Сборка. Часть первая

Алгоритмы в биоинформатике

Антон Елисеев
eliseevantoncoon@gmail.com

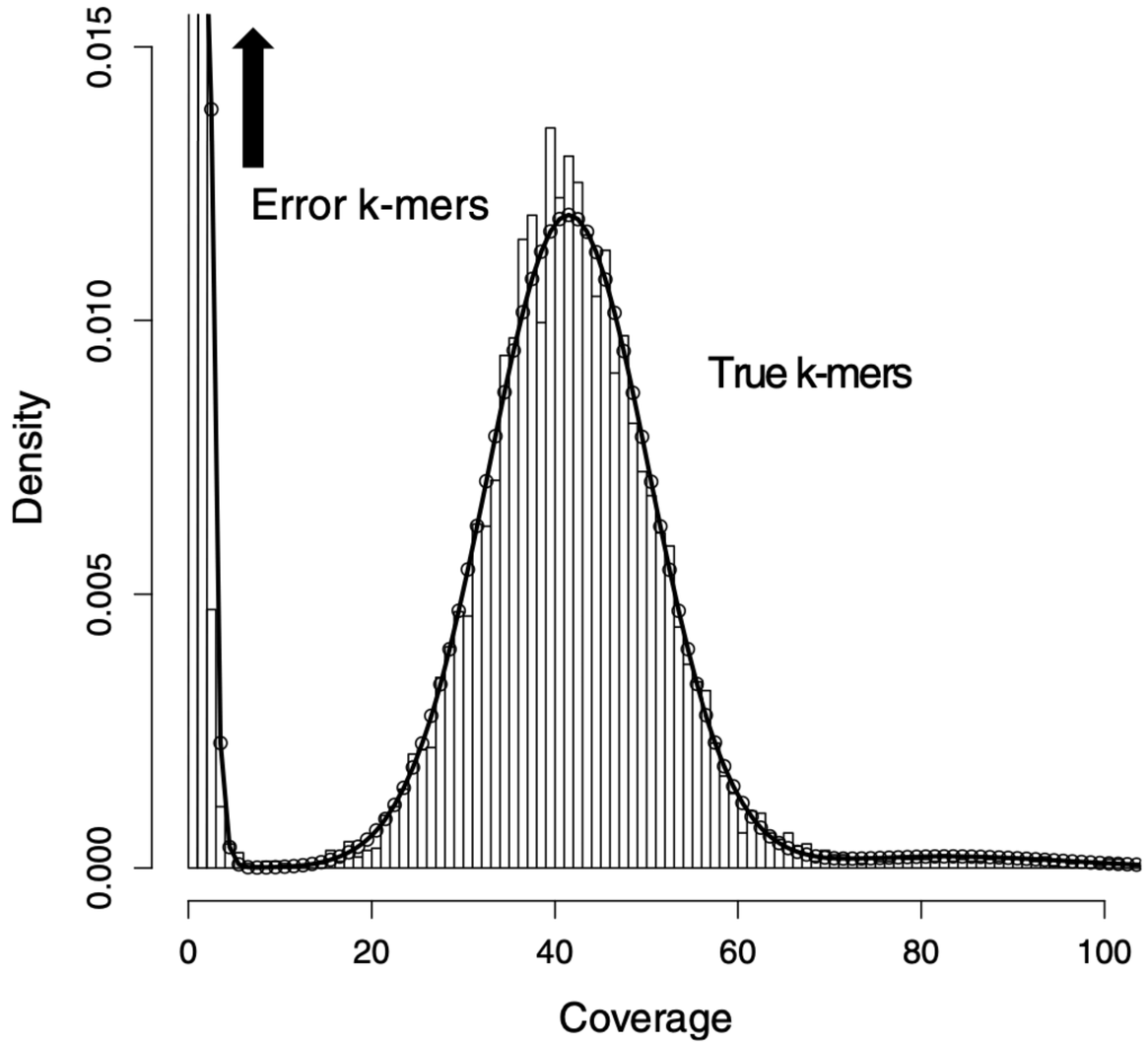
В прошлой лекции

- Секвенирование — случайный процесс
- Случаются ошибки секвенирования
- Риды с ошибками можно отбрасывать либо исправлять

В этой лекции

- Подсчет k-меров, фильтр Блума
- Задача сборки генома
- Задача SCS
- Жадное решение и overlap graph

Подсчет k-меров



Подсчет k-меров

Сколько существует всего k-меров над алфавитом {A, T, G, C}?

Подсчет k-меров

Сколько существует всего k-меров над алфавитом {A, T, G, C}?

[# all k-mers] $= 4^k$

Сколько разных k-меров может быть в строке длинны L?

Подсчет k-меров

Сколько существует всего k-меров над алфавитом {A, T, G, C}?

[# all k-mers] $= 4^k$

Сколько разных k-меров может быть в строке длинны L?

$N_{k,L} = L - k$

Подсчет k-меров

Сколько существует всего k-меров над алфавитом {A, T, G, C}?

[# all k-mers] $= 4^k$

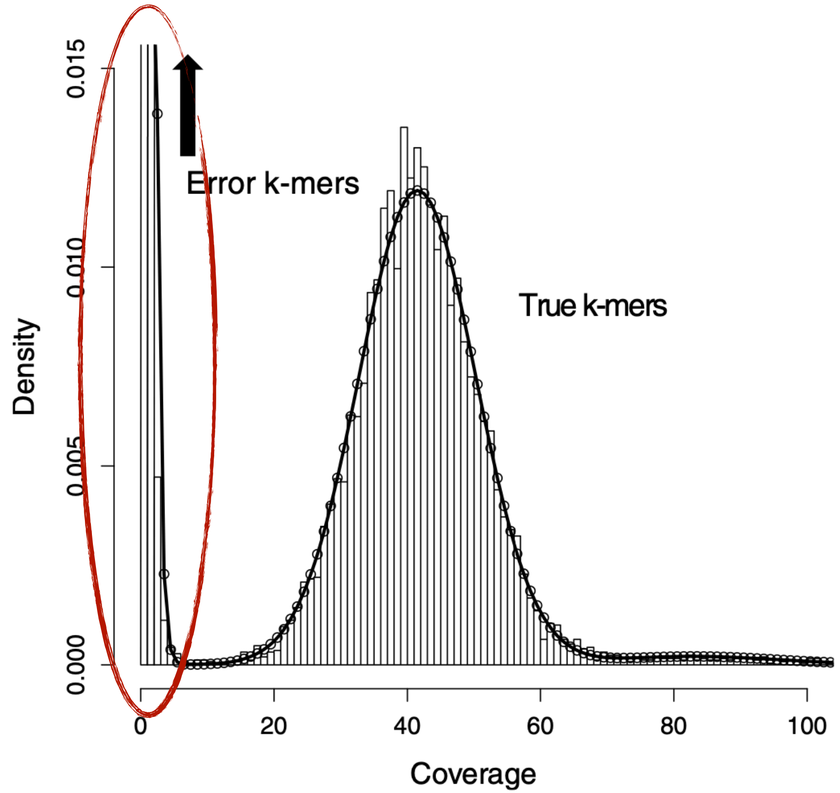
Сколько разных k-меров может быть в строке длинны L?

$N_{k,L} = L - k$

Чтобы их хранить в хеш-таблице необходимо $O(k(k - L))$ памяти

Подсчет k-меров

Основная проблема — ошибки секвенирования



Фильтр Блума

Рецепт :)

Взять массив B из m бит и добавить k независимых хеш-функций h_i : element $\rightarrow [1,m)$

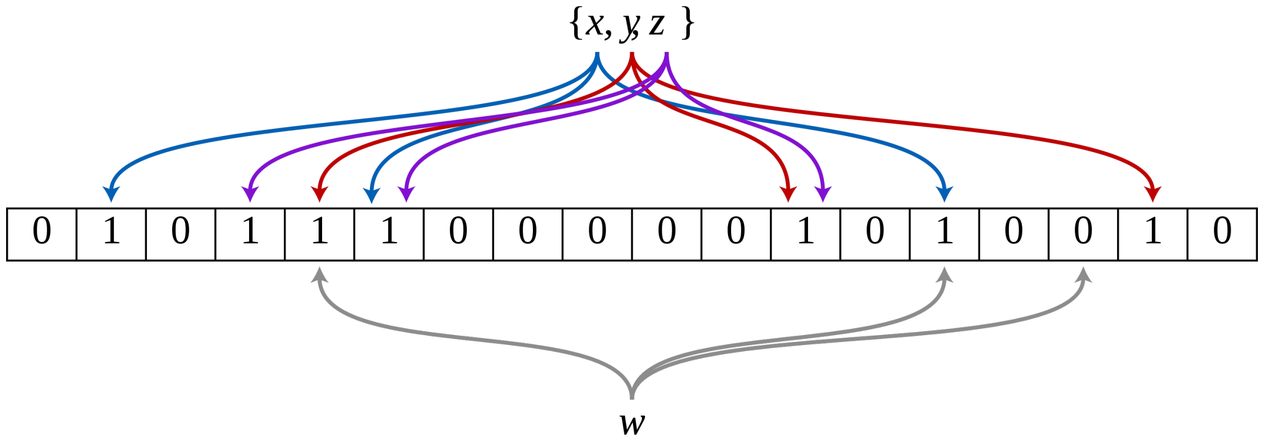
Операции:

- Добавить элемент
- Проверить наличие

Фильтр Блума: добавить элемент

Собираемся добавить элемент e

- Посчитаем $h_1(e), \ldots \ h_k(e)$
- Создадим битовый вектор $A(e)$ с единицами в позициях $h_1(e), \ldots \ h_k(e)$
- $B := B \vee A(e)$



Фильтр Блума: проверить элемент

Чтобы проверить есть ли в фильтре элемент e

- Посчитаем вектор $A(e)$
- Проверим $A(e) = B \wedge A(e)$
  - $True$: возможно e присутствует
  - $False$: e еще точно не встречался

Какая есть проблема?

Фильтр Блума: False positives

Фильтр Блума с вероятностью ошибки 0.01 и оптимально подобранным k требует всего 9.6 бит на 1 элемент и это не зависит от размера самого элемента.

Фильтр Блума: False positives

- k — число хеш-функций
- m — длина битового вектора
- n — число добавленных элементов
- p — вероятность FP

Фильтр Блума: False positives

- k — число хеш-функций
- m — длина битового вектора
- n — число добавленных элементов
- p — вероятность FP
- h_i — “хорошие” и независимые

$$\Pr(h_i(x) = p) = \frac{1}{m}, \quad p = 1 \ldots m$$

Фильтр Блума: False positives

При вставке первого элемента e

- Вероятность что бит B_j останется 0: $P_0(B_j = 0) = \left(1 - \frac{1}{m}\right)^k$
- То же самое после n вставок $P_n(B_j = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{\frac{-kn}{m}}$
- FP — когда для элемента y не равного ни одному из вставленных все $h_i(y)$ позиции равны 1: $P_{FP} = (1 - e^{\frac{-kn}{m}})^k$

Фильтр Блума: False positives

Оптимальное число хеш-функций $k = \frac{m}{n} \ln 2 \approx 0{,}6931 \frac{m}{n}$

Как следствие — чтобы фильтр Блума поддерживал заданную ограниченную вероятность FP, $|B| : O(n)$

Фильтр Блума в подсчете k-меров

”Efficient counting of k-mers in DNA sequences using a bloom filter” by P´all Melsted and Jonathan K Pritchard, 2011

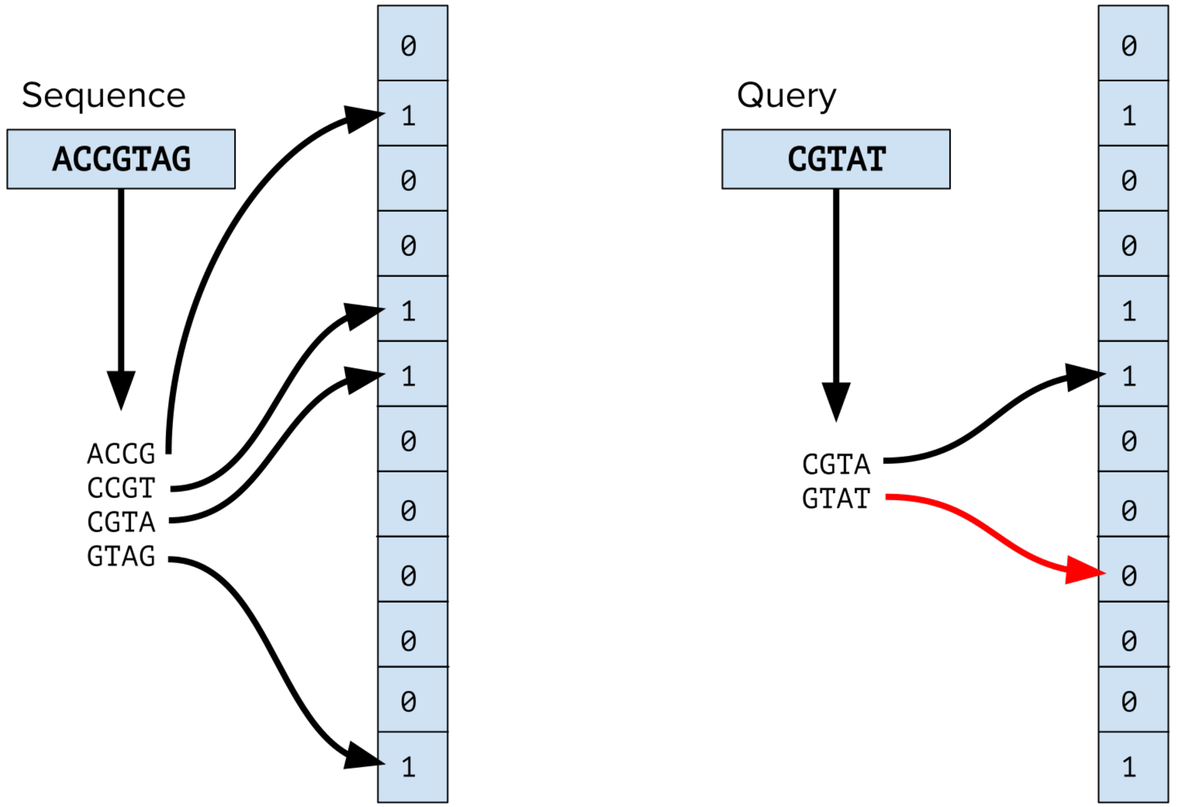
Задача: посчитать распределение k-меров

Алгоритм:

Создаем хеш-таблицу и фильтр Блума и дальше для всех k-меров проверяем, есть ли k-мер в фильтре?

- False: добавляем в фильтр
- True: либо вставляем в хэш-таблицу со значением 2, либо увеличиваем счетчик.

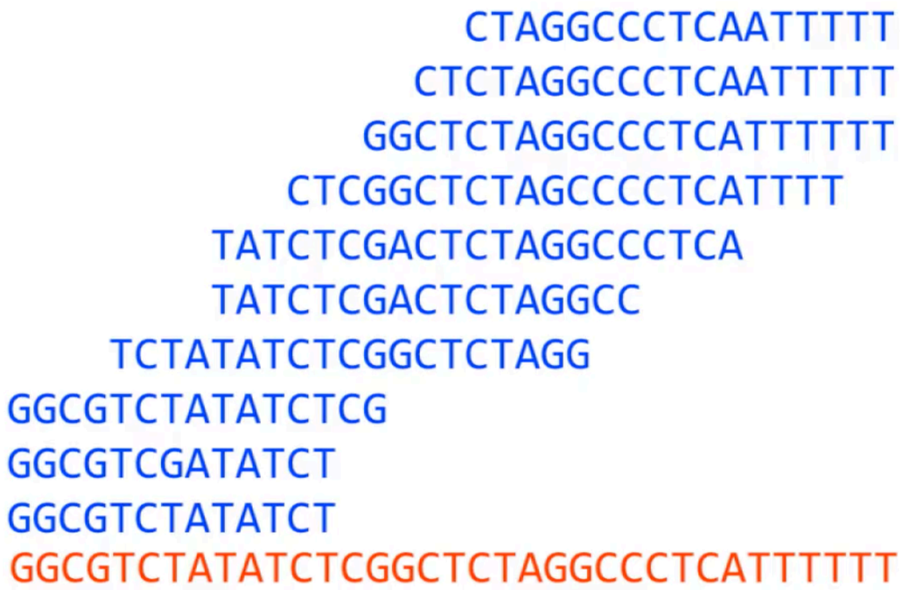
Фильтр Блума при поиске



Сборка геномов

Дано:
Множество ридов

Цель:
Найти геном, из которого эти риды получены



Сборка геномов

Дано:
Множество ридов

Цель:
Найти геном, из которого эти риды получены

Сборка геномов. SCS

Задача поиска SCS (shortest common supersequence) — для заданного множества подстрок, найти такую строку минимальной длинны, которая содержит все подстроки.

AAA, AAT, ATT, TTA, TTT

CS = AAAAATATTTTATTT

Но это не самая короткая строка

Сборка геномов. SCS

Задача поиска SCS (shortest common supersequence) — для заданного множества подстрок, найти такую строку минимальной длинны, которая содержит все подстроки.

AAA, AAT, ATT, TTA, TTT

CS = AAAAATATTTTATTT

Но это не самая короткая строка

CSC = AAATTTA

SCS

SCS (shortest common supersequence) — очень трудная задача.
NP-трудная!

Как решить?

SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1,2,3,4,5) → AAATTATTT

(1,5,2,4,3) → AAATTTAATTATT

SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1,2,3,4,5) → AAATTATTT

(1,5,2,4,3) → AAATTTAATTATT

(1,2,3,5,4) → AAATTTA

SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1,2,3,4,5) → AAATTATTT

(1,5,2,4,3) → AAATTTAATTATT

(1,2,3,5,4) → AAATTTA

SCS

AAA, AAT, ATT, TTA, TTT

Рассмотрим разные порядки

(1,2,3,4,5) → AAATTATTT

(1,5,2,4,3) → AAATTTAATTATT

(1,2,3,5,4) → AAATTTA

SCS

Алгоритм:

- Перебираем все возможные порядки
- Для каждого порядка объединяем склеиваем строки, начиная с первого не перекрывающегося символа в строке

SCS

Алгоритм:

- Перебираем все возможные порядки
- Для каждого порядка объединяем склеиваем строки, начиная с первого не перекрывающегося символа в строке

Замечания:

Сложность такого алгоритма $O(n!nl^2)$

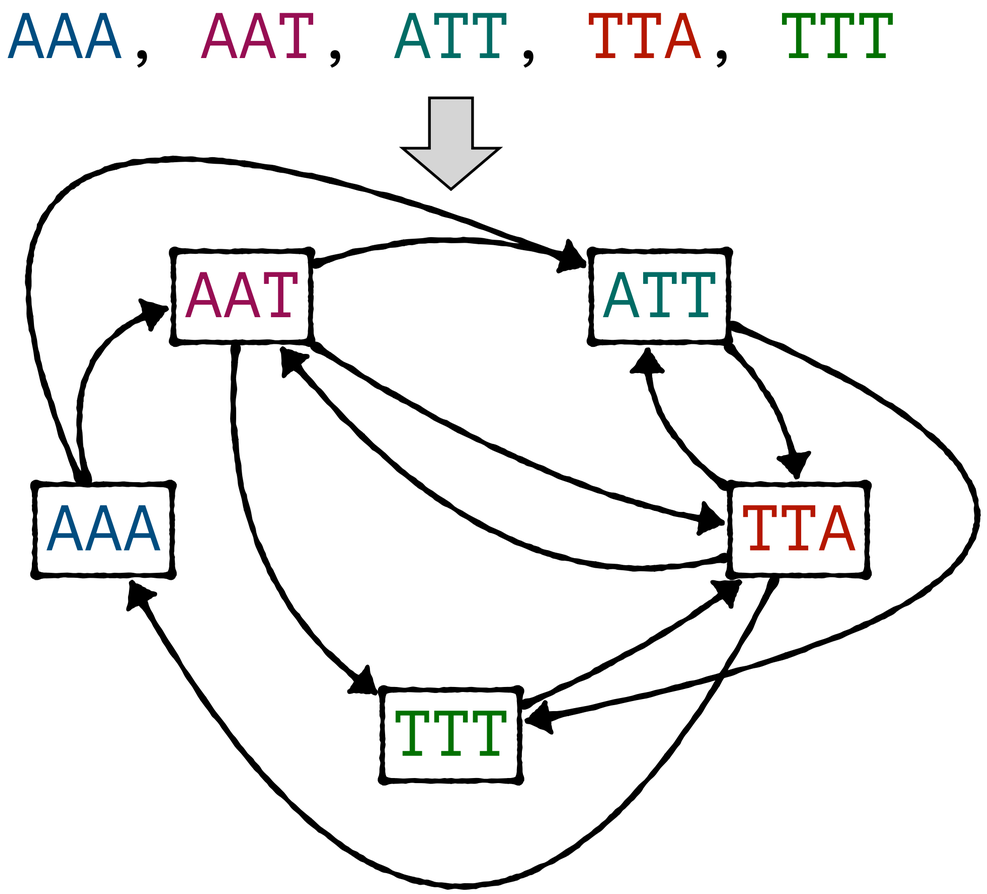
Если подстроки разной длины то нужно еще проверять, есть ли уже подстрока в строке, иначе контрпример {ATA, T}

SCS. Граф перекрытий

Определение: для заданного множества строк $\{S\}$ и порога t, графом перекрытий $O(\{S\}, t)$ будем называть такой направленный граф, в котором

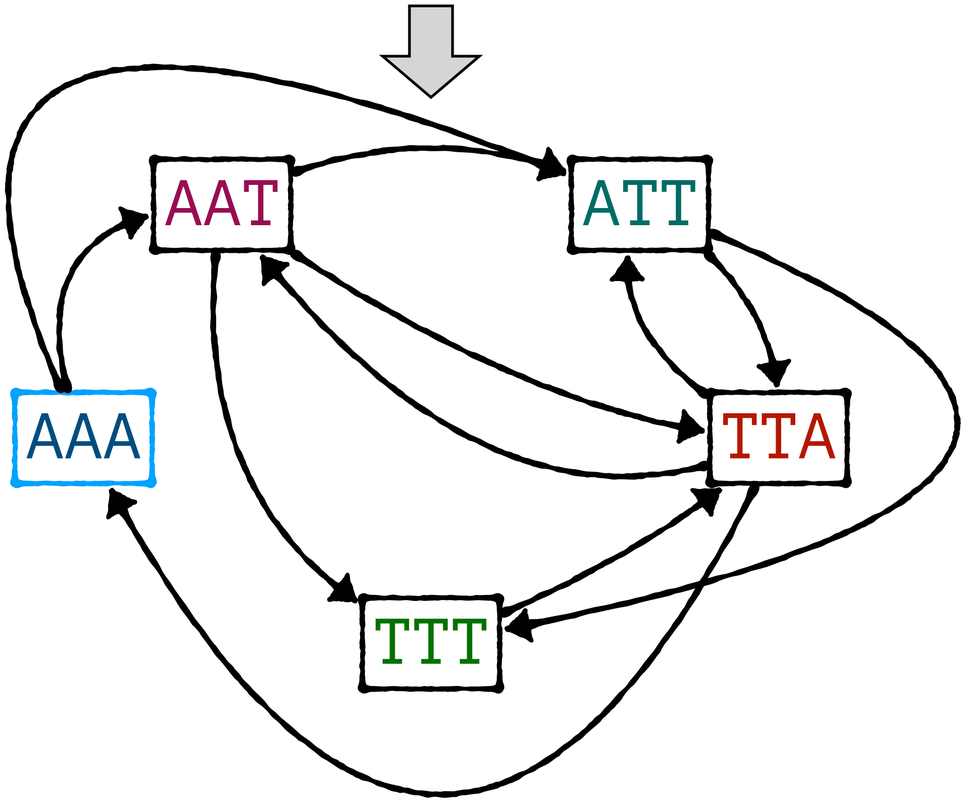
- Вершины соответствуют строкам S_i
- Ребро проводится из вершины i в вершину j если самый длинный суффикс S_i совпадающий с префиксом S_j длиннее чем t
- Вес ребра $e : (i, j)$ — размер суффикса

SCS. Граф перекрытий



SCS. Обход графа перекрытий

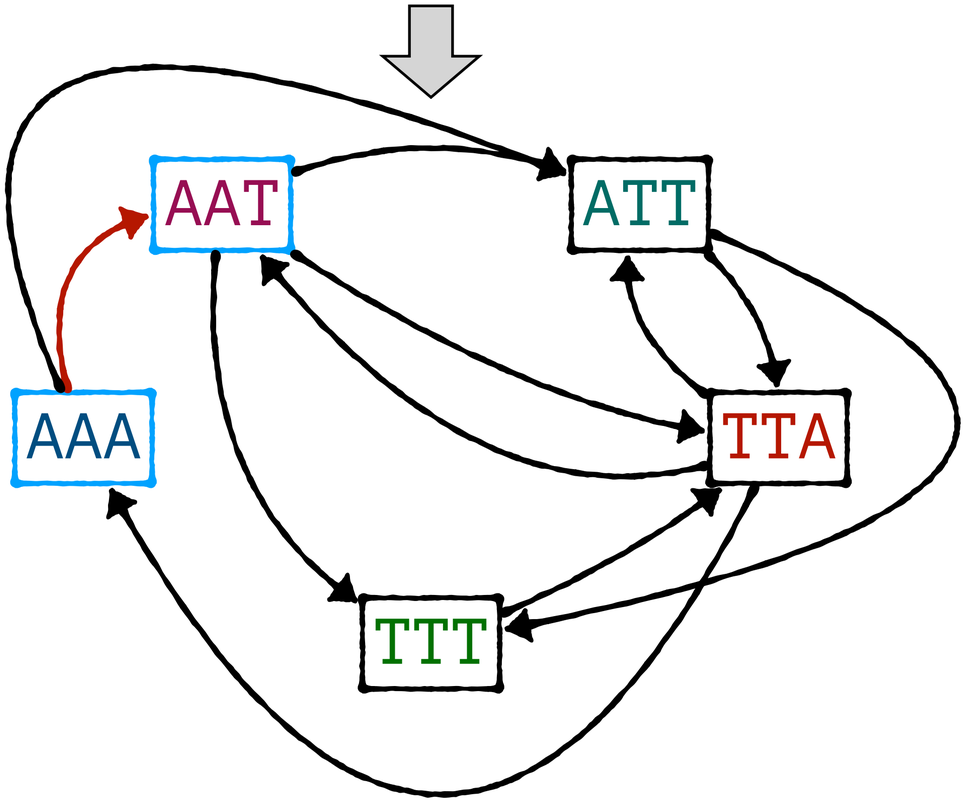
AAA, AAT, ATT, TTA, TTT



AAA

SCS. Обход графа перекрытий

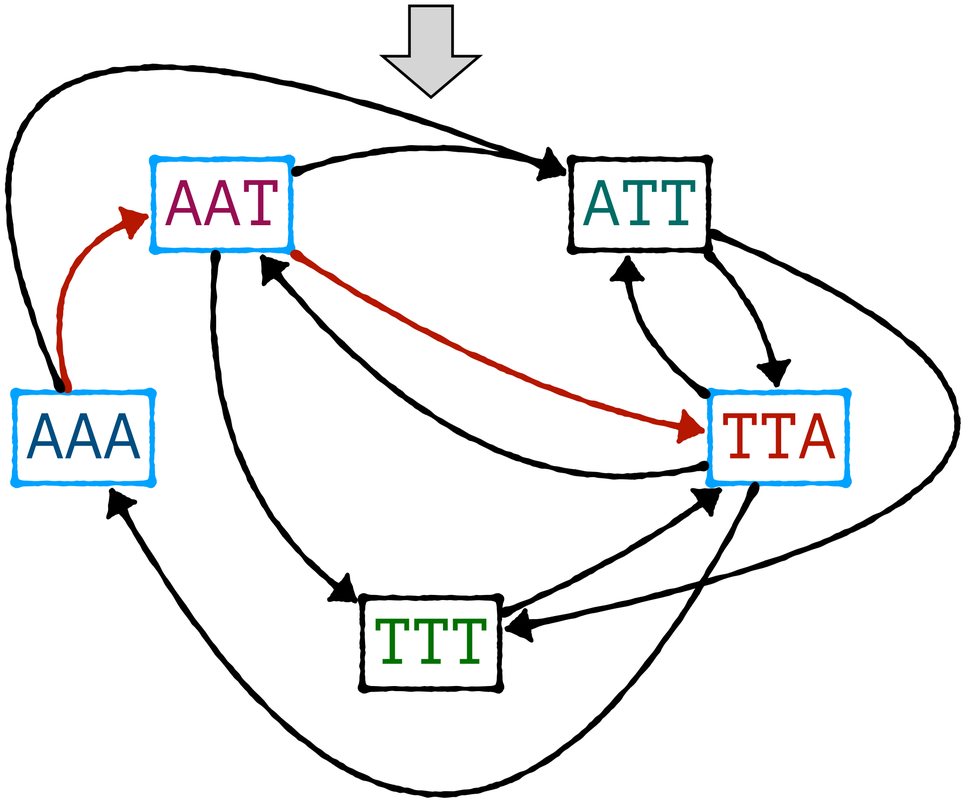
AAA, AAT, ATT, TTA, TTT



AAAT

SCS. Обход графа перекрытий

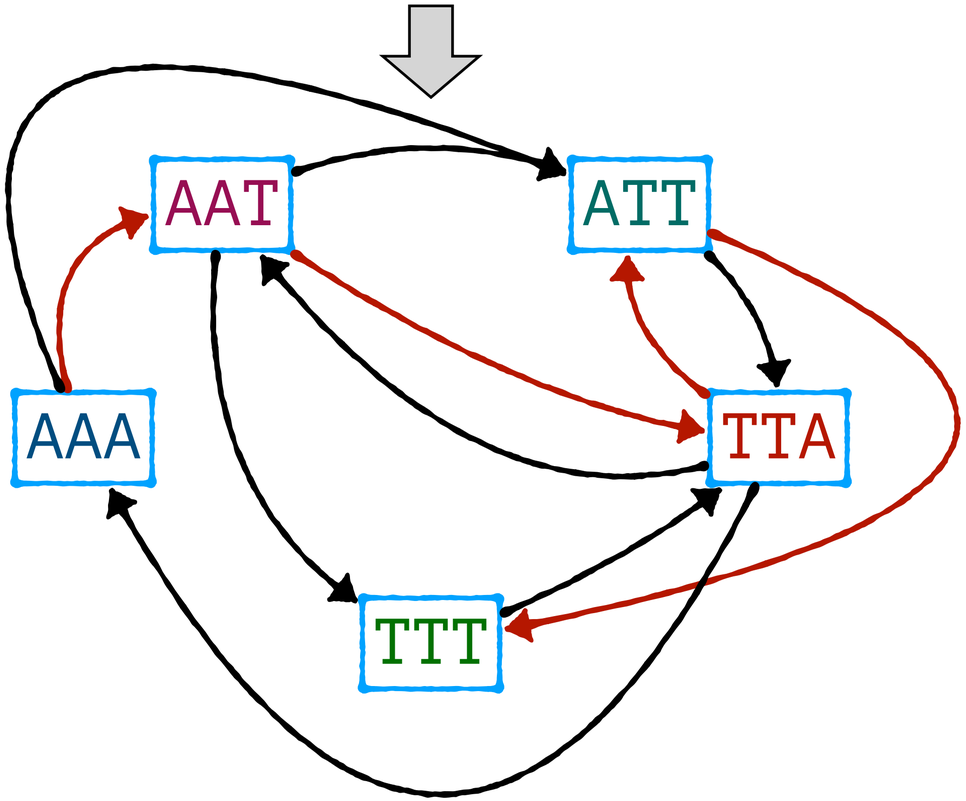
AAA, AAT, ATT, TTA, TTT



AAATTA

SCS. Обход графа перекрытий

AAA, AAT, ATT, TTA, TTT



AAATTATTT

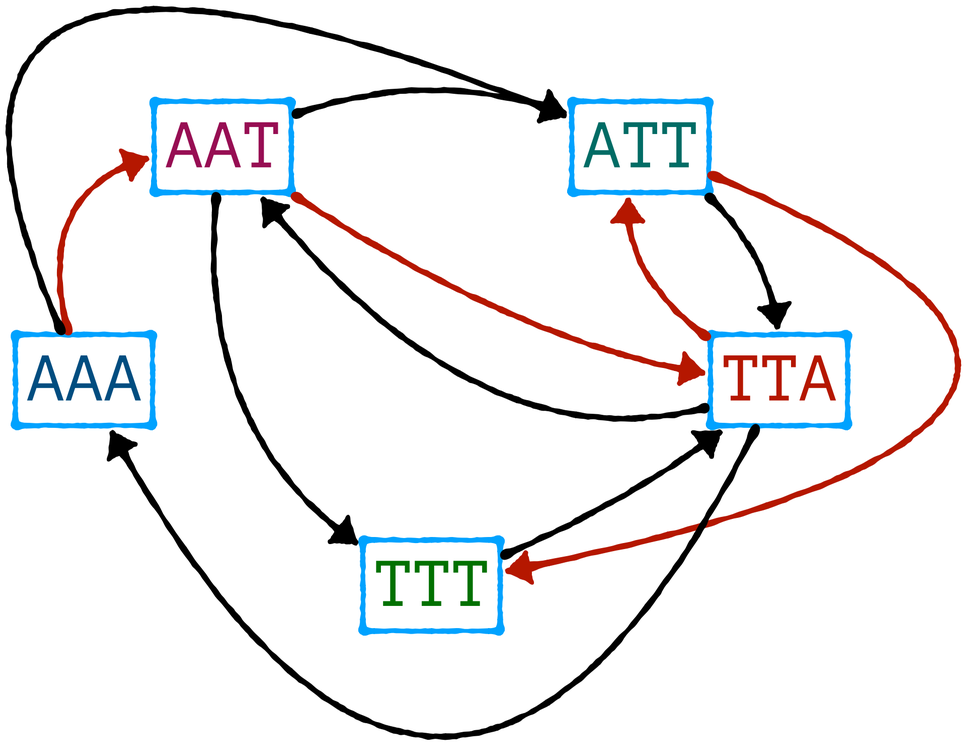
SCS. Обход графа перекрытий

Алгоритм:

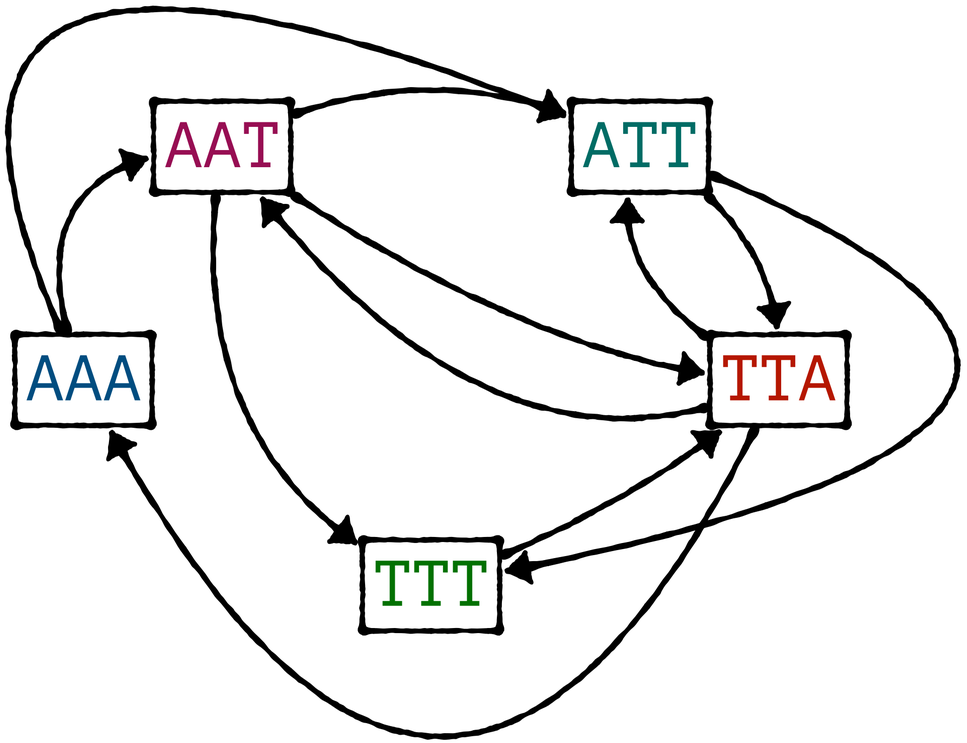
- Найдем Гамильтонов путь в графе перекрытий

Замечания:

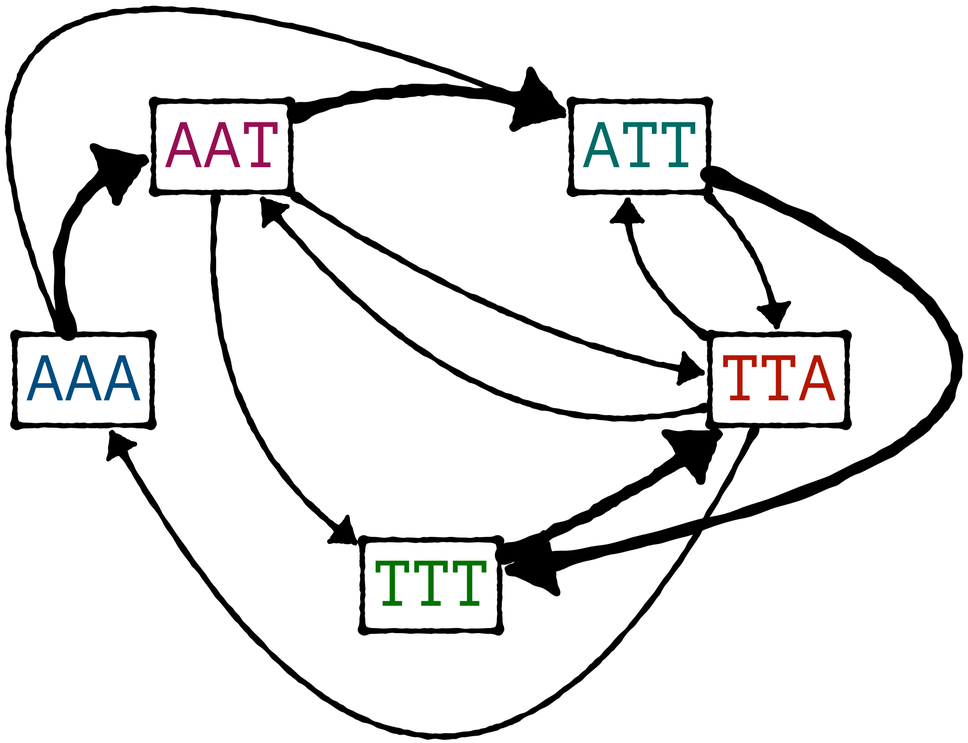
- Находит не кратчайшую суперстроку
- Все еще трудная задача
- Пути может и не существовать



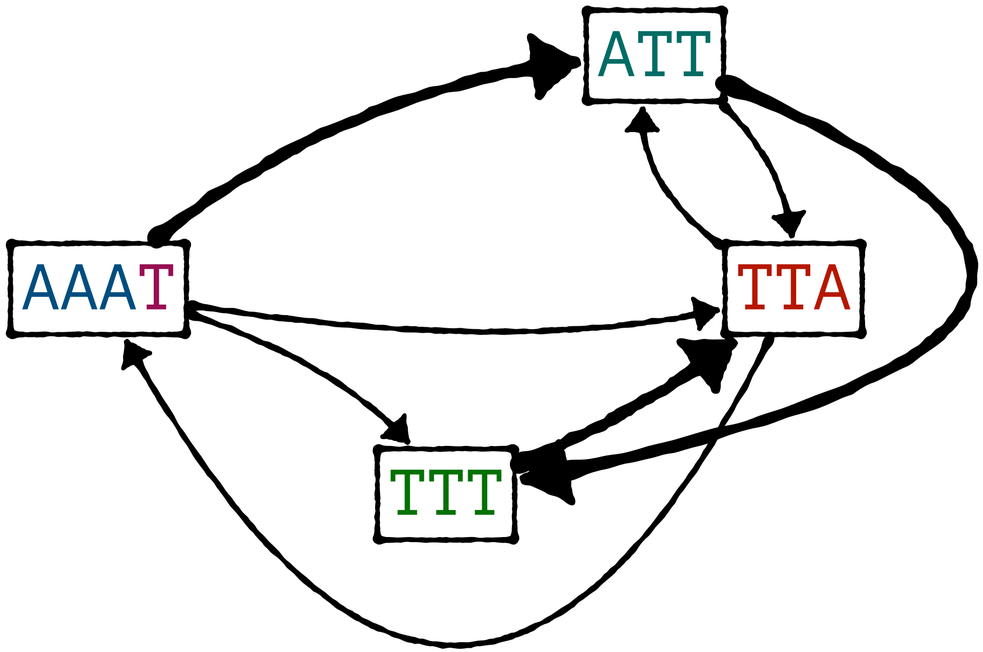
SCS. Жадный алгоритм



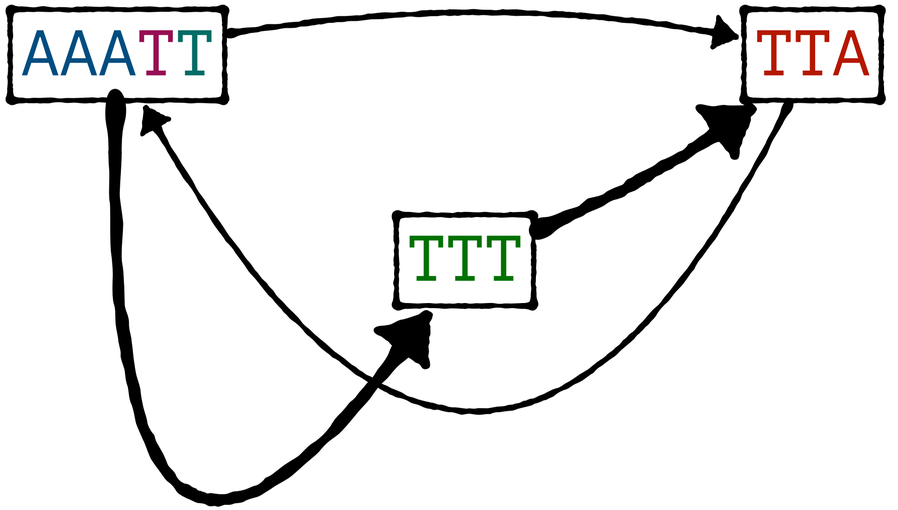
SCS. Жадный алгоритм



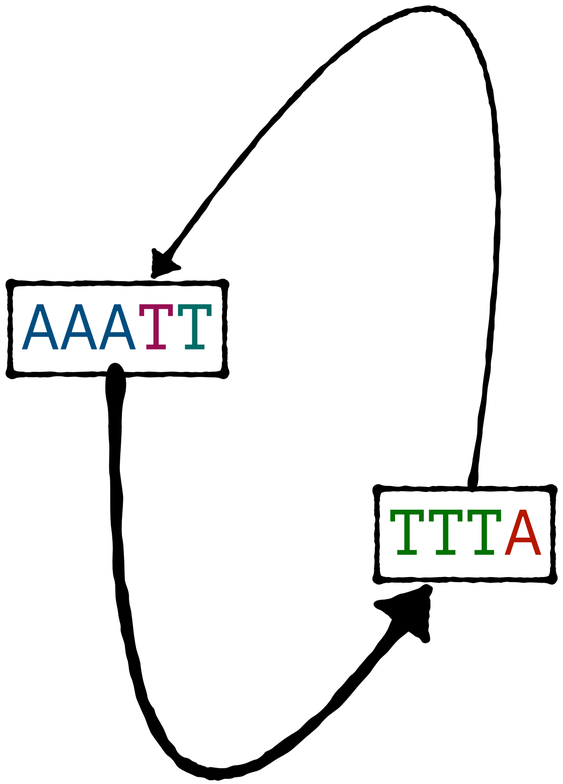
SCS. Жадный алгоритм



SCS. Жадный алгоритм



SCS. Жадный алгоритм



SCS. Жадный алгоритм

AAATTTA

SCS. Жадный алгоритм

Алгоритм:

- Строим граф перекрытий
- Пока есть вершины объединяем те, что соединены тяжелыми ребрами

Замечания:

- Находит не кратчайшую суперстроку
- Если ребер нет, а вершины еще остались то просто конкатенируем их
- Задача полиномиальная

Резюмируем

- При анализе ридов нужно экономить память и на помощь приходят таки структуры данный как фильтр Блума
- Задача SCS — абстрактное приближение задачи сборки генома является сложной задачей
- overlap graph позволяет реализовать жадное решение задачи поиска SCS