Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет информационных технологий и программирования
Кафедра компьютерных технологий

Мельников Сергей Вячеславович

Разработка алгоритма разбиения набора парных чтений на части для параллельного восстановления фрагментов геномной последовательности

Научный руководитель: ассистент кафедры КТ НИУ ИТМО
Ф. Н. Царев

Санкт-Петербург
2012

Введение

Актуальность темы. Многие современные задачи биологии и медицины требуют знания генома живых организмов, который состоит из нескольких нуклеотидных последовательностей ДНК.

Скорость развития аппаратных комплексов позволяющих считывать ДНК превышает скорость роста производительности компьютеров, поэтому возникает необходимость в высоко масштабируемых алгоритмах для сборки геномной последовательности.

Объект исследования – масштабируемый алгоритм сборки геномной последовательности из данных paired-end секвенирования.

Исследование состоит из следующих частей:

разработка алгоритма разбиения набора парных чтений получаемых в результате paired-end секвенирования на части для дальнейшего параллельного восстановления фрагментов геномной последовательности;

реализация разработанного алгоритма в виде программного обеспечения для распределенной системы ЭВМ;

экспериментальная проверка эффективности разработанного алгоритма.

Научная новизна. В бакалаврской работе разработан метод разбиения набора парных чтений получаемых в результате paired-end секвенирования на части для дальнейшего восстановления фрагментов геномной последовательности. Предложенный алгоритм может быть использован как один из этапов сборки генома, в том числе для больших по размеру геномов с использование кластеров с больших количеством компьютеров.

Кроме того, разработанный метод реализован в виде программного обеспечения для одного компьютера и для кластера, с которым были проведены эксперименты, подтверждающие его работоспособность.

Теоретическая и практическая значимость. Разработанный метод может использоваться как часть процесса секвенирования генома.

Структура работы. Работа состоит из введения и четырех глав.

В первой главе приведен обзор предметной области — рассмотрены необходимые для понимания тематики основы биоинформатики, объяснена актуальность задачи, описаны основные моменты, необходимые для понимания процесса сборки генома. Также в первой главе дается набор терминов, используемых в работе, и их определений.

Во второй главе кратко описаны уже существующие алгоритмы распределенной сборки генома, а также описаны их недостатки.

В третьей главе описан предлагаемый метод.

В четвертой главе описаны экспериментальные исследования, рассмотрены итоги и результаты работы.

Выступления на конференциях. Работа выполняется в рамках государственного контракта №07.514.11.4010 «Разработка алгоритмов сборки геномных последовательностей для вычислительных систем экзафлопсного уровня производительности» (заключен в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы»). Результаты, полученные по результатам данной работы, были доложены на следующих конференциях:

1. *А. В. Александров, С. В. Казаков, С. В. Мельников, А. А. Сергушичев, П. В. Федотов, Ф. Н. Царев.* Метод сборки генома с использованием технологии MapReduce. I Всероссийский конгресс молодых ученых
2. *А. В. Александров, С. В. Казаков, С. В. Мельников, А. А. Сергушичев, П. В. Федотов, Ф. Н. Царев*. Метод сборки генома с использованием технологии MapReduce. СПИСОК-2012
3. *А. В. Александров, С. В. Казаков, С. В. Мельников, А. А. Сергушичев, П. В. Федотов, Ф. Н. Царев*. Метод сборки генома с использованием технологии MapReduce. XXXXII научная и учебно-методическая конференция профессорско-преподавательского и научного состава

Глава 1. Обзор предметной области

1.1. Биоинформатика

Современные задачи, возникающие в биологии и медицине, требуют работы с большим объемом данных. Поэтому применение вычислительных устройств и алгоритмов находит все более широкое применение в этих областях науки.

1.1.1. ДНК

ДНК (дезоксирибонуклеиновая кислота) — химическое вещество, биологический полимер, обеспечивающий хранение и передачу из поколения в поколение генетической информации. В клетках эукариотов (например, животных и растений) ДНК находится в ядре каждой клетки организма.

ДНК состоит из двух противонаправленных последовательностей нуклеотидов. Каждый нуклеотид состоит из азотистого основания, сахара и фосфатной группы. В нуклеотидах могут быть четыре типа азотистых оснований — аденин, цитозин, гуанин и тимин, которые обозначаются соответственно буквами A, C, G и T. Азотистые основания одной цепочки ДНК соединены с азотистыми основаниями другой водородными связями согласно принципу комплементарности — аденин соединяется только с тимином, а гуанин — с цитозином. Таким образом, одна из двух цепочек ДНК однозначно получается из другой путем разворачивания и замены каждого нуклеотида на комплементарный.

Информация, хранящаяся в ДНК организмов, очень важна для их исследования, так как отражает важные свойства живых организмов — наследственность и изменчивость. Так, ДНК особей разных видов различаются значительно сильнее, тем ДНК особей одного вида, а ДНК потомков одной особи значительно больше схожи, чем похожи в среднем ДНК двух особей одного вида.

Еще одно важное приложение исследования ДНК — генетические заболевания. У особей, зараженных одним генетическим заболеванием, наблюдаются одинаковые изменения в ДНК, что может быть использовано в

медицине как для теоретического исследования заболевания, так и для лечения от него.

1.1.2. Задача секвенирования

Секвенирование ДНК — определение нуклеотидной последовательности по имеющемуся образцу ДНК. В результате этого процесса получается линейная цепочка, отражающая последовательность нуклеотидов в ДНК. Существующие технологии не позволяют прочитать нуклеотиды в ДНК не посредственно, для секвенирования ДНК разработаны несколько методов. Одним из современных методов секвенирования ДНК являются методы нового поколения (*next-generation sequencing*).[1]

При секвенировании этими методами ДНК случайным образом дробится на мелкие участки, длиной до 500 нуклеотидов, каждый из которых затем считывается.

Для того, чтобы было возможным восстановить исходную последовательность ДНК, обеспечивается многократное *покрытие* генома чтениями. Пусть имеется набор из N чтений длины l. Пусть также геном имеет длину L. Тогда говорят, что данный набор обеспечивает покрытие генома, равное $\frac{N \cdot l}{L}$.

Для еще большего удешевления процесса считывания используются так называемые парные чтения. При прочтении парных чтений секвенатором выделяется расположенный в случайном месте последовательности ДНК фрагмент, из которого затем считываются префикс и суффикс. Важно отметить, что эти префикс и суффикс считываются с разных нитей ДНК, причем неизвестно, какой был считан с прямой нити, а какой — с обратной.

Результатом работы секвенатора в случае использования парных чтений являются пары последовательностей, про которые известно, на каком расстоянии они располагались в исходной последовательности ДНК.

Современные технологии, например Illumina GAIIx [2], позволяют получить длину фрагмента около 500 нуклеотидов, и длину считанных префиксов и суффиксов до 150 нуклеотидов.

Полученные данные собираются в единую последовательность при помощи специального программного обеспечения. Из-за наличия повторов в геномной последовательности однозначно восстановить саму последовательность нельзя, поэтому требуется восстановить как можно более длинные непрерывные фрагменты геномной последовательности называемые контигами.

1.1.3. Оценка качества секвенирования

Для оценки результатов работы сборщика и, как следствие, для сравнения одного сборщика с другим обычно используется так называемая величина N50, или просто N50.

Пусть имеется набор контигов, для которого нужно посчитать N50. Отсортируем этот набор по длине. Пусть f(N) — суммарная длина всех контигов с длинами хотя бы N. Тогда N50 — максимальное значение N, для которого f(N) не меньше половины длины генома. Вообще говоря, для любого целого числа x, лежащего в промежутке от 1 до 100, Nx - максимальное значение N, для которого f(x) не меньше x процентов от длины генома.

Заметим, что N50 тем больше, чем больше собралось больших контигов. Таким образом, эту величину можно использовать в качестве показателя эффективности работы сборщика. Помимо N50 часто используются такие значения, как N25 и N90.

1.2. Сборка генома

В работе [3] был предложен подход к сборке геномных последовательностей состоящий из нескольких этапов (Рис. 1):

- Исправление ошибок

- Восстановление фрагментов по парным чтениям, такие фрагменты называются квазиконтигами
- Сборка контигов из квазиконтигов

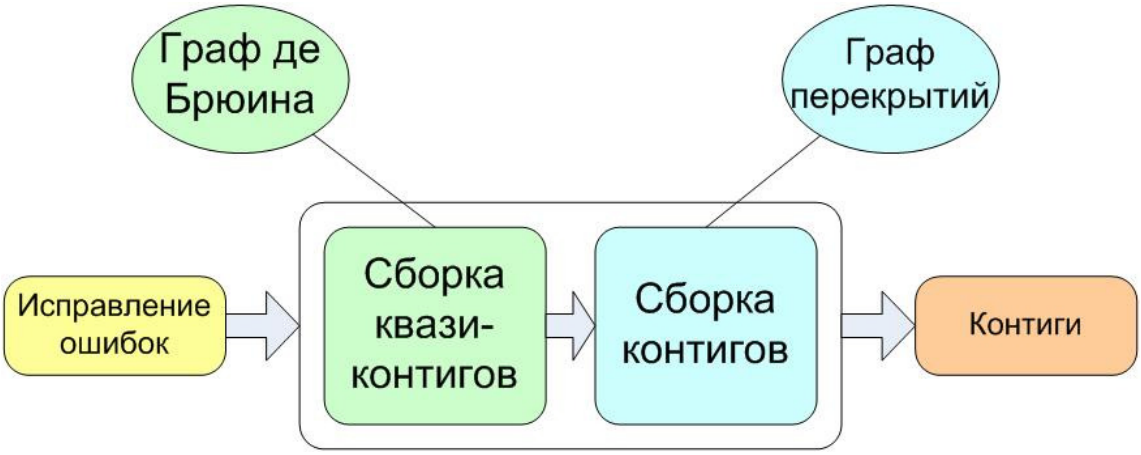

Рис. 1. Архитектура сборщика

Предложенный подход показал свою эффективность в рамках проектов de novo Genome Assembly Project [5] и Assemblathon 2 [6].

1.2.1. Восстановление фрагментов по парным чтениям

Алгоритм восстановления фрагментов по парным чтениям был предложен в работах: [3], [4].

В результате применения данного алгоритма к набору парных чтений, в каждом из них определяется, какие нуклеотиды находятся в геноме между концами парных чтений, и в результате из одной пары чтений получается фрагмент генома, длина которого совпадает с расстоянием между концами парных чтений.

Данный метод используется граф 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, 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, то есть 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

Метод основан на поиске такого пути для фрагмента, соответствующего парным чтениям. Из всех путей интересуют только те, которые укладываются в априорные границы длин фрагментов, поэтому слишком короткие и слишком длинные пути можно отбрасываются. Из оставшихся путей выбираются 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. Метод Overlap-Layout-Consensus

Метод Overlap-Layout-Consensus применяется для сборки протяженных фрагментов геномной последовательности (контигов) из более коротких фрагментов (квазконтигов).

В этом методе сначала производится поиск перекрытий между квазиконтигами. Для этого строится строка вида C1$C2$C3$...Cn$, где Ci – i-ый квазиконтиг, а n – число квазиконтигов.

После этого необходимо строится суффиксный массив для этой строки. С его помощью можно найти все квазиконтиги, в которых встречается заданная подстрока. Это можно сделать, например, с помощью бинарного поиска суффиксов в суффиксном массиве, которые начинаются с заданной подстроки. Они будут располагаться рядом за счет сортировки.

Зафиксируем квазиконтиг, все перекрытия с которым требуется найти. Будем рассматривать его префиксы в порядке увеличения длины, начиная с минимального порога. Для каждого префикса будем проверять, входит ли такая подстрока с добавленным в конце $ в суффиксный массив. Для того чтобы учесть еще и неточные перекрытия, проверяются не только сами префиксы, но и префиксы, в которые внесены небольшие изменения – замена одного символа на другой.

Таким образом, получается граф, в котором вершинами являются квазиконтиги, а ребрами – перекрытия. Этот граф упрощается – из него удаляются транзитивные перекрытия – перекрытия между контигами A и C такие, что существует квазиконтиг B, который перекрывается и с квазиконтигом A, и с квазиконтигом C.

В конце алгоритма полученный граф перекрытий модифицируется с помощью набора эвристик: удаление квазиконтигов, не имеющих перекрытий с другими, удаление небольших «отростков» в графе, объединение путей одинаковой длины и др. Затем происходит построение контигов из последовательностей квазиконтигов без ветвлений.

1.3. Модель Map Reduce

1.3.1. Краткое описание

Map Reduce – подход к написанию распределенных программ, предложенный компанией Google, для упрощения работы над большими объемами данных на больших кластерах [7]. Существует много реализаций этого подхода, которые используются большим числом компаний.

1.3.2. Общая концепция

В модели Map Reduce данные рассматриваются как набор пар ключ-значение над которыми последовательно производятся две фазы действий.

На фазе Map каждая пара ключ-значение обрабатывается независимо, результатом ее обработки может быть ноль, одна или несколько новых пар ключ-значение.

На фазе Reduce данные с одинаковым значение ключа группируются вместе, и функция преобразующая эти данные получает значение ключа и все значения с таким ключом. Результатом обработки может быть ноль, одна или несколько новых пар ключ-значение.

Операции Map и Reduce могут выполняться параллельно на большом числе узлов, передача данных осуществляется только между фазами Map и Reduce.

Исходные данные и результат могут храниться в распределенной файловой системе, благодаря этому возможна обработка больших объемов данных.

1.3.3. Программные реализации

Существует несколько программных реализаций подхода MapReduce таких, как Google MapReduce , Apache Hadoop, GridGain, Hazelcast.

В данной работе применятся реализация Apache Hadoop [8] поскольку она является реализацией с открытым исходным кодом, активно развивается и применяется в крупнейших мировых компаниях для обработки больших объемов данных.

Глава 2. Описание существующих методов распределенной сборки геномной последовательности

2.1. Сборщик ABySS

Сборщик ABySS основан на подходе Message Passing Interface. В качестве основной структуры данных используется граф де Брюина.

Подход к сборке контигов в программном средстве ABySS изложен в статье [9]. Этот подход состоит из двух этапов:

- сборка контигов без учета парной информации;
- разрешение неоднозначностей с помощью парной информации и наращивание контигов.

В основе всего подхода лежит хранение графа де Брюина в распределенной хеш-таблице. Для того, чтобы собрать первоначальные версии контигов происходит объединение последовательностей смежных однозначных ребер — ребро называется однозначным, если исходящая степень его начальной вершины и входящая степень конечной вершины равны единице.

На втором этапе между контигами устанавливаются связи, используя парную информацию. Пара чтений называется связывающей два контига, если первое чтение картируется на первый контиг, а второе — на второй. Между двумя

контигами устанавливается связь, если число связывающих их чтений больше некоторой константы p (по умолчанию используется $p = 5$). Для каждого контига C_i строится множество связанных с ним контигов P_i. Затем в графе связей контигов ищется уникальный путь, проходящий через все контиги из P_i. В качестве ограничений при поиске выступают оценка на расстояния между контигами на основе принципа максимального правдоподобия и эвристическая оценка на число посещенных вершин. После того, как поиск таких путей для каждого контига завершился (успешно или нет), согласующиеся пути сливаются, образуя конечные контиги.

2.2. Сборщик Contrail

В презентации [10] изложен метод работы сборщика, использующего фреймворк MapReduce, что позволяет легко его масштабировать. Этот метод использует граф де Брюина в качестве основной структуры данных.

Для построения графа просматривается каждое чтение и создаются пары (u, v) — ребра графа де Брюина для двух последовательных k-меров u и v. Затем ребра для одного и того же k-мера группируются.

После этого производятся четыре типа упрощений графа: сжатие путей, удаление ошибок, раздвоение вершин, из которых выходит несколько ребер и, если доступны парные чтения, разрешение небольших повторов (Рис. 2). Для сжатия неветвящихся путей используется параллельное рандомизированное ранжирование списков, для исправления ошибок – параллельный поиск шаблонов в сети.

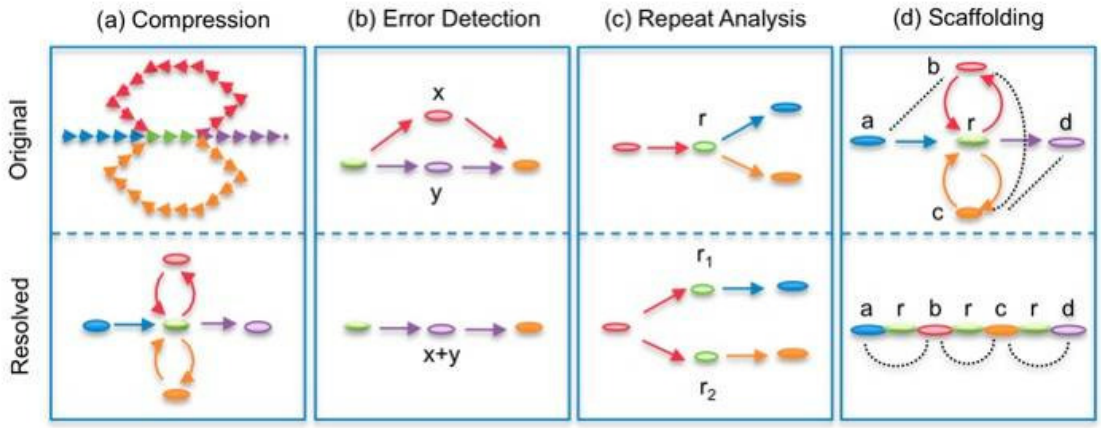

Рис. 2. Упрощения используемые в Contrail

2.3. Сборщик Phusion

Сборщик Phusion основан на идее разбиения входных данных на части и независимой их обработке [11].

Сборка в нем состоит из двух этапов:

- Разбиение чтений на группы, производится на одном узле.
- Независимая сборка контигов в каждой из групп, может производится на большом числе узлов.

Для разбиения чтений на группы строится граф, вершины которого соответствуют чтениям, а вес ребра между вершинами соответствует количеству общих *k*-меров у чтений. Количество вершин этого графа совпадает с количеством чтений, количество ребер пропорционально произведению количества чтений на величину покрытия генома чтениями. Ребра веса меньше некоторой величины M удаляются из рассмотрения.

Рассматриваются компоненты связности получившегося графа. Если размер компоненты связности превосходит величину C, то такие компоненты разбиваются на несколько, величина M внутри этого кластера увеличивается, а соответствующие ребра удаляются. Такие изменения происходят до тех пор, пока размер всех получившихся компонент не будет превосходить C.

Далее каждая компонента обрабатывается независимо, эта обработка может происходить независимо, на разных узлах. Для сборки внутри одной компоненты используется сборщик Phrap, но также может использоваться любой другой сборщик.

2.4. Сравнение описанных методов

Существует несколько подходов к распределенной сборке генома. Один из них основан на распределенном хранении графа де Брюина и распределенной работе с ним, его последовательных модификациях. Граф де Брюина может храниться в виде распределенной хеш-таблицы или в виде набора данных ключ-значение. Такой подход применяется в сборщиках Abyss, в нем хранение осуществляется при помощи распределенной хеш-таблицы, а обработка графа при помощи MPI.

Другой подход заключается в разбиении набора исходных данных на части, и независимой обработке этих частей.

2.5. Достоинства и недостатки описанных методов

Преимуществом первого метода является то, что обрабатываются сразу все данные, таким образом лучше происходит сборка «сложных» фрагментов геномной последовательности, например повторов. Недостатком является сложность алгоритмов осуществляющих данную обработку, а также большой объем данных передаваемых между узлами.

Преимуществом второго метода является независимая обработка частей данных, при их обработке процессы и узлы никак не взаимодействуют, что существенно снижает нагрузку на сеть. Недостатком метода применяемого в сборщике Phusion является необходимость хранения графа с большим количеством вершин на одном узле, для этого требуются узлы с большим объемом оперативной памяти, так для сборки генома описанным методом необходим узел с не менее чем 200 Гб оперативной памяти. Другим недостатком является то, что сборщик Phusion никак не используется информацию о парных чтениях, все чтения в данном сборщике рассматриваются как не парные. В данной работе предлагается метод

лишенный данных недостатков, он позволяет производить разбиение набора парных чтений с использование узлов с небольшим объемом оперативной памяти, при этом парная информация будет использоваться на этапе сборки квазиконтигов.

Глава 3. Описание предлагаемого метода

3.1. Введение

В данной работе предлагается алгоритм позволяющий создать распределенную версию этапа сборки квазиконтигов.

Предлагаемый метод состоит в разбиении набора парных чтений на части. На следующих шагах сборки генома в каждой из полученных частей независимо восстанавливаются фрагменты геномной последовательности.

3.2. Предлагаемый алгоритм

Необходимо разбить набор парных чтений на такие части, чтобы чтения в каждой части были считаны из близких позиций геномной последовательности, и в каждой из таких частей было возможно восстановление квазиконтигов. При этом эти части могут пересекаться, то есть одно чтение может входить более чем в одну часть. Будем называть такое разбиение кластеризацией.

Для кластеризации набора парных чтений используется следующий алгоритм. Строится граф, в котором вершины соответствуют чтениям, а ребро между двумя вершинами есть, если у соответствующих чтений есть общая подстрока длины k. Эта подстрока может входить в любую из частей парного чтения. Граф строится не взвешенный: ребро есть, если есть общий *k*-мер, и отсутствует в противном случае. Пример графа для пяти не парных чтений, и k равного трем представлен на рис. 3.

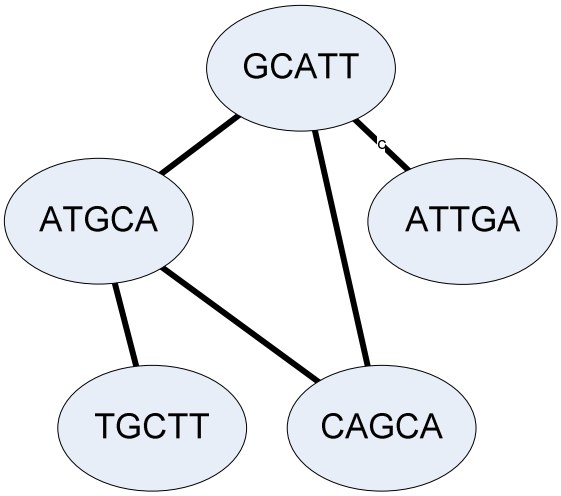

Рис. 3. Пример графа кластеризации для 5 чтений, k = 3

Далее в построенном графе происходит выделение компонент. Одна компонента выделяется следующим образом, выбирается вершина, являющаяся «центром» компоненты, далее в компоненту добавляются все чтения находящиеся на расстоянии не более чем некоторая заданная величина. Так на рис. 4 показана компонента выделенная в графе: исходное чтения обозначено красным цветом, чтения на расстоянии один – желтым, чтения на расстоянии два – зеленым.

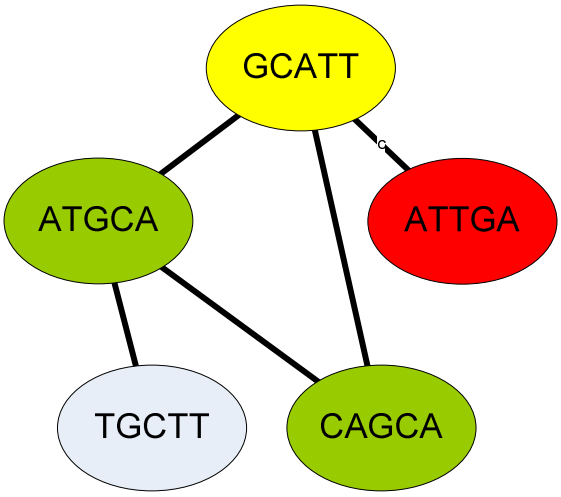

Рис. 4. Компонента в графе для 5 чтений, k = 3

В случае парных чтений ситуация может выглядеть следующим образом, рис. 5. Начальное чтения обозначено красным цветов, чтения на расстоянии один – желтым, чтения на расстоянии два – зеленым.

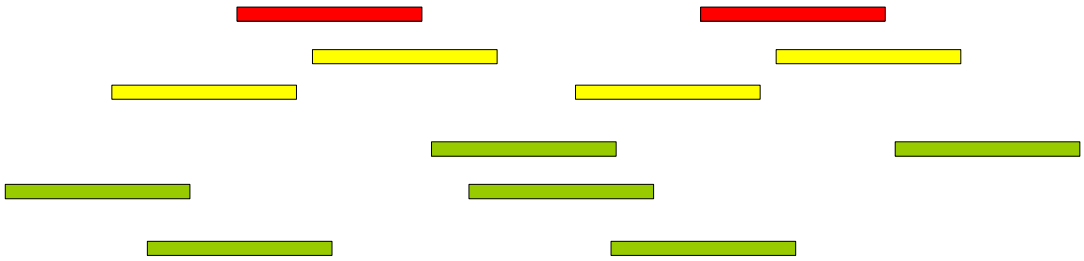

Рис. 5. Чтения на разном расстоянии от начального

3.3. Описание алгоритма на языке MapReduce

Построение графа кластеризации состоит из нескольких шагов

3.3.1. Построение обратного индекса

Для каждого *k*-мера определяется список чтений, в которых он лежит.

На фазе Map для каждого чтения, для всех его *k*-меров выводится набор пар <*k*-мер, чтение>.

На фазе Reduce полученные пары группируются по *k*-меру, и происходит фильтрация неправильных и часто встречающихся в геноме *k*-меров.

Если *k*-мер встречается малое количество раз, сильно меньшее, чем покрытие генома, то, как показано в работе [11], он является ошибочным, и поэтому удаляется из дальнейшего рассмотрения.

Если *k*-мер встречается большое количество раз, в несколько раз превосходящее покрытие генома чтениями, то это значит, что этот *k*-мер является подстрокой частого повтора в геноме, и как показано в результатах экспериментов (раздел 4.3) его можно удалить из рассмотрения без заметного уменьшения размера покрытой части генома. Так на рис. 6 приведен граф для набора чтений последовательности TGCATTGCACA, в которой *k*-мер `GCA` встречается два раза. После удаления это *k*-мера из рассмотрения (рис. 7), граф соответствует упорядочению чтений в исходной последовательности

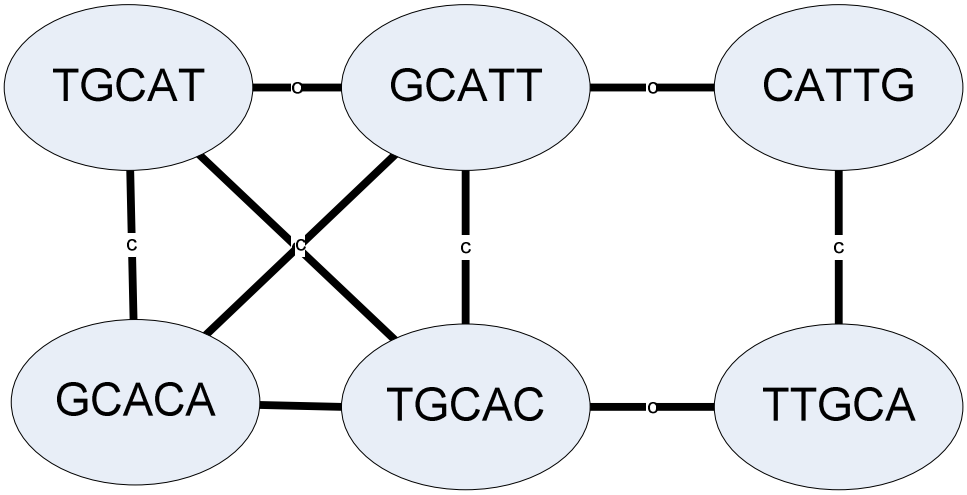

Рис. 6. Граф с часто встречающимся *k*-мером GCA

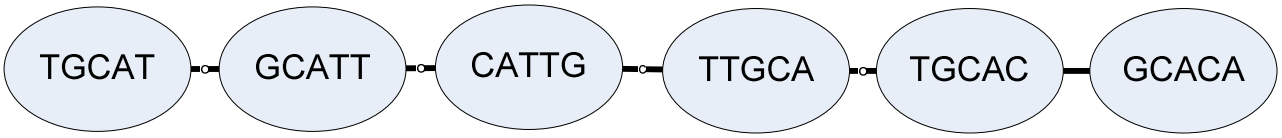

Рис. 7. Граф после удаления из обратного индекса *k*-мера GCA

Для каждого *k*-мера, который остался после выполнения фильтрации выводиться список чтений, в которых он лежит, причем, если *k*-мер входит в чтение более одного раза, то это чтение в список для *k*-мера входит только один раз.

3.3.2. Построение графа

На фазе Map для каждого *k*-мера выводятся все пары чтений, в которых он лежит.

На фазе Reduce одинаковые пары чтений, те которые были выведены на этапе Map один или более раза, группируются вместе и вместо них выводится два направленных ребра, от одного чтения к другому, и в обратную сторону. Каждое ребро представляются собой в виде пары <Вершина, Направленное ребро>

3.3.3. Инициализация компонент

Выбирается набор вершин, из которых будет происходить построение компонент. Для этого вычисляется вероятность, с которой необходимо брать

каждую из вершин в качестве начальной. Эта вероятность равна отношению числа начальных вершин к общему числу вершин.

3.3.4. Шаг обхода в ширину

Выделение компонент для выделения компонент делается несколько шагов обхода в ширину и шагов фильтрации размера компонент.

Шаг обхода в ширину делается следующим образом, аналогичным описанному в работе [13].

На вход подается набор ребер в виде пар <Вершина, Направленное ребро> и текущие компоненты в виде набора пар <Вершина, Компонента> при этом вершина может принадлежать более чем одной компоненте, соответственно для одной вершины может быть несколько таких пар.

На фазе Map ничего не делается.

На фазе Reduce данные группируются по ключу, значению Вершины. Далее в каждой из таких групп для каждой Компоненты, в которой лежит вершина, и для каждого направленного ребра из этой вершины выводиться пара <Вершина, Компонента>.

3.3.5. Фильтрация размера компонент

Компоненты, превысившие некоторый максимальный размер, удаляются из рассмотрения и из вершин в этих компонентах дальнейшие шаги обхода в ширину не производятся.

3.3.6. Получение результата

В каждой из получившихся компонент происходит восстановление фрагментов геномной последовательности. Для этого в начале применяется метод восстановления фрагментов геномной последовательности по парным чтениям. Затем для полученных фрагментов применяется метод Overlap-Layout-Consensus.

3.4. Оценка эффективности предложенного метода

Рассмотрим парные чтения, для того чтобы успешно заполнить промежуток между концами необходимо, чтобы все чтения, прочитанные из позиций между началом и концом, находились в нашей компоненте.

В предложенном алгоритме на каждом шаге обхода в ширину добавляются те чтения, которые имеют с уже добавленными пересечение длины хотя бы k.

Таким образом, на каждом шаге обхода в ширину длина фрагмента генома полностью покрытого чтениями увеличивается не более чем на L - k, а в среднем на $\frac{L-k}{2}$. Пусть D – расстояние между парными чтениями, тогда для того, чтобы весь фрагмент между ними был покрыт необходимо $\frac{D}{\frac{L-k}{2}2} = \frac{D}{L-k}$, так как наращивание покрытого фрагмента происходит с обоих концов

Таким образом, например для чтений бактерии E. Coli [14] необходимо совершить $\frac{200}{36-17} \approx 10$ шагов обхода в ширину

3.5. Метод сборки протяженных фрагментов геномной последовательности

Метод аналогичный предложенному может быть применен для сборки протяженных фрагментов геномной последовательности.

Строится граф, в котором вершины соответствуют квазиконтигам, а две вершины соединяются ребром, если у соответствующих квазиконтигов есть общий *k*-мер.

Далее для этого графа применятся алгоритм кластеризации аналогичный описанному выше, а далее в каждой из полученных компонент применяется метод Overlap-Layout-Consensus.

Глава 4. Реализация алгоритма и полученные результаты

4.1. Реализация алгоритма

В рамках работы была создана реализация предложенного алгоритма на языке программирования Java с использованием фрэймворка Apache Hadoop.

Для последующего восстановления фрагментов сборки использовалась реализация алгоритма восстановления фрагментов нуклеотидной последовательности предложенная в работе [4].

Также был реализован алгоритм сборки контигов описанный в разделе 3.5.

4.2. Экспериментальные исследования

4.2.1. Исходные данные

В качестве исходных данных были использованы парные чтения генома бактерии Escherichia Coli с покрытием 40. Данные парные чтения имеют длину 36 нуклеотидов и расстояние между концами 200 нуклеотидов. Всего было использовано 2 миллиона чтений.

При проведении эксперимента, в исходных данных были исправлены ошибки чтения при помощи специализированного программного средства. [12]

4.2.2. Оборудование

Исследования проводились на вычислительном кластере научно-исследовательского института наукоемких компьютерных технологий (НИИ НКТ) НИУ ИТМО.

Кластер НИИ НКТ НИУ ИТМО состоит из следующих вычислительных узлов:

головного сервера:

- четыре процессора *Intel® Xeon® Processor X5570* с тактовой частотой 2.93 ГГц;
- оперативная память – 24 ГБ с тактовой частотой 1.3 ГГц;
- объем жесткого диска – 3,5 ТБ;

- сетевой адаптер *Gigabit Ethernet*;

десяти вычислительных узлов со следующей конфигурацией:

- два процессора *Intel® Xeon® Processor X5570* с тактовой частотой 2.93 ГГц;
- оперативная память – 24 ГБ с тактовой частотой 1.3 ГГц;
- объем жесткого диска – 250 ГБ;
- сетевой адаптер *Gigabit Ethernet*;

4.2.3. Программное обеспечение

Для проведения экспериментов использовался фреймворк *Hadoop* версии 0.20.205. Он был установлен в каталог головного сервера, который по протоколу *NFS* был подключен ко всем вычислительным узлам.

Каждый из десяти вычислительных узлов был:

- узлом распределенной файловой системы *Hadoop Distributed File System* (сервис `DataNode`), в качестве хранилища данных использовался локальный жесткий диск объемом 250 ГБ;
- вычислительным узлом (сервис `TaskTracker`), на каждом узле одновременно выполнялось не более 8 операций *Map*, и не более 4 операций *Reduce*.

4.2.4. Проведение эксперимента

Далее проведена кластеризация чтений со следующими параметрами:

- Размер *k*-мера: 17;
- Минимальное число чтений, в которые должен входит *k*-мер, при построении графа: 5;
- Максимальное число чтений, в которые может входить *k*-мер, при построении графа: 60;
- Размер компоненты, при котором компонента далее не увеличивается: 10000;

- Число шагов обхода в ширину: 12;
- Число компонент: 100000

В результате получены квазиконтиги со следующими характеристиками:

Количество квазиконтигов	161426
Суммарная длина квазиконтигов	61234287
Максимальная длина квазиконтига	1088
Минимальная длина квазиконтига	150
Средняя длина квазиконтига	379
Метрика N50	452
Метрика N90	215
Часть генома не покрытого квазиконтигами	2.73%

Время работы составило 44 минуты, из которых непосредственно кластеризация заняла 35 минут, и 9 минут заняло восстановления фрагментов по парным чтениям.

С получившимися квазиконтигами были проведены следующие исследования:

Были собраны контиги при помощи программного средства для сборки генома Abyss.

Количество квазиконтигов	3005
Суммарная длина квазиконтигов	4492824
Максимальная длина контига	41728
Минимальная длина контига	30
Средняя длина контига	1495
Метрика N50	9468
Метрика N90	2344
Часть генома не покрытого контигами	4.90%

Были собраны контиги при помощи не распределенной версии сборщика описанной в разделе 1.2.

Количество квазиконтигов	3250
Суммарная длина квазиконтигов	5081736
Максимальная длина контига	15495

Минимальная длина контига	160
Средняя длина контига	1563
Метрика N50	3835
Метрика N90	743
Часть генома не покрытого контигами	3.40%

Были собраны контиги при помощи распределенной версии сборщика описанной в разделе 3.5.

Количество квазиконтигов	10508
Суммарная длина квазиконтигов	19226887
Максимальная длина контига	26677
Минимальная длина контига	150
Средняя длина контига	1829
Метрика N50	4718
Метрика N90	774
Часть генома не покрытого контигами	6.69%

Был проведен запуск сборщика Contrail (раздел 2.2) на тех же входных данных, он показал следующий результат:

Количество квазиконтигов	9477
Суммарная длина квазиконтигов	4515372
Максимальная длина контига	5372
Минимальная длина контига	100
Средняя длина контига	476.455840456
Метрика N50	672
Метрика N90	222
Часть генома не покрытого контигами	4.91%

Время работы составило 100 минут.

Данный эксперимент показывает, что предложенный алгоритм превосходит сборщик Contrail по покрытию генома и по длине контигов.

Необходимо отметить, что предложенный алгоритм является только частью процесса сборки геномной последовательности и полученные фрагменты нуждаются в дальнейшей обработке. При помощи обработки произведенной

несколькими различными сборщиками были получены контиги со значением метрики N50 превосходящей 2000, это показывает применимость предложенного алгоритма для решения задачи распределенной сборки геномных последовательностей.

4.3. Выводы

Предложенный алгоритм показал достаточную эффективность для использования как этап процесса сборки геномных последовательностей.

Заключение

В ходе работы все поставленные задачи были выполнены:

- Был предложен алгоритм, позволяющий эффективно разбивать набор парных чтений на части для параллельного восстановления фрагментов геномной последовательности;
- Была создана реализация предложенного алгоритма на языке Java с использование фрэймворка Apache Hadoop;
- Созданная реализация была протестирована на чтениях бактерии Escherichia Coli.

Источники

1. *Schuster S. C.* Next-generation sequencing transforms today's biology // Nature Methods. 2008. Vol. 5. P. 16–18.
2. Illumina, Inc. http://www.illumina.com/.
3. *Исенбаев В. В., Шалыто А. А.* Разработка системы секвенирования ДНК с использованием paired-end данных. 2010. http://is.ifmo.ru/genom/_isenbaev_thesis.pdf.
4. *Сергушичев А.А., Царев Ф. Н.* Разработка метода восстановления фрагментов нуклеотидной последовательности по парным чтениям. 2011. Бакалаврская работа.
5. De novo Genome Assembly Assesment Project. http://cnag.bsc.es.
6. Assemblathon 2. http://assemblathon.org
7. *Dean J., Ghemawat S.* MapReduce: Simplified Data Processing on Large Clusters http://labs.google.com/papers/mapreduce.html
8. Apache Hadoop. http://hadoop.apache.org/
9. *Simpson J. T., Wong K., Jackman S. D., Schein J. E., Jones S. J. M., Birol I.* Abyss: a parallel assembler for short read sequence data. // Genome Res. Jun 2009. Vol. 19, no. 6. Pp. 1117–1123.
10. *Schat M., Sommer D., Kelley D., Pop M.*. Contrail: Assembly of Large Genomes using Cloud Computting. http://sourceforge.net/apps/mediawiki/contrail-bio/index.php?title=Contrail
11. *Mullikin JC, Ning Z.,* The phusion assembler. // Genome Res. 2003 Jan;Vol. 13 no. 1. Pp. :81-90.
12. *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.
13. *Cohen J.* Graph Twiddling in a MapReduce World // Computing in Science & Engineering. 2009. Vol. 11. № 4, pp. 29 – 41.
14. NCBI: Experiment: SRX000429 – Illumina sequencing of Escherichia coli str. K-12 substr. MG1655 genomic paired-end library

Приложение 1. Исходный код алгоритма

1. Класс *InitComponentsTask*

```
package ru.ifmo.genetics.distributed.clusterization.tasks;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.IdentityReducer;
import ru.ifmo.genetics.distributed.clusterization.types.ComponentID;
import
        ru.ifmo.genetics.distributed.clusterization.types.ComponentIdO
        rEdge;
import ru.ifmo.genetics.distributed.clusterization.types.Vertex;
import ru.ifmo.genetics.distributed.io.KmerIterable;
import
        ru.ifmo.genetics.distributed.io.writable.Int128WritableCompara
        ble;

import java.io.IOException;
import java.util.Random;

public class InitComponentsTask {
    private   static   final   String   COMPONENTS_SELECT_PERCENT   =
         "componentSelectPercent";
    private  static  final  String  DEFAULT_COMPONENTS_SELECT_PERCENT  =
         "0.05";

    public   static   void   initComponents(Path   sourceFolder,   Path
         componentsFolder, double percent) throws IOException {
        final JobConf conf = new JobConf(EdgesBuilderTask.class);

        conf.setJobName("Init components");
        conf.set(COMPONENTS_SELECT_PERCENT, "" + percent);

        FileInputFormat.setInputPaths(conf, sourceFolder);
        FileOutputFormat.setOutputPath(conf, componentsFolder);

        conf.setInputFormat(SequenceFileInputFormat.class);
        conf.setOutputFormat(SequenceFileOutputFormat.class);

        conf.setOutputKeyClass(Vertex.class);
        conf.setOutputValueClass(ComponentIdOrEdge.class);

        conf.setMapperClass(InitComponentsMapper.class);
        conf.setReducerClass(IdentityReducer.class);
        JobClient.runJob(conf);


    }
```

```
    public static class InitComponentsMapper<A extends KmerIterable>
        extends MapReduceBase implements
        Mapper<Int128WritableComparable, A, Vertex, ComponentIdOrEdge>
        {
        private final Vertex outKey = new Vertex();
        private final ComponentID outValueReal = new ComponentID();
        private final ComponentIdOrEdge outValue = new
        ComponentIdOrEdge(outValueReal);
        private final Random random = new Random(1);
        private double componentSelectPercent =
        Double.parseDouble(DEFAULT_COMPONENTS_SELECT_PERCENT);

        @Override
        public void configure(JobConf job) {
            componentSelectPercent =
        Double.parseDouble(job.get(COMPONENTS_SELECT_PERCENT,
        DEFAULT_COMPONENTS_SELECT_PERCENT));
            System.err.println("componentSelectPercent = " +
        componentSelectPercent);
        }

        @Override
        public void map(Int128WritableComparable key, A value,
        OutputCollector<Vertex, ComponentIdOrEdge> output, Reporter
        reporter) throws IOException {
            if (random.nextDouble() < componentSelectPercent) {
                outKey.copyFieldsFrom(key);
                outValueReal.copyFieldsFrom(key);
                output.collect(outKey, outValue);
            }
        }
    }
}
```

2. Класс `ReverseIndexTask`

```
package ru.ifmo.genetics.distributed.clusterization.tasks;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapred.*;
import
        ru.ifmo.genetics.distributed.clusterization.types.Int128ArrayW
        ritable;
import ru.ifmo.genetics.distributed.clusterization.types.Kmer;
import ru.ifmo.genetics.distributed.io.KmerIterable;
import
        ru.ifmo.genetics.distributed.io.writable.Int128WritableCompara
        ble;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
```

```
public class ReverseIndexTask {

    private static final String KMER_LENGTH = "KMER_LENGTH";
    private static final String DEFAULT_KMER_LENGTH = "17";
    public static final String MAXIMUM_OCCURRENCES_THRESHOLD =
         "MAXIMUM_OCCURRENCES_THRESHOLD";
    public static final String DEFAULT_MAXIMUM_OCCURRENCES_THRESHOLD =
         "250";

    private static class Map<A extends KmerIterable> extends
         MapReduceBase implements Mapper<Int128WritableComparable, A,
         LongWritable, Int128WritableComparable> {
        private final LongWritable outKey = new LongWritable();
        private int kmerLength;

        @Override
        public void configure(JobConf job) {
            this.kmerLength = Integer.parseInt(job.get(KMER_LENGTH,
         DEFAULT_KMER_LENGTH));
                    }

        @Override
        public void map(Int128WritableComparable key, A value,
         OutputCollector<LongWritable,        Int128WritableComparable>
         output, Reporter reporter) throws IOException {
            Iterator<Kmer>                                  kmerIterator=
         value.kmerIterator(kmerLength);
            while (kmerIterator.hasNext()) {
                outKey.set(kmerIterator.next().get());
                output.collect(outKey, key);
            }
        }
    }

    private static class Reduce extends MapReduceBase implements
         Reducer<LongWritable, Int128WritableComparable, LongWritable,
         Int128ArrayWritable> {
        private     final     Int128ArrayWritable     law     =     new
         Int128ArrayWritable();
        private int maximumOccurrencesThreshold;

        @Override
        public void configure(JobConf job) {
            maximumOccurrencesThreshold                                 =
         Integer.parseInt(job.get(MAXIMUM_OCCURRENCES_THRESHOLD,
         DEFAULT_MAXIMUM_OCCURRENCES_THRESHOLD));
        }

        @Override
        public          void          reduce(LongWritable          key,
         Iterator<Int128WritableComparable>                        values,
         OutputCollector<LongWritable,   Int128ArrayWritable>   output,
         Reporter reporter) throws IOException {
```

```
            ArrayList<Int128WritableComparable>      al      =      new
        ArrayList<Int128WritableComparable>();
            while (values.hasNext()) {
                Int128WritableComparable value = values.next();
                al.add(new            Int128WritableComparable(value));
        }
            if (al.size() > maximumOccurrencesThreshold) {
                return;
            }
            law.set(al.toArray(new
        Int128WritableComparable[al.size()]));

            output.collect(key, law);
        }

    }

    public   static   void   buildReverseIndex(Path   sourceDir,   Path
         targetDir,  int  kmerLength,  int  maximumOccurrencesThreshold)
         throws IOException {
        JobConf conf = new JobConf(ReverseIndexTask.class);
        conf.set(KMER_LENGTH, "" + kmerLength);
        conf.set(MAXIMUM_OCCURRENCES_THRESHOLD,          ""          +
         maximumOccurrencesThreshold);

        conf.setJobName("ReverseIndex");

        conf.setMapOutputValueClass(Int128WritableComparable.class);
        conf.setOutputKeyClass(LongWritable.class);
        conf.setOutputValueClass(Int128ArrayWritable.class);

        conf.setMapperClass(Map.class);
        conf.setReducerClass(Reduce.class);

        conf.setInputFormat(SequenceFileInputFormat.class);
        conf.setOutputFormat(SequenceFileOutputFormat.class);

        FileInputFormat.setInputPaths(conf, sourceDir);
        FileOutputFormat.setOutputPath(conf, targetDir);


        JobClient.runJob(conf);
    }
}
```

3. Класс *EdgesBuilderTask*

```
package ru.ifmo.genetics.distributed.clusterization.tasks;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.compress.GzipCodec;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.IdentityReducer;
```

```
import ru.ifmo.genetics.distributed.clusterization.types.*;
import
        ru.ifmo.genetics.distributed.io.writable.Int128WritableCompara
        ble;


import java.io.IOException;
import java.util.Iterator;

public class EdgesBuilderTask {


    public    static    final    String    MINIMUM_EDGE_WEIGHT    =
        "MINIMUM_EDGE_WEIGHT";
    public static final String DEFAULT_MINIMUM_EDGE_WEIGHT = "1";

    public static void convertEdgesToDirect(Path oldEdgesFolder, Path
        newEdgesFolder) throws IOException {
        final JobConf conf = new JobConf(EdgesBuilderTask.class);
        conf.setJobName("Convert edges");
        FileInputFormat.setInputPaths(conf, oldEdgesFolder);
        FileOutputFormat.setOutputPath(conf, newEdgesFolder);
        conf.setInputFormat(SequenceFileInputFormat.class);
        conf.setOutputFormat(SequenceFileOutputFormat.class);

        conf.setOutputKeyClass(Vertex.class);
        conf.setOutputValueClass(ComponentIdOrEdge.class);

        conf.setMapperClass(ConvertEdgeToDirect.class);
        conf.setReducerClass(IdentityReducer.class);



        JobClient.runJob(conf);
    }

    private   static   class   Map   extends   MapReduceBase   implements
        Mapper<LongWritable,                        Int128ArrayWritable,
        UndirectUnweightEdge, IntWritable> {

        final UndirectUnweightEdge e = new UndirectUnweightEdge();
        final IntWritable one = new IntWritable(1);
        int numberOfFirst = 20;

        @Override
        public void map(LongWritable key, Int128ArrayWritable value,
         OutputCollector<UndirectUnweightEdge,   IntWritable>   output,
         Reporter reporter) throws IOException {
            int sz = value.get().length;
            Object[] array = value.get();
            int to = sz;
            for (int i = 0; i < to; i++) {
                for (int j = i + 1; j < sz; j++) {
```

```
                Int128WritableComparable            a            =
     ((Int128WritableComparable) array[i]);
                Int128WritableComparable            b            =
     ((Int128WritableComparable) array[j]);
                if (a.compareTo(b) > 0) {
                    e.first = b;
                    e.second = a;
                } else {
                    e.first = a;
                    e.second = b;
                }
                output.collect(e, one);
            }
        }
    }
}

private static class Reduce extends MapReduceBase implements
     Reducer<UndirectUnweightEdge,                    IntWritable,
     UndirectUnweightEdge, IntWritable> {
    IntWritable iw = new IntWritable();

    @Override
    public       void       reduce(UndirectUnweightEdge       key,
     Iterator<IntWritable>                                  values,
     OutputCollector<UndirectUnweightEdge,   IntWritable>   output,
     Reporter reporter) throws IOException {
        int count = 0;
        while (values.hasNext()) {
            count += values.next().get();
        }
        iw.set(count);
        output.collect(key, iw);
    }
}

public static void buildEdges(Path sourceDir, Path targetDir, int
     minimumEdgeWeight) throws IOException {
    JobConf conf = new JobConf(EdgesBuilderTask.class);
    conf.set(MINIMUM_EDGE_WEIGHT, "" + minimumEdgeWeight);
    conf.setJobName("UndirectUnweightEdge builder");

    conf.setOutputKeyClass(UndirectUnweightEdge.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(SequenceFileInputFormat.class);
    conf.setOutputFormat(SequenceFileOutputFormat.class);

    FileInputFormat.setInputPaths(conf, sourceDir);
```

```
        FileOutputFormat.setOutputPath(conf, targetDir);


        JobClient.runJob(conf);
    }

    public static class ConvertEdgeToDirect extends MapReduceBase
         implements Mapper<UndirectUnweightEdge, IntWritable, Vertex,
         ComponentIdOrEdge> {

        final Vertex vertex = new Vertex();
        final DirectEdge directEdge = new DirectEdge();

        final     ComponentIdOrEdge     componentIdOrEdge     =     new
         ComponentIdOrEdge(directEdge);
        private int minimumEdgeWeight;

        @Override
        public void configure(JobConf job) {
            minimumEdgeWeight                                          =
         Integer.parseInt(job.get(MINIMUM_EDGE_WEIGHT,
         DEFAULT_MINIMUM_EDGE_WEIGHT));
        }

        @Override
        public void map(UndirectUnweightEdge key, IntWritable value,
         OutputCollector<Vertex, ComponentIdOrEdge> output, Reporter
         reporter) throws IOException {
            if (value.get() >= minimumEdgeWeight) {
                directEdge.setWeight(value.get());

                vertex.copyFieldsFrom(key.first);
                directEdge.setTo(key.second);
                output.collect(vertex, componentIdOrEdge);

                vertex.copyFieldsFrom(key.first);
                directEdge.setTo(key.second);
                output.collect(vertex, componentIdOrEdge);
            }
        }
    }
}
```

4. Класс *BfsTurnTask*

```
package ru.ifmo.genetics.distributed.clusterization.tasks;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.compress.GzipCodec;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.IdentityMapper;
import ru.ifmo.genetics.distributed.clusterization.types.ComponentID;
```

```
import
        ru.ifmo.genetics.distributed.clusterization.types.ComponentIdO
        rEdge;
import ru.ifmo.genetics.distributed.clusterization.types.DirectEdge;
import ru.ifmo.genetics.distributed.clusterization.types.Vertex;

import java.io.IOException;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Set;

public class BfsTurnTask {
    public   static   void   makeBfsTurn(Path   edgesFolder,   Path
        componentFolder, Path resultFolder) throws IOException {
        final JobConf conf = new JobConf(BfsTurnTask.class);
        final FileSystem fs = FileSystem.get(conf);
        conf.setJobName("BFS turn");

        FileInputFormat.setInputPaths(conf,           componentFolder,
        edgesFolder);

        FileOutputFormat.setOutputPath(conf, resultFolder);
        conf.setInputFormat(SequenceFileInputFormat.class);
        conf.setOutputFormat(SequenceFileOutputFormat.class);

        conf.setOutputKeyClass(Vertex.class);
        conf.setOutputValueClass(ComponentIdOrEdge.class);

        conf.setMapperClass(new                  IdentityMapper<Vertex,
        ComponentIdOrEdge>().getClass());
        conf.setReducerClass(BfsTurnReducer.class);


        JobClient.runJob(conf);

    }

    public   static   class   BfsTurnReducer   extends   MapReduceBase
        implements   Reducer<Vertex,   ComponentIdOrEdge,   Vertex,
        ComponentIdOrEdge> {
        ComponentIdOrEdge componentIdOrEdge = new ComponentIdOrEdge();
        Vertex vertex = new Vertex();

        @Override
        public  void  reduce(Vertex  key,  Iterator<ComponentIdOrEdge>
        values,  OutputCollector<Vertex,  ComponentIdOrEdge>  output,
        Reporter reporter) throws IOException {
            Set<ComponentID>        componentIDs        =        new
        HashSet<ComponentID>();
            Set<DirectEdge> directEdges = new HashSet<DirectEdge>();
            while (values.hasNext()) {
                ComponentIdOrEdge componentIdOrEdge = values.next();
                Writable writable = componentIdOrEdge.get();
```

```
                if (writable instanceof ComponentID) {
                    componentIDs.add((ComponentID) writable);
                } else {
                    directEdges.add((DirectEdge) writable);
                }
            }

            for (ComponentID componentID : componentIDs) {
                componentIdOrEdge.set(componentID);
                output.collect(key, componentIdOrEdge);
                for (DirectEdge directEdge : directEdges) {
                    componentIdOrEdge.set(componentID);
                    vertex.copyFieldsFrom(directEdge.getTo());
                    output.collect(vertex, componentIdOrEdge);
                }
            }
        }
    }
}
```

5. Класс *ExtractComponentTaskPairedDnaQ*

```
package ru.ifmo.genetics.distributed.clusterization.tasks;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.GenericWritable;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.mapred.lib.IdentityMapper;
import org.apache.hadoop.mapred.lib.IdentityReducer;
import
        ru.ifmo.genetics.distributed.clusterization.types.PairedDnaQWi
        thIdWritable;
import ru.ifmo.genetics.distributed.clusterization.types.ComponentID;
import
        ru.ifmo.genetics.distributed.clusterization.types.ComponentIdO
        rEdge;
import ru.ifmo.genetics.distributed.clusterization.types.Vertex;
import
        ru.ifmo.genetics.distributed.io.writable.Int128WritableCompara
        ble;
import ru.ifmo.genetics.distributed.io.writable.PairedDnaQWritable;


import java.io.IOException;
import java.util.*;

public class ExtractComponentTaskPairedDnaQ {

    public static void extractComponent(Path dnaFolder, Path
        sourceFolder, Path tmpFolder, Path resultFolder) throws
        IOException {
        Path tmpFolder1 = new Path(tmpFolder, "1");
        Path tmpFolder2 = new Path(tmpFolder, "2");
```

```
    Path tmpFolder3 = new Path(tmpFolder, "3");
    convertReads(dnaFolder, tmpFolder1);
    convertBFSoutput(sourceFolder, tmpFolder2);
    mergeComponentAndReads(tmpFolder1, tmpFolder2, resultFolder);
}

public static void convertBFSoutput(Path source, Path target)
     throws IOException {
    final          JobConf          conf          =          new
     JobConf(ExtractComponentTaskPairedDnaQ.class);
    final FileSystem fs = FileSystem.get(conf);
    conf.setJobName("Convert BFS output");

    FileInputFormat.setInputPaths(conf, source);
    FileOutputFormat.setOutputPath(conf, target);

    conf.setInputFormat(SequenceFileInputFormat.class);
    conf.setOutputFormat(SequenceFileOutputFormat.class);

    conf.setMapOutputKeyClass(Vertex.class);
    conf.setMapOutputValueClass(ComponentID.class);

    conf.setOutputKeyClass(Vertex.class);

     conf.setOutputValueClass(ComponentIDOrPairedDnaQWithIdWritable
     .class);

    conf.setMapperClass(ConvertBFSOutputMap.class);
    conf.setReducerClass(ConvertBFSOutputMapReduce.class);
    JobClient.runJob(conf);
}

public static void convertReads(Path source, Path target) throws
     IOException {

    final          JobConf          conf          =          new
     JobConf(ExtractComponentTaskPairedDnaQ.class);
    final FileSystem fs = FileSystem.get(conf);
    conf.setJobName("Convert Reads");

    FileInputFormat.setInputPaths(conf, source);
    FileOutputFormat.setOutputPath(conf, target);

    conf.setInputFormat(SequenceFileInputFormat.class);
    conf.setOutputFormat(SequenceFileOutputFormat.class);

    conf.setOutputKeyClass(Vertex.class);

     conf.setOutputValueClass(ComponentIDOrPairedDnaQWithIdWritable
     .class);

    conf.setMapperClass(ConvertReadsMap.class);
    conf.setReducerClass(IdentityReducer.class);
```

```
    JobClient.runJob(conf);
}

public static void mergeComponentAndReads(Path sourceComponent,
     Path sourceReads, Path target) throws IOException {
    final JobConf conf = new
     JobConf(ExtractComponentTaskPairedDnaQ.class);
    final FileSystem fs = FileSystem.get(conf);
    conf.setJobName("Merge components and reads");

    FileInputFormat.setInputPaths(conf, sourceComponent,
     sourceReads);
    FileOutputFormat.setOutputPath(conf, target);

    conf.setInputFormat(SequenceFileInputFormat.class);
    conf.setOutputFormat(SequenceFileOutputFormat.class);

    conf.setMapOutputKeyClass(Vertex.class);

     conf.setMapOutputValueClass(ComponentIDOrPairedDnaQWithIdWrita
     ble.class);


    conf.setOutputKeyClass(ComponentID.class);
    conf.setOutputValueClass(PairedDnaQWithIdWritable.class);

    conf.setMapperClass(IdentityMapper.class);
    conf.setReducerClass(TurnReduce.class);
    JobClient.runJob(conf);
}


public static class ConvertBFSOutputMap extends MapReduceBase
     implements Mapper<Vertex, ComponentIdOrEdge, Vertex,
     ComponentID> {

    @Override
    public void map(Vertex key, ComponentIdOrEdge value,
     OutputCollector<Vertex, ComponentID> output, Reporter
     reporter) throws IOException {
        final Writable writable = value.get();
        if (writable instanceof ComponentID) {
            output.collect(key, (ComponentID) writable);
        }
    }
}

public static class ConvertBFSOutputMapReduce extends
     MapReduceBase implements Reducer<Vertex, ComponentID, Vertex,
     ComponentIDOrPairedDnaQWithIdWritable> {
    final ComponentIDOrPairedDnaQWithIdWritable
     componentIDOrPairedDnaQWithIdWritable = new
     ComponentIDOrPairedDnaQWithIdWritable();
```

```
    final ComponentID componentID = new ComponentID();

    @Override
    public void reduce(Vertex key, Iterator<ComponentID> values,
     OutputCollector<Vertex, ComponentIDOrPairedDnaQWithIdWritable>
     output, Reporter reporter) throws IOException {
        HashSet<Int128WritableComparable>    components    =    new
     HashSet<Int128WritableComparable>();
        while (values.hasNext()) {
            components.add(new
     Int128WritableComparable(values.next()));
        }
        componentIDOrPairedDnaQWithIdWritable.set(componentID);
        for (Int128WritableComparable value : components) {
            componentID.copyFieldsFrom(value);
            output.collect(key,
     componentIDOrPairedDnaQWithIdWritable);
        }
    }
}

public static class ConvertReadsMap extends MapReduceBase
        implements                  Mapper<Int128WritableComparable,
     PairedDnaQWritable,
        Vertex, ComponentIDOrPairedDnaQWithIdWritable> {
    final                       ComponentIDOrPairedDnaQWithIdWritable
     componentIDOrPairedDnaQWithIdWritable =
            new ComponentIDOrPairedDnaQWithIdWritable();
    final  PairedDnaQWithIdWritable  pairedDnaQWithIdWritable  =  new
     PairedDnaQWithIdWritable();
    Vertex vertex = new Vertex();

    @Override
    public       void       map(Int128WritableComparable       key,
     PairedDnaQWritable value,
                    OutputCollector<Vertex,
     ComponentIDOrPairedDnaQWithIdWritable> output,
                    Reporter reporter) throws IOException {

     componentIDOrPairedDnaQWithIdWritable.set(pairedDnaQWithIdWrit
     able);
        pairedDnaQWithIdWritable.first = key;
        pairedDnaQWithIdWritable.second = value;
        vertex.copyFieldsFrom(key);
        output.collect(vertex,
     componentIDOrPairedDnaQWithIdWritable);
    }
}

public  static  class  TurnReduce  extends  MapReduceBase  implements
     Reducer<Vertex,         ComponentIDOrPairedDnaQWithIdWritable,
     ComponentID, PairedDnaQWithIdWritable> {
```

```
        @Override
        public            void            reduce(Vertex            key,
         Iterator<ComponentIDOrPairedDnaQWithIdWritable>        values,
         OutputCollector<ComponentID, PairedDnaQWithIdWritable> output,
         Reporter reporter) throws IOException {
            ArrayList<ComponentID>       componentIDs       =       new
         ArrayList<ComponentID>();
            ArrayList<PairedDnaQWithIdWritable>
         pairedDnaQWithIdWritables                =                new
         ArrayList<PairedDnaQWithIdWritable>();
            while (values.hasNext()) {
                ComponentIDOrPairedDnaQWithIdWritable
         componentIDOrPairedDnaQWithIdWritable = values.next();
                Writable                   writable                   =
         componentIDOrPairedDnaQWithIdWritable.get();
                if (writable instanceof ComponentID) {
                    componentIDs.add((ComponentID) writable);
                } else {

         pairedDnaQWithIdWritables.add((PairedDnaQWithIdWritable)
         writable);
                }
            }
            for (ComponentID componentID : componentIDs) {
                for  (PairedDnaQWithIdWritable  pairedDnaQWithIdWritable
         : pairedDnaQWithIdWritables) {
                    output.collect(componentID,
         pairedDnaQWithIdWritable);
                }
            }
        }
    }


    public  static  class  ComponentIDOrPairedDnaQWithIdWritable  extends
         GenericWritable {
        public ComponentIDOrPairedDnaQWithIdWritable() {
        }

        public ComponentIDOrPairedDnaQWithIdWritable(Writable w) {
            set(w);
        }

        private static final Class[] CLASSES = {
                PairedDnaQWithIdWritable.class,
                ComponentID.class
        };
        @Override
        protected Class[] getTypes() {
            return CLASSES;
        }
    }
}
```