Задача сборки геномов

Задача сборки геномов возникает в следствии того, что мы умеем “читать” лишь короткие нуклеотидные последовательности - как привило в 200-800 нуклеотидов. При этом размер даже бактериальных геномов - это миллионы нуклеотидов, размер генома человека 3,1 млрд нуклеотидов, а геном некоторых растений составляет десятки милиардов нуклеотидов.

Поэтому, чтобы узнать последовательсность генома, необходимо сначала разбить исходные молекулы ДНК на фрагменты (например, ультразвуком), прочесть последовательности этих фрагментов, а затем собрать их в исходную последовательность генома. Сборка этих фрагментов возможна только в случае, если они перекрываются, так что исходно необходимо взять много одинаковых копий генома (например, много клеток одного человека или бактериальную культуру).

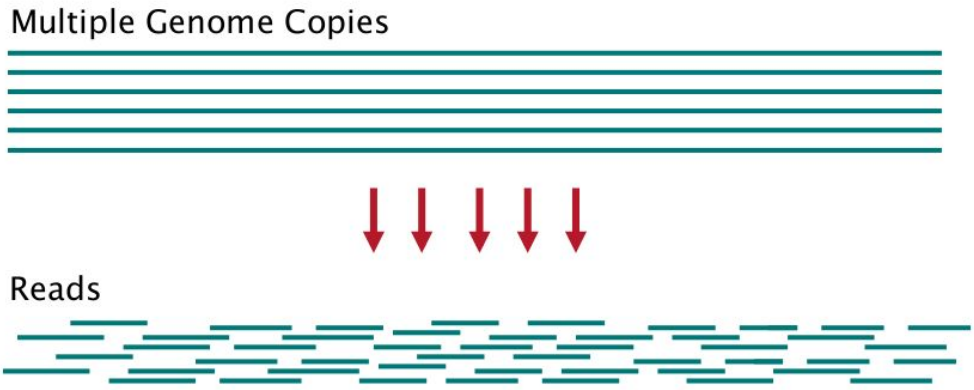


Рис 1. Чтобы прочитать геном целиком, множество его копий сначала разрушается в случайных местах и получается множество перекрывающихся фрагментов, которые потом уже програмно нужно “склеить” в исходную последовательность

OLC-подход

Исторически первым был способ сборки ридов получивший название OLC (Overlap Layout Consensus). Состоит он из трех этапов:

- Overlap (найти перекрывающиеся риды)
- Layout (выстроить риды друг под другом)
- Consensus (найти наиболее вероятный нуклеотид в каждой позиции)

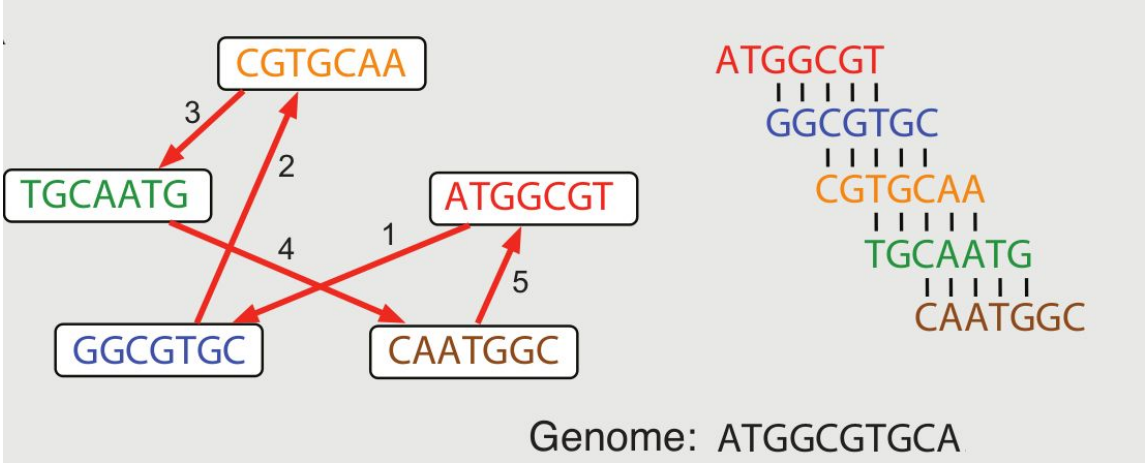


Рис 2. Сборка генома по методу OLC: вначале составляется граф из перекрывающихся ридов, затем в соответствии с обходом графа риды выстраиваются друг под другом и "схлопываются" в последовательность генома

На первом этапе мы ищем перекрытия между ридами. Результат удобно представлять в виде графа, в котором вершинами являются риды, а ребрами мы соединяем вершины, соответствующие перекрывающимся ридам (рис 2, слева).
Чтобы понять как наилучшим образом расположить риды относительно друг друга, нам требуется найти путь в графе, который посещает все вершины графа, ни одну вершину не посещая дважды. Такой путь называется **гамильтоновым путем**. Как только гамильтонов путь найден мы выстраиваем риды друг под другом в соотсветствующем ему порядке (рис 2, справа).
На третьем этапе, мы в каждой позиции выбираем наиболее вероятный нуклеотид (разночтения могут появиться в следствие ошибок секвенирования).
Отметим, что такой подход вычислительно довольно сложен. Чтобы собрать n ридов сперва необходимо произвести n*(n-1) ~ n^2 поисков пересечений последовательностей ридов, а затем найти гамильтонов путь в графе из n вершин.
Задача поиска гамильтонова пути сложно решается классическими компьютерами и относится к классу NP-полных задач, что подразумевает, что не существует алгоритма, котороый бы гарантировано решал ее за полиномиальное время (в худшем случе время будет экспоненциально зависить от количества вершин).

Подход на графах де Брёйна

В 2000-х годах появились высокопроизводительные секвенаторы, способные генерировать до десяти миллионов ридов за один запуск. Сильно возросшее количество ридов привело к тому, что сборка геномов методом OLC стала занимать слишком много времени. Это способствовало расцвету программ-сборщиков, которые работали на основе графов де Брёйна.
Математик Николас де Брёйн (de Bruijn) в 1946-ом году поставил и решил следующую задачу: найти строку минимальной длины, которая содержит в себе все строки длиной k состоящие из символов 0 и 1. Для кратости строки длиной k мы будем называть

k-мерами. Например, при k = 3 у нас есть весемь тримеров: 000, 001, 010, 011, 100, 101, 110, 111, а искомая кротчайшая строка, которая содержит в себе все эти последовательности: 0001110100.

Предложенное де Брейном решение состоит в следующем:

1) сделать все (k-1)-меры вершинами графа;
2) для каждого k-мера соединить вершины, соответвующие его префиксу (строка без последнего символа) и суффиксу (без первого символа) направленным ребром. Так 4-мер 1001 позволит соединить ребром вершины 100 и 001, а 3-мер 000 даст ребро выходящее и входящее в вершину 00;
3) найти такой обход графа, который посещает все ребра и ни одно ребро не посещает дважды;
4) выстроить (k-1) меры в соответствие с обходом и “схлопнув” их получить искомую строку

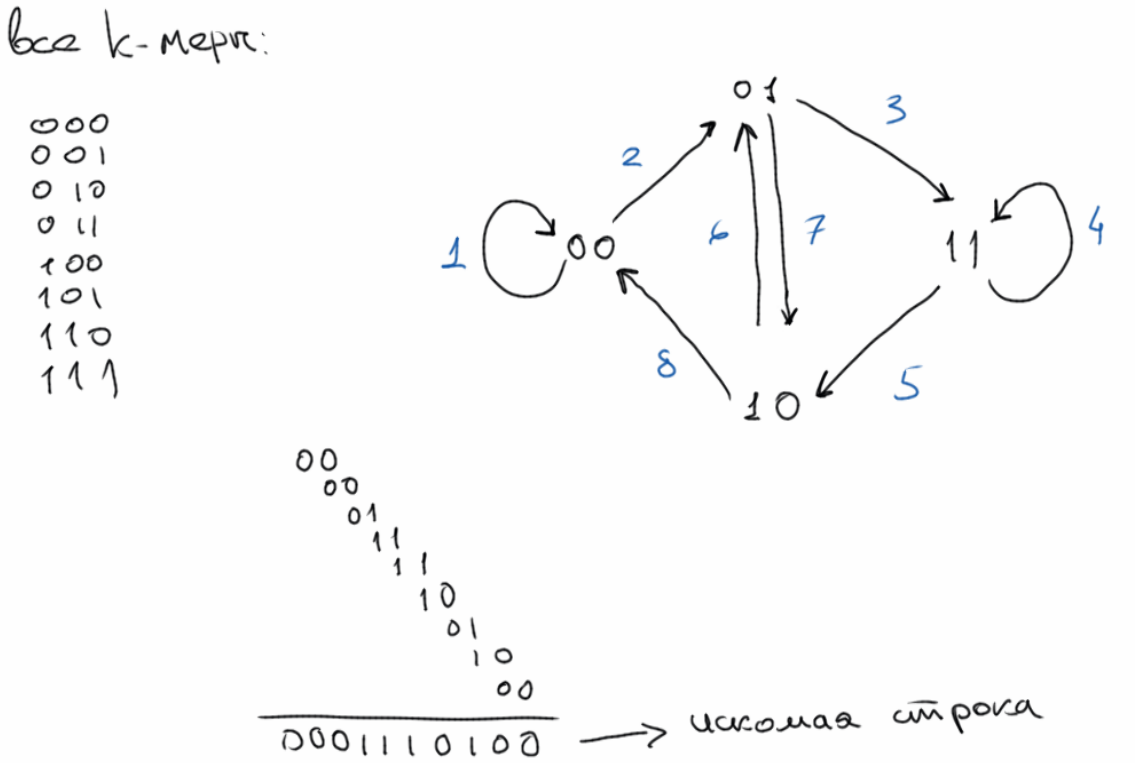


Рис 3. Иллюстрация поиска строки, содержащей все строки длиной 3, состоящие из символов 0 и 1. Справа показан граф де Бреёна, синим обозначен порядок обхода ребер, так чтобы пройти по всем ребрам ровно 1 раз

Такой путь на графе - посещающий каждое ребро ровно один раз - известен как **эйлеров путь**. Задача поиска эйлерова пути, в отличии от задачи поиска гамильтонова пути, вычислительно существенно проще и - при оптимальном алгоритме - может быть решена за линейное от числа ребер время.

Ниже приведен пример псевдокода, реализующего довольно простой алгоритм поиска эйлерова пути в направленном графе. На вход подается некоторая начальная вершина v, множество всех вершин V, множество всех ребер E.

```
function findEulerPath(v):
    for u : u ∈ V
        if deg(u) mod 2 == 1
            v = u
            break
    S.push(v)  // S – стек
    while not S.empty()
        w = S.top()
        for u : u ∈ V
            if (w, u) ∈ E  // нашли ребро, по которому ещё не прошли
                S.push(u)  // добавили новую вершину в стек
                E.remove(w, u)
                break
        if w == S.top()
            S.pop()  // не нашлось инцидентных вершине w рёбер, по которым ещё не прошли
            print(w)
```

Рис 3. Алгоритм поиска Эйрерова пути в направленном графе

Первый цикл for нужен только для графов, которые содержат вершины с нечетным числом ребер, которые могут быть либо началом, либо концом эйлерова пути (вспомните или познакомтесь с методом решения задачи о мостах Кёнигсберга, предложенного Эйлером). Чтобы лучше понять работу алгоритма, попробуйте проследить за состоянием стека и выводимыми на экран значениями для приведенных ниже графов:

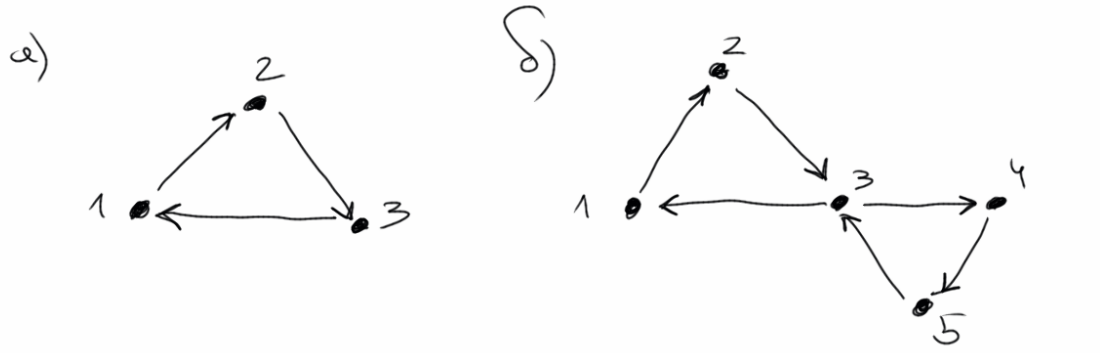


Рис 4. Попробуйте смоделировать работу алгоритма поиска Эйлерова пути на графах а) и б)

Задача, которую поставил де Брейн, схожа с задачей сборки генома: имея все k-меры, представленные в геноме, нам необходимо найти последовательность минимальной длины, которая содержит в себе данные k-меры, что и будет наиболее вероятной последовательностью генома. При этом, основная идея решения этой задачи - представить последовательности строк в виде ребер, а не вершин графа - позволяет быстро находить решение за счет простых алгоритмов поиска Эйлерова пути.
Алгоритм сборки геномов на основе графов де Брейна следующий:

1) Найти все k-меры, содержащиеся в последовательностях ридов (k выбирается заранее и принимает значения в 20-60 букв).
2) Сделать все (k-1) меры вершинами графа и соединять их на основе имеющихся k-меров на основе принципа, описанного выше (ребро исходит из (k-1)-мера, соответствующего префиксу, и вхсодит в (k-1)-мер, соответствующий суффиксу k-мера.
3) Найти эйлеров путь на графе.
4) Выстроить (k-1) меры в соответствии с найденным путем и “схлопнув” их получить итоговою последовательность

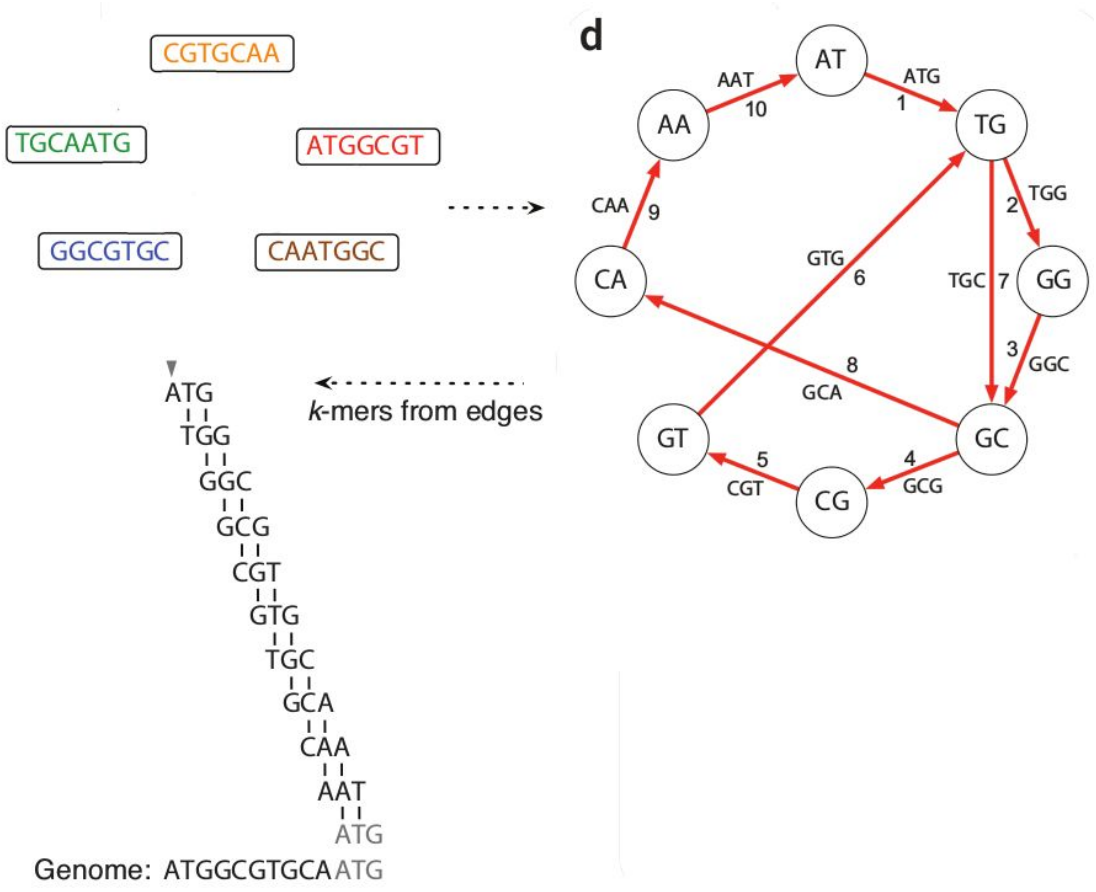


Рис 5. Сборка генома на основе графа де Бреёна. Вначале из ридов получаем все k-меры (в примере k = 3). Делаем все встреченые (k-1) меры вершинами графа, а k-меры - ребрами. Находим эйлеров

путь в графе и в соответствии с ним выстраиваем k-меры. Схлопываем их в "суперстроку", которая содержит все эти k-меры - наиболее вероятную последовательность генома

Такая процедура позволит собрать риды в исходную последовательность генома, но лишь при условии, что в геноме нет повторов, которые по длине больше, чем k. При наличии подобных повторов возникает неоднозначность выбора пути. На рис 6 показан случай, когда в геноме представлен повтор, область которого мы обозначим буквой П. Области слева от него обозначим как А и В, а справа как Б и Г. При построении графа де Брейна у нас будут два равнозначных обхода графа А->П->Г->...->В->П->Б (неправильный) и А->П->Б->...->В->П->Г (правильный). Определить, какой из них правильный, а какой нет мы не сможем - не хватает информации.

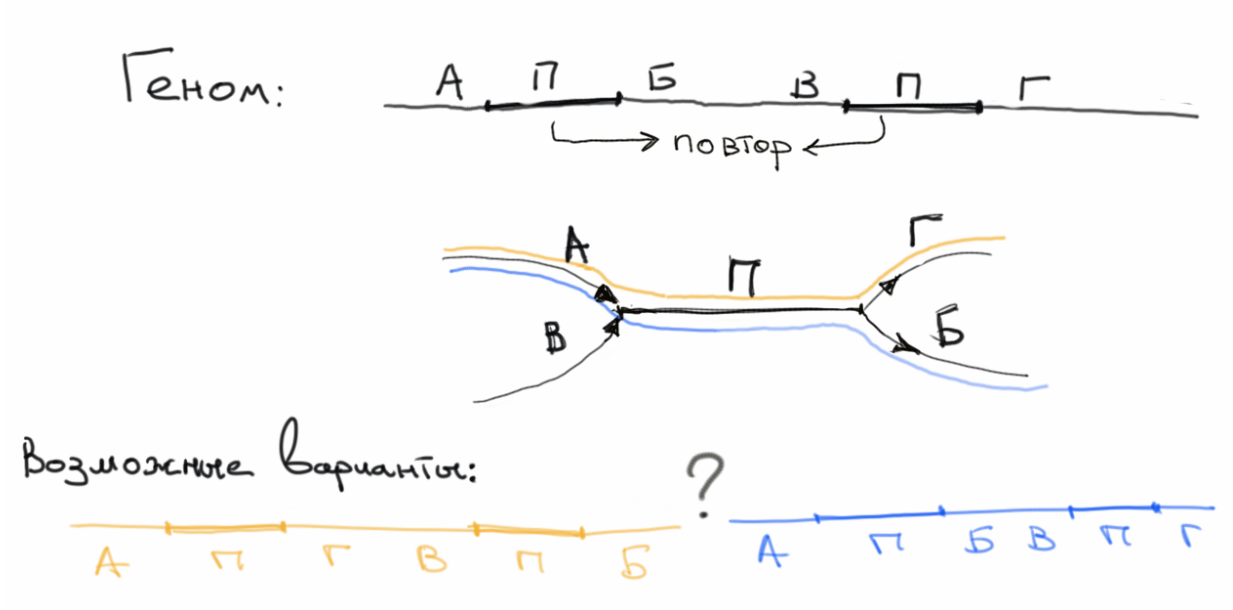


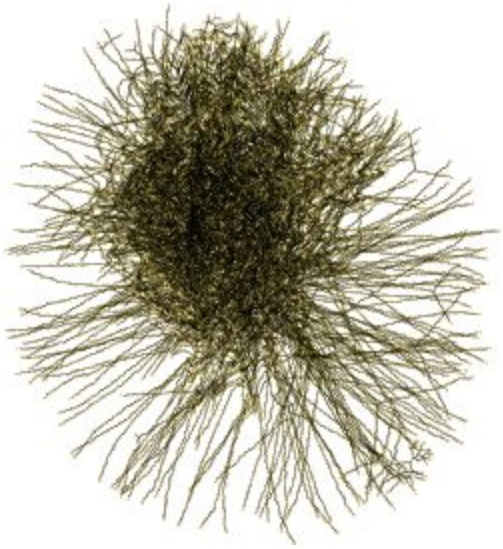
Рис 6. Неодназначность выбора пути в графе де Брейна в случае наличия повторов

В таких случаях программы-сборщики разделяют собираюмую последовательность на отдельные однозначно собирающиеся части, называемые контигами (contig). "Опасность" представляют не любые повтора, а только павторы, которые длиннее чем выбранный размер k. Если повтор длиннее, чем k, но короче чем риды, то можно использовать информацию о том, из каких ридов пришли k-меры, чтобы определить правильный путь. Если же повтор длиннее, чем рид, то ситуация алгоритмически становится неразрешимой.
Отметим, что таким же образом обстоят дела и в случае OLC-сборщиков: повторы, длина которых превышает длину ридов, выделяются в отельные контиги, также отдельными контигами становятся фрагменты генома между повторами.

Синопсис

- Геном целиком прочесть нельзя, только его короткие фрагменты, взаимное расположение которых не известно
- Для сборки этих фрагментов в исходную последовательность генома применяются алгоритмы основаные на графах.
- Эйлеров путь на графе - посещаем все ребра, ни одно ребро не посещая дважды. Гамильтонов путь - все вершины, ни одну вершину не посещая дважды.
- Задача поиска эйлерова пути решается просто, гамильтонова - сложно.
- Есть два основных метода сборки: OLC и на графах де Брёйна.
- OLC подход подразумевает поиск гамильтонова пути в графе, вершинами которого являются риды. Вычислительно затратен.
- В подход на графах де Брейна мы работаем не с исходными ридами, но с k-мерами, которые в них представлены (k = 20..60). На основе k-меров составляется граф, в котором необходимо найти эйлеров путь. Такой метод вычислительно проще.
- Повторы в геноме, которые длиннее среднего размера рида, приводят к неодназначностям при сборке генома.
- Вместо того, чтобы выбрать случайный вариант, программы-сборщики выдают отдельные однозначно собираемые фрагменты, называемые контигами.

P. S. Попытки визуализировать графы, получающиеся при сборки реальных данных с секвенаторов приводят к чему-то подобному:



В иллюстрациях использованы материалы из:
. Genome Reconstruction: A Puzzle with a Billion Pieces. Phillip E. C. Compeau and Pavel A. Pevzner

. How to apply de Bruijn graphs to genome assembly. Phillip E C Compeau, Pavel A Pevzner & Glenn Tesler. Computational Biology
. http://ifmo.ru
. https://github.com/redayounsi/2kplus2