> e: leftQueue.entrySet()) {
            if (rightQueue.containsKey(e.getKey())) {
                leftHalfOfIntersection.add(e.getValue());
                rightHalfOfIntersection.add(rightQueue.get(e.getKey()));
```

```
                        if (graphOut != null) {
                            graphOut.println(
                                    dotNodeId(e.getValue(), leftLength) + " -> " +
                                    dotNodeId(rightQueue.get(e.getKey()),
        (int)(maxPathLength + 1 - rightLength)) +
                                    "[ color = \"green\", arrowhead=none ];");
                        }
                    }
                }

                if (!leftHalfOfIntersection.isEmpty()) {
                    if (res != null) {
                        isAmbiguous = true;
//                        ambiguous++;
//                        return null;
                    } else  if (!buildReversePath(leftDnaBuilder, leftHalfOfIntersec-
        tion)) {
//                        return null;
                    } else if (!buildPath(leftDnaBuilder, rightHalfOfIntersection)) {
//                        return null;
                    } else {
                        res = leftDnaBuilder.build();
                    }
                }
            }

            maxSize = Math.max(maxSize, leftQueue.size() + rightQueue.size());

            if (maxSize > MAX_VISITED_SIZE) {
                tooBig++;
                if (graphOut != null) {
                    graphOut.println("}");
                    graphOut.close();
                }
                return null;
            }


        }
        if (graphOut != null) {
            graphOut.println("}");
            graphOut.close();
            if (isAmbiguous)
                System.err.println("graph " + hashCode() + "_" + processed + " was
        ambiguous");
        }

        if (isAmbiguous) {
            ambiguous++;
            return null;
        }

        if (res == null) {
            notFound++;
            if (maxSize == 2) {
                lowBranching++;
            }
        }

        return res;
    }

    private static class BfsTreeNode {
        public long v;
```

```
    public BfsTreeNode[] prev = new BfsTreeNode[4];
    public double rightnessMetric;

    private BfsTreeNode(long v, double rightnessMetric) {
        this.v = v;
        this.rightnessMetric = rightnessMetric;
    }

    public BfsTreeNode(long v) {
        this(v, 0);
    }

    public String toString(int len) {
        return Util.long2DnaQString(v, len);
    }
}

private String dotNodeId(BfsTreeNode node, int level) {
    return "node" + node.toString(context.k) + "_" + level;
}

private void printLayer(Collection<BfsTreeNode> layer, int i) {
    for (BfsTreeNode node: layer) {
        graphOut.printf(
                "%s [ label = \"%s\", shape=point];\n",
                dotNodeId(node, i),
                node.toString(context.k)
                );
    }

    graphOut.print("{ rank=same; ");
    for (BfsTreeNode node: layer) {
        graphOut.print(dotNodeId(node, i));
        graphOut.print("; ");
    }
    graphOut.println("}");

}

private void printLeftLayer(Collection<BfsTreeNode> layer, int i) {
    if (graphOut == null)
        return;
    printLayer(layer, i);
    for (BfsTreeNode node: layer) {
        for (int j = 0; j < 4; ++j) {
            if (node.prev[j] != null) {
                graphOut.printf(
                        "%s -> %s [ label = \"%s\", color=blue];\n",
                        dotNodeId(node.prev[j], i - 1),
                        dotNodeId(node, i),
                        DnaTools.toChar((byte) (node.v & 3)));
            }
        }
    }
}

private void printRightLayer(Collection<BfsTreeNode> layer, int i) {
    if (graphOut == null)
        return;
    printLayer(layer, i);
    for (BfsTreeNode node: layer) {
        for (int j = 0; j < 4; ++j) {
            if (node.prev[j] != null) {
                graphOut.printf(
```

```
                            "%s -> %s [ label = \"%s\", color=red];\n",
                            dotNodeId(node, i),
                            dotNodeId(node.prev[j], i + 1),
                            DnaTools.toChar((byte)(node.v & 3)));
                }
            }
        }
    }

    public static void main(String[] args) {
        // 3 3 0 1 1 0 2|0 3 0 2 0 0
        int k = 2;
        CompactDeBruijnGraph graph = new CompactDeBruijnGraph(k, 128 * 8);
        graph.addEdge(3 * 16L + 0 * 4 + 1);
        graph.addEdge(0 * 16L + 1 * 4 + 1);
        graph.addEdge(1 * 16L + 1 * 4 + 0);
        graph.addEdge(1 * 16L + 0 * 4 + 2);
        graph.addEdge(0 * 16L + 2 * 4 + 3);
        graph.addEdge(2 * 16L + 3 * 4 + 0);
        graph.addEdge(3 * 16L + 0 * 4 + 2);
        graph.addEdge(0 * 16L + 2 * 4 + 0);

        graph.addEdge(1 * 16L + 0 * 4 + 0);
        graph.addEdge(0 * 16L + 0 * 4 + 3);
        graph.addEdge(0 * 16L + 3 * 4 + 0);
        Queue<List<DnaQ>> ans = new ArrayDeque<List<DnaQ>>();
        Queue<List<Pair<DnaQ>>> failed = new ArrayDeque<List<Pair<DnaQ>>>();
        GlobalContext c = new GlobalContext(ans, failed, k, 11, 13, graph);

        DnaQBuilder leftBuilder = new DnaQBuilder();
        leftBuilder.append((byte)3, (byte)10);
        leftBuilder.append((byte)3, (byte)10);
        leftBuilder.append((byte)0, (byte)10);
        leftBuilder.append((byte)1, (byte)10);
        leftBuilder.append((byte)0, (byte)3);

        DnaQ left = leftBuilder.build();

        DnaQBuilder rightBuilder = new DnaQBuilder();
        rightBuilder.append((byte)0, (byte)10);
        rightBuilder.append((byte)0, (byte)10);
        rightBuilder.append((byte)2, (byte)10);
        rightBuilder.append((byte)0, (byte)10);
        rightBuilder.append((byte)3, (byte)10);

        DnaQ right = rightBuilder.build().complement();

        List<ru.ifmo.genetics.io.Pair<DnaQ>>        x        =        new        Ar-
           rayList<ru.ifmo.genetics.io.Pair<DnaQ>>();
        x.add(new ru.ifmo.genetics.io.Pair<DnaQ>(left, right));

        FillingTask t = new FillingTask(c, x);
        t.run();

        List<DnaQ> ansList = ans.poll();

        assert ansList.size() == 1;
        assert ansList.get(0).toString().equals("TAGGAATACA");
        System.out.println("ok");
    }

}
```

Класс GlobalContext

```
package ru.ifmo.genetics.tools.assembling.task;

import java.util.*;

import ru.ifmo.genetics.dna.DnaQ;
import ru.ifmo.genetics.dna.IDna;
import ru.ifmo.genetics.io.Pair;
import ru.ifmo.genetics.tools.set.LongsHashSet;
import ru.ifmo.genetics.tools.CompactDeBruijnGraph;

public class GlobalContext {
    final Queue<List<DnaQ>> queue;
    final Queue<List<Pair<DnaQ>>> failedQueue;
    final int k;
    final int minHalfPathLength;
    final int maxHalfPathLength;

    final int maxFragmentSize; // = averageFragmentSize + fragmentSizeDeviation;
    final int minFragmentSize; // = averageFragmentSize - fragmentSizeDeviation;

    final long toVertexMask;
    final int dnaBuilderCapacity;

    final CompactDeBruijnGraph graph;

    public     GlobalContext(Queue<List<DnaQ>>     queue,     Queue<List<Pair<DnaQ>>>
           failedQueue,
                          int k, int minFragmentSize,
             int maxFragmentSize, CompactDeBruijnGraph graph) {
        this.queue = queue;
        this.failedQueue = failedQueue;
        this.k = k;
        this.minFragmentSize = minFragmentSize;
        this.maxFragmentSize = maxFragmentSize;
        this.minHalfPathLength = (minFragmentSize - k) / 2;
        this.maxHalfPathLength = (maxFragmentSize - k) / 2;
        this.graph = graph;

        toVertexMask = (1L << (2 * k)) - 1;
        dnaBuilderCapacity = maxHalfPathLength + k;
    }
}
```

ПАКЕТ RU.IFMO.GENETICS.TOOLS.SET

Класс BigLongsHashSet

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

import java.util.HashSet;
import java.util.Random;

public class BigLongsHashSet implements LongsHashSet {
    private static final long serialVersionUID = 1L;

    static final int smallCapacityPowerOf2 = 24;    // 16 M  [ * 8 byte = 0.13 Gb]
    static final long smallCapacity = 1 << smallCapacityPowerOf2;
    static final long smallCapacityMask = smallCapacity - 1;

    SmallLongsHashSet[] hs;
```

```
long offset[];

long capacity;
long size = 0;

static final int hashSizePowerOf2 = 16;
static final int hashMask = (1 << hashSizePowerOf2) - 1;
int[] m;

public BigLongsHashSet(long minCapacity, double loadFactor) {
    minCapacity = (long) (minCapacity / loadFactor);

    int cnt = (int) Math.ceil(minCapacity / (double) smallCapacity);
    capacity = smallCapacity * cnt;

    offset = new long[cnt];
    hs = new SmallLongsHashSet[cnt];
    for (int i = 0; i < cnt; i++) {
        hs[i] = new SmallLongsHashSet(smallCapacityPowerOf2);
        offset[i] = i * smallCapacity;
    }

    m = new int[1 << hashSizePowerOf2];
    for (int i = 0; i < m.length; i++) {
        m[i] = i % hs.length;
    }
}

/**
 * @param memoryUsage in bytes
 */
public BigLongsHashSet(long memoryUsage) {
    this(memoryUsage / 8, 1);
}

int getIndex(long v) {
    int i = (int) (v ^ (v >>> 33));
    i = i ^ (i >>> 16);
    i = i & hashMask;
    return m[i];
}

@Override
public boolean contains(long v) {
    int i = getIndex(v);
    return hs[i].contains(v);
}

@Override
public boolean put(long v) {
    int i = getIndex(v);
    boolean res = hs[i].put(v);
    if (res) {
        size++;
    }
    return res;
}

@Override
public long size() {
    return size;
}

@Override
```

```
public long capacity() {
    return capacity;
}

@Override
public long getPosition(long v) {
    int i = getIndex(v);
    long p = hs[i].getPosition(v);
    return (p == -1) ? -1 : (p + offset[i]);
}

@Override
public long elementAt(long i) {
    return  hs[(int)(i  >>>  smallCapacityPowerOf2)].elementAt(i  &  smallCapaci-
      tyMask);
}

// Test
public static void main(String[] args) {
    int cnt = 1000000;
    BigLongsHashSet b = new BigLongsHashSet(500000000);
    Random r = new Random(832449587);

    System.out.println("Testing...");
    boolean g = true;

    HashSet<Long> els = new HashSet<Long>();
    for (int i = 0; i < cnt; i++) {
        long v = -1;
        while ((v == -1) || (els.contains(v))) {
            v = r.nextLong();
        }
        els.add(v);
    }

    for (long v : els) {
        g &= b.put(v);
    }

    g &= (b.size == els.size());

    for (long v : els) {
        g &= !b.put(v);
    }

    for (long v : els) {
        g &= b.contains(v);
    }

    for (int i = 0; i < cnt; i++) {
        long v = -1;
        while ((v == -1) || (els.contains(v))) {
            v = r.nextLong();
        }
        g &= !b.contains(v);
    }

    for (long v : els) {
        long p = b.getPosition(v);
        long gv = b.elementAt(p);
        g &= (gv == v);
    }

    for (int i = 0; i < cnt; i++) {
```

```
            long v = -1;
            while ((v == -1) || (els.contains(v))) {
                v = r.nextLong();
            }
            long p = b.getPosition(v);
            g &= (p == -1);
        }


        if (!g) {
            throw new AssertionError();
        }
        System.out.println("Test passed.");
    }
}
```

Интерфейс LongsHashSet

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

import java.io.*;

public interface LongsHashSet extends Serializable {
    public boolean contains(long v);
    public boolean put(long v);

    public long getPosition(long v);
    public long elementAt(long i);

    public long size();
    public long capacity();
}
```

Класс SmallLongsHashSet

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

import java.util.Arrays;

public class SmallLongsHashSet implements LongsHashSet {
    private static final long serialVersionUID = 1L;

    static final float DEFAULT_LOAD_ASSERTION = 0.90f;

    public static final long FREE = -1;

    long[] t;
    int mask;
    int size = 0;

    public SmallLongsHashSet(int capacityPowerOf2) {
        t = new long[1 << capacityPowerOf2];
        Arrays.fill(t, FREE);

        mask = t.length - 1;
    }

    @Override
    public boolean put(long v) {
        int i = get(v);
        if (t[i] == v) {
            return false;
        } else {
```

```
            t[i] = v;
            size++;
            if (size > t.length * DEFAULT_LOAD_ASSERTION) {
                throw new AssertionError("High load in small longs hash set");
            }
            return true;
        }
    }

    @Override
    public boolean contains(long v) {
        int i = get(v);
        return (t[i] != FREE);  // if v == FREE this condition will return false
    }

    /**
     * Returns index for entry with such long value, or free place for it
     */
    int get(long v) {
        int h = (int) (v ^ (v >>> 33));
        int i = h & mask;
        while ((t[i] != v) && (t[i] != FREE)) {
            i = (i + 1) & mask;
        }
        return i;
    }

    @Override
    public long capacity() {
        return t.length;
    }

    @Override
    public long getPosition(long v) {
        int i = get(v);
        return (t[i] != FREE) ? i : -1;
    }

    @Override
    public long size() {
        return size;
    }

    @Override
    public long elementAt(long i) {
        return t[(int)i];
    }

}
```

ПАКЕТ RU.IFMO.GENETICS.TOOLS

Класс CompactDeBruijnGraph

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

import java.io.Serializable;
import java.util.Arrays;

import ru.ifmo.genetics.dna.IDna;
```

```
import ru.ifmo.genetics.tools.set.*;

public class CompactDeBruijnGraph implements Serializable {
    private static int MAX_K = 30;
    private LongsHashSet edges;

    public final int k;
    public final int k2; // k * 2
    public final long incomeEdgeIncrement;
    public final long vertexMask;

    private final int unusedBits;

    public CompactDeBruijnGraph(int k, long memSize) {
        if ((k > MAX_K) || (k <= 0)) {
            throw new IllegalArgumentException("k should be in range 1.." + MAX_K);
        }
        edges = new BigLongsHashSet(memSize);
        this.k = k;
        k2 = k * 2;
        incomeEdgeIncrement = 1L << k2;
        vertexMask = incomeEdgeIncrement - 1;
        unusedBits = 64 - k2 - 2;
    }

    public void addEdges(IDna dna) {
        if (dna.length() <= k)
            return;
        long cur = 0;
        for (int i = 0; i < k; i++) {
            cur = (cur << 2) | dna.nucAt(i);
        }
        // String s = dnaq.toString();
        for (int i = k; i < dna.length(); i++) {
            cur = cur & vertexMask;
            cur = (cur << 2) | dna.nucAt(i);
            addEdge(cur);
        }
    }

    public boolean containsEdges(IDna dna) {
        if (dna.length() <= k)
            return true;
        long cur = 0;
        for (int i = 0; i < k; i++) {
            cur = (cur << 2) | dna.nucAt(i);
        }
        // String s = dnaq.toString();
        for (int i = k; i < dna.length(); i++) {
            cur = cur & vertexMask;
            cur = (cur << 2) | dna.nucAt(i);
            if (!containsEdge(cur)) {
                return false;
            }
        }
        return true;

    }

    private long reverseComplementEdge(long e) {
        e = ((e & 0x3333333333333333L) << 2) | ((e & 0xccccccccccccccccL) >>> 2);
        e = ((e & 0x0f0f0f0f0f0f0f0fL) << 4) | ((e & 0xf0f0f0f0f0f0f0f0L) >>> 4);
        e = ((e & 0x00ff00ff00ff00ffL) << 8) | ((e & 0xff00ff00ff00ff00L) >>> 8);
        e = ((e & 0x0000ffff0000ffffL) << 16) | ((e & 0xffff0000ffff0000L) >>> 16);
```

```
        e = ((e & 0x00000000ffffffffL) << 32) | ((e & 0xffffffff00000000L) >>> 32);

        e = ~e;

        return e >>> unusedBits;
    }

    private long getEdgesKey(long e, long rcE) {
        return Math.min(e, rcE);
    }
    private long getEdgeKey(long e) {
        return getEdgesKey(e, reverseComplementEdge(e));
    }

    public boolean addEdge(long e) {
        return edges.put(getEdgeKey(e));
    }

    public boolean containsEdge(long e) {
        return edges.contains(getEdgeKey(e));
    }

    public long[] outcomeEdges(long v) {
        long e = v << 2;
        long rcE = reverseComplementEdge(e);
        long[] t = new long[4];
        int tl = 0;
        for (int i = 0; i < 4; i++, e++, rcE -= incomeEdgeIncrement) {
            assert reverseComplementEdge(e) == rcE;
            if (edges.contains(getEdgesKey(e, rcE))) {
                t[tl++] = e;
            }
        }

        long[] res = new long[tl];
        System.arraycopy(t, 0, res, 0, tl);
        return res;
    }

    public long[] incomeEdges(long v) {
        long e = v;
        long rcE = reverseComplementEdge(e);
        long[] t = new long[4];
        int tl = 0;
        for (int i = 0; i < 4; i++, e += incomeEdgeIncrement, rcE--) {
            assert reverseComplementEdge(e) == rcE;
            if (edges.contains(getEdgesKey(e, rcE))) {
                t[tl++] = e;
            }
        }

        long[] res = new long[tl];
        System.arraycopy(t, 0, res, 0, tl);
        return res;
    }

    public static void main(String[] args) {
        int[][] bases = new int[][] {
            { 2, 1, 0, 3, 2, 0, 1, 0, 2, 3, 0, 3, 1, 3, 2, 0, 1},
            { 2, 1, 0, 3, 2, 0, 1, 0, 2, 3, 0, 3, 1, 3, 2, 0, 3},
            { 0, 2, 1, 0, 3, 2, 0, 1, 0, 2, 3, 0, 3, 1, 3, 2, 0},
            { 1, 2, 1, 0, 3, 2, 0, 1, 0, 2, 3, 0, 3, 1, 3, 2, 0},
            { 3, 2, 1, 0, 3, 2, 0, 1, 0, 2, 3, 0, 3, 1, 3, 2, 0},
        };
```

```
        int p = bases[0].length;
        int k = p - 1;
        int sz = bases.length;

        CompactDeBruijnGraph g = new CompactDeBruijnGraph(k, 1024 * 8L);

        long[] es = new long[sz];
        long[] rcEs = new long[sz];
        for (int i = 0; i < sz; ++i) {
            for (int j = 0; j < p; ++j) {
                es[i] <<= 2;
                rcEs[i] <<= 2;

                es[i] += bases[i][j];
                rcEs[i] += bases[i][p - j - 1] ^ 3;
            }
            assert g.reverseComplementEdge(es[i]) == rcEs[i];
            assert g.reverseComplementEdge(rcEs[i]) == es[i];
            g.addEdge(es[i]);
            g.addEdge(rcEs[i]);
            assert g.containsEdge(es[i]);
            assert g.containsEdge(rcEs[i]);
        }

        assert Arrays.equals(g.outcomeEdges(es[0] >>> 2), new long[] {es[0], es[1]});
        assert Arrays.equals(g.incomeEdges(es[0] >>> 2), new long[] {es[2], es[3],
          es[4]});
        System.out.println("ok");
    }

    public long edgesSize() {
        return edges.size();
    }
}
```

Класс Util

```
package ru.ifmo.genetics.tools;
import java.io.*;
import java.util.Iterator;
import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.TimeUnit;

import org.apache.commons.cli.CommandLine;
import org.apache.commons.configuration.Configuration;

import ru.ifmo.genetics.dna.*;
import ru.ifmo.genetics.framework.Dataset;
import ru.ifmo.genetics.io.formats.Illumina;
import ru.ifmo.genetics.io.Pair;

public class Util {

    public static void datasets2BinqFiles(Iterable<Dataset> datasets, String dir)
        throws IOException {
        for (Dataset dataset : datasets) {
            dataset2BinqFile(dataset, dir);
        }
    }

    public static void datasets2FastqFiles(Iterable<Dataset> datasets, String dir)
        throws IOException {
```

```
    for (Dataset dataset : datasets) {
        dataset2FastqFile(dataset, dir);
    }
}

public static void putInt(int v, OutputStream out) throws IOException {
    out.write((v >>> 24) & 0xFF);
    out.write((v >>> 16) & 0xFF);
    out.write((v >>>  8) & 0xFF);
    out.write((v >>>  0) & 0xFF);
}

public static void putByteArray(byte[] array, OutputStream out) throws IOExcep-
      tion {
    putInt(array.length, out);
    out.write(array);
}

public static int getInt(InputStream in) throws IOException {
    int ch1 = in.read();
    int ch2 = in.read();
    int ch3 = in.read();
    int ch4 = in.read();
    if ((ch1 | ch2 | ch3 | ch4) < 0)
        throw new EOFException();
    return (ch1 << 24) + (ch2 << 16) + (ch3 << 8) + (ch4 << 0);
}

public static byte[] getByteArray(InputStream in) throws IOException {
    int len = getInt(in);
    byte[] b = new byte[len];
    in.read(b);
    return b;
}

public static long getPrefixCode(String prefix, int len) {
    long res = 0;
    for (int i = 0; i < prefix.length(); i++) {
        res = (res << 2) | DnaTools.fromChar(prefix.charAt(i));
    }
    res = res << (2 * (len - prefix.length()));
    return res;
}

public static long getPrefixMask(String prefix, int len) {
    long res = 0;
    for (int i = 0; i < prefix.length(); i++) {
        res = (res << 2) | 3;
    }
    res = res << (2 * (len - prefix.length()));
    return res;
}

public static long getCode(String string) {
    long res = 0;
    for (int i = 0; i < string.length(); i++) {
        res = (res << 2) | DnaTools.fromChar(string.charAt(i));
    }
    return res;
}

public static void dnaQs2BinqFile(Iterable<DnaQ> dnaqs, String datasetName,
                                  String filename) throws IOException {
    dnaQs2BinqFile(dnaqs, datasetName, filename, false);
```

```
}

public static void dnaQs2BinqFile(Iterator<DnaQ> dnaqsIt, String datasetName,
                                  String filename, boolean append) throws IOEx-
    ception {
    OutputStream out = new BufferedOutputStream(new FileOutputStream(filename,
      append));
    while (dnaqsIt.hasNext()) {
        putByteArray(dnaqsIt.next().toByteArray(), out);
    }
    out.close();
}

public static void dnaQs2BinqFile(Iterable<DnaQ> dnaqs, String datasetName,
                                  String filename, boolean append) throws IOEx-
    ception {
    dnaQs2BinqFile(dnaqs.iterator(), datasetName, filename, append);
}

public static void dataset2BinqFile(Dataset dataset, String dir) throws IOExcep-
    tion {
    String fullDatasetName = dir + File.separator + dataset.name();
    System.err.println("fullDatasetName = " + fullDatasetName);

    System.err.print(fullDatasetName + "_1.binq...");
    dnaQs2BinqFile(dataset.allFirsts(), fullDatasetName, fullDatasetName +
      "_1.binq");
    System.err.println(" done");

    System.err.print(fullDatasetName + "_2.binq...");
    dnaQs2BinqFile(dataset.allSeconds(), fullDatasetName, fullDatasetName +
      "_2.binq");
    System.err.println(" done");
}

public static void dnaQs2FastqFile(Iterable<DnaQ> dnaqs, String datasetName,
                                   String filename) throws IOException {
    dnaQs2FastqFile(dnaqs, datasetName, filename, false, 0);
}

public static void dnaQs2FastqFile(Iterable<DnaQ> dnaqs, String datasetName,
                                   String filename, boolean append, long
    startNumber) throws IOException {
    Illumina il = new Illumina();
    PrintWriter out = new PrintWriter(new FileOutputStream(filename, append));
    long i = startNumber;
    for (DnaQ dnaq : dnaqs) {
        out.println("@" + datasetName + ":" + ++i + "#0/1");
        out.println(Util.DnaQ2String(dnaq));
        out.println("+");
        for (int j = 0; j < dnaq.length(); ++j) {
            out.print(il.getPhredChar((byte) dnaq.phredAt(j)));
        }
        out.println();
    }
    out.close();
}

public static void dnaQs2DoubleFastaFile(Iterable<DnaQ> dnaqs, String filePrefix)
    throws IOException {
    dnaQs2DoubleFastaFile(dnaqs, filePrefix, false, 0);
}

public static void dnaQs2DoubleFastaFile(Iterable<DnaQ> dnaqs,
```

```
                                                String filePrefix, boolean append, long
      startNumber)
        throws IOException {
    Illumina il = new Illumina();
    PrintWriter out1 = new PrintWriter(new FileOutputStream(filePrefix +
      ".fasta", append));
    PrintWriter out2 = new PrintWriter(new FileOutputStream(filePrefix + ".qual",
      append));
    long i = startNumber;
    String baseName = filePrefix;
    int id = 1;
    if (filePrefix.endsWith("_1") || filePrefix.endsWith("_2")) {
        baseName = baseName.substring(0, baseName.length() - 2);
        id = filePrefix.endsWith("_2") ? 2 : 1;
    }
    for (DnaQ dnaq : dnaqs) {
        out1.println(">" + baseName + ":" + i + "#0/" + id);
        out2.println(">" + baseName + ":" + i + "#0/" + id);
        ++i;
        out1.println(Util.DnaQ2String(dnaq));
        for (int j = 0; j < dnaq.length(); ++j) {
            out2.print(dnaq.phredAt(j) + " ");
        }
        out2.println();
    }
    out1.close();
    out2.close();
}

public static void dataset2FastqFile(Dataset dataset, String dir) throws IOExcep-
      tion {
    String fullDatasetName = dir + File.separator + dataset.name();

    System.err.print(fullDatasetName + "_1.fastq...");
    dnaQs2FastqFile(dataset.allFirsts(), fullDatasetName, fullDatasetName +
      "_1.fastq");
    System.err.println(" done");

    System.err.print(fullDatasetName + "_2.fastq...");
    dnaQs2FastqFile(dataset.allSeconds(), fullDatasetName, fullDatasetName +
      "_2.fastq");
    System.err.println(" done");
}

public static <K> void addInt(Map<K, Integer> map, K key, int value) {
    Integer t = map.get(key);
    if (t == null)
        t = 0;
    map.put(key, t + value);
}

public static <K> void addLong(Map<K, Long> map, K key, long value) {
    Long t = map.get(key);
    if (t == null)
        t = 0L;
    map.put(key, t + value);
}

public static <K> void incrementInt(Map<K, Integer> map, K key) {
    addInt(map, key, 1);
}

public static <K> void incrementLong(Map<K, Long> map, K key) {
    addLong(map, key, 1L);
```

```
}

public static long DnaQ2Long(DnaQ d) {
    int l = d.length();
    long r = 0;
    for (int i = 0; i < l; ++i) {
        r = (r << 2) | (d.byteAt(i) & 3);
    }
    return r;
}

public static String long2DnaQString(long x, int len) {
    StringBuilder s = new StringBuilder();
    for (int i = len - 1; i >= 0; --i) {
        byte c = (byte) ((x >> (2 * i)) & 3);
        s.append(DnaTools.toChar(c));
    }
    return s.toString(); }

public static String DnaQ2String(DnaQ d) {
    int l = d.length();
    StringBuilder s = new StringBuilder();
    for (int i = 0; i < l; ++i) {
        s.append(DnaTools.toChar(d.byteAt(i)));
    }
    return s.toString();
}

public static String join(Iterable<String> ss, String delimiter) {
    StringBuilder sb = new StringBuilder();
    boolean firstly = true;
    for (String s: ss) {
        if (!firstly) {
            sb.append(delimiter);
        }
        sb.append(s);
        firstly = false;
    }
    return sb.toString();
}

public static void shutdownAndAwaitTermination(ExecutorService pool) {
    pool.shutdown(); // Disable new tasks from being submitted
    try {
        // Wait a while for existing tasks to terminate
        while (!pool.awaitTermination(60, TimeUnit.SECONDS));
    } catch (InterruptedException ie) {
        // (Re-)Cancel if current thread also interrupted
        pool.shutdownNow();
        // Preserve interrupt status
        Thread.currentThread().interrupt();
    }
}

public static int sumOfDnaLengths(IDna... dnas) {
    int sum = 0;
    for (IDna dna : dnas) {
        sum += dna.length();
    }
    return sum;
}

public static <E> Iterable<E> extractFirsts(final Iterable<Pair<E>> pairs) {
    return new Iterable<E>() {
```

```
            public Iterator<E> iterator() {
                return new Iterator<E>() {
                    Iterator<Pair<E>> pairIt = pairs.iterator();
                    public boolean hasNext() {
                        return pairIt.hasNext();
                    }

                    public E next() {
                        return pairIt.next().first;
                    }

                    public void remove() {
                        pairIt.remove();
                    }
                };
            }
        };
    }
    public static <E> Iterable<E> extractSeconds(final Iterable<Pair<E>> pairs) {
        return new Iterable<E>() {
            public Iterator<E> iterator() {
                return new Iterator<E>() {
                    Iterator<Pair<E>> pairIt = pairs.iterator();
                    public boolean hasNext() {
                        return pairIt.hasNext();
                    }

                    public E next() {
                        return pairIt.next().second;
                    }

                    public void remove() {
                        pairIt.remove();
                    }
                };
            }
        };
    }
    public static void swap(int[] x, int i, int j) {
        int t = x[i];
        x[i] = x[j];
        x[j] = t;
    }
    public static void swap(short[] x, int i, int j) {
        short t = x[i];
        x[i] = x[j];
        x[j] = t;
    }

    public  static  void  addOptionToConfig(CommandLine  cmd,  Configuration  config,
          String option) {
        addOptionToConfig(cmd, config, option, option);
    }

    public static void addOptionToConfig(
            CommandLine cmd,
            Configuration config,
            String option,
            String property) {
        if (cmd.hasOption(option)) {
            config.setProperty(property, cmd.getOptionValue(option));
        }
    }
}
```