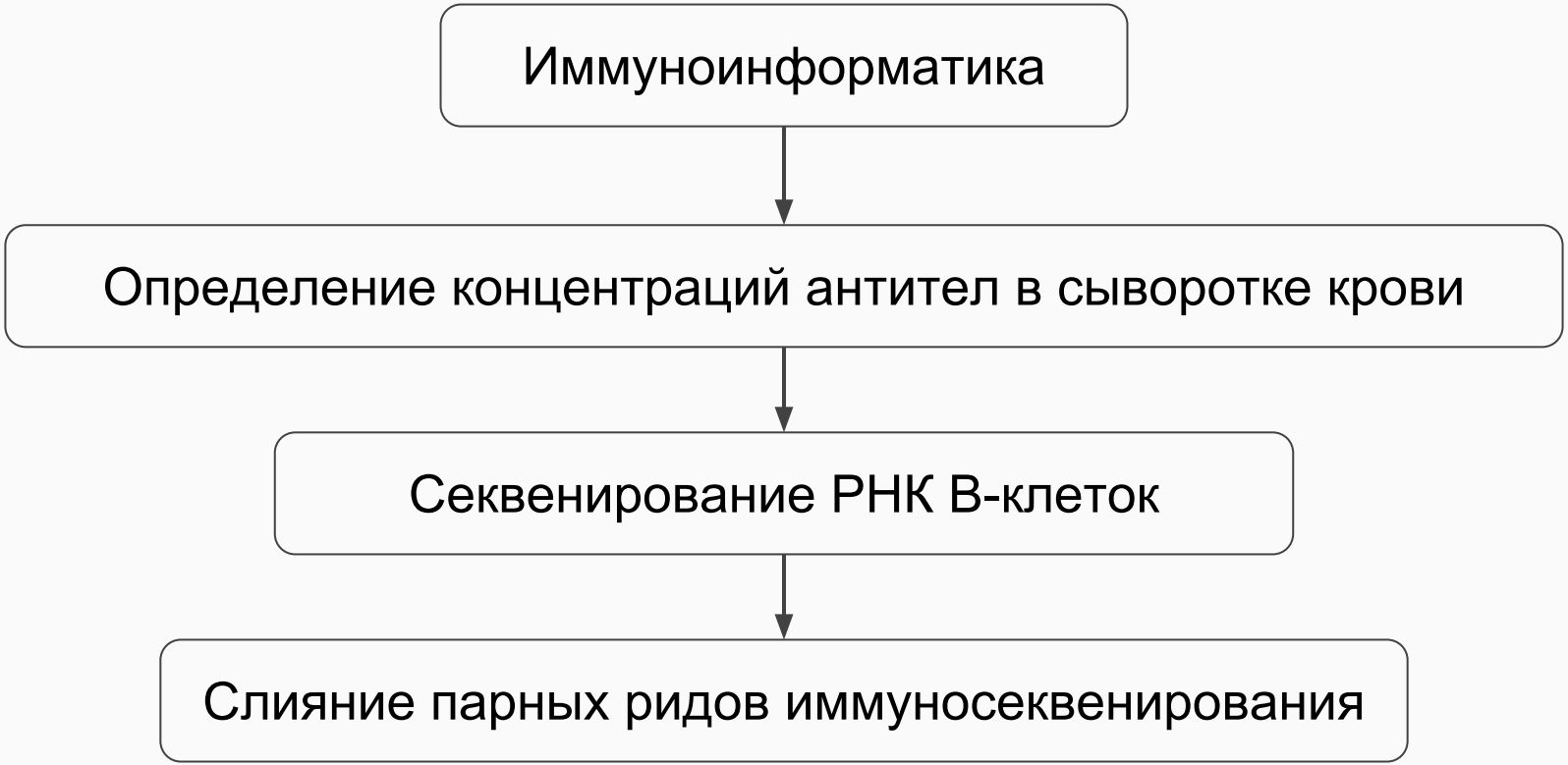
Слияние парных ридов иммуносеквенирования

Автор: Егор Богомолов
Научный руководитель: Александр Шлемов

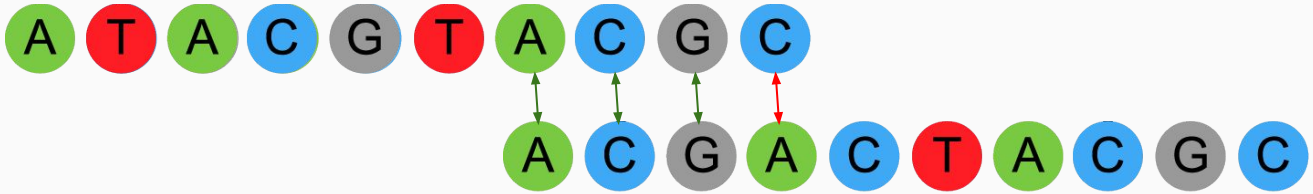
Откуда взялась задача?



Задача в теории:

Для пары строк над алфавитом {A, C, T, G, N} находить такие суффикс первой и префикс второй, что:

1. Они имеют равную длину;
2. Доля попарно несовпадающих символов не больше заданной и их количество минимально;
3. Длина не меньше заданной;



Задача на практике:

В уже имеющемся проекте написать программу, “цивилизованно” решающую вышеописанную задачу, то есть:

1. Разбор командной строки с помощью библиотеки boost::program_options.
2. Работа со строками, их чтение и запись с помощью библиотеки seqan.
3. Иметь возможность без добавления лишнего кода менять алгоритм поиска перекрытия.
4. Иметь реализованный наивный алгоритм.
5. Попробовать различные более умные подходы к решению, сравнить их и выбрать лучший.
6. Распараллелить работу программы с помощью OpenMP.

I этап - внешняя оболочка:

1. Документация boost.
2. Документация seqan.
3. Описание формата fastq.
4. Имеющийся код проекта.

→

1. Класс для работы с ридами.
2. Класс для объединения ридов при найденном перекрытии.
3. Парсер командной строки.
4. Адекватные стандартные настройки объединения и возможность их изменения пользователем.

II этап - наивный алгоритм:

Что делает?

- Перебирает все возможные сдвиги второй строки, проходом считает расстояние Хемминга.

Как работает в теории?

- $O(n^2)$

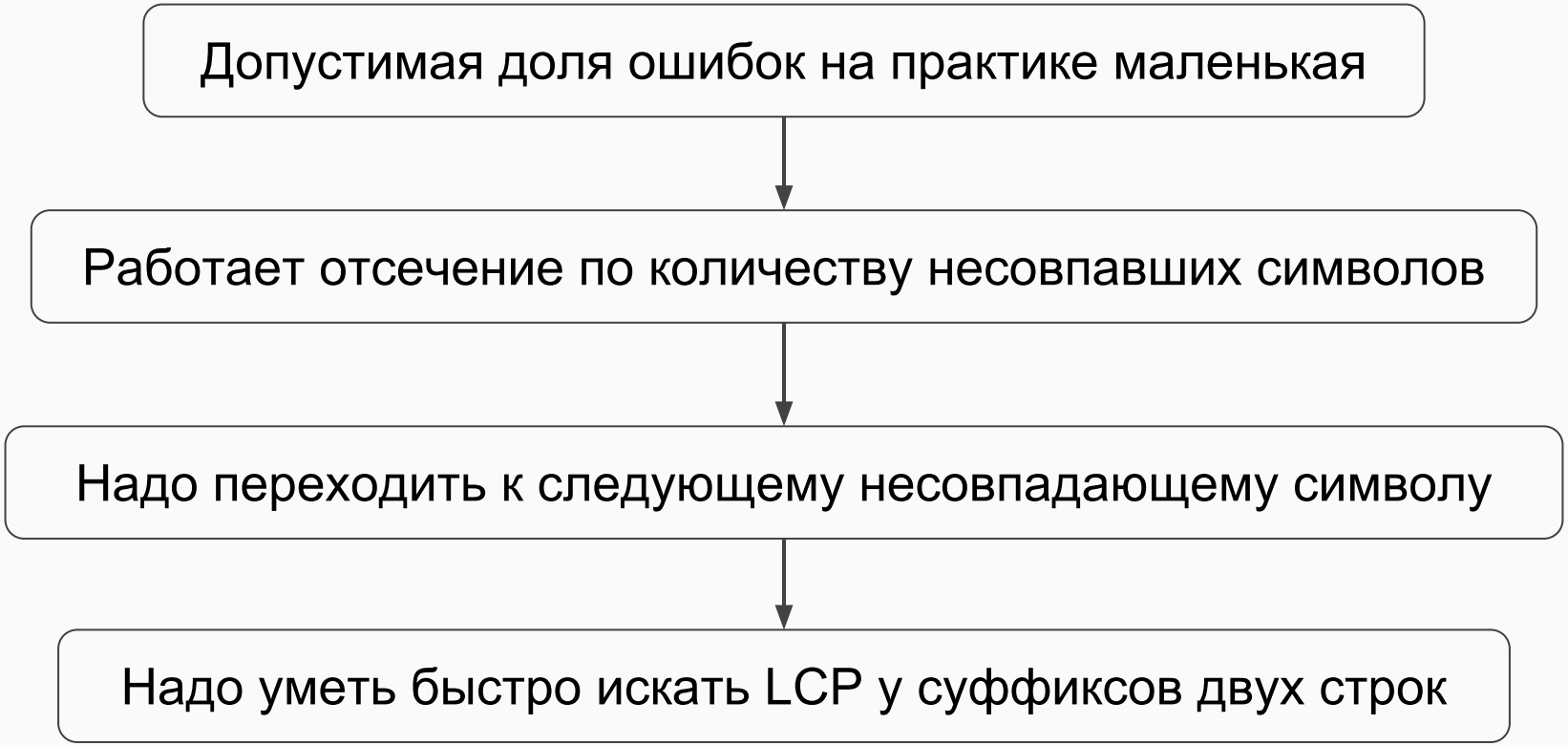
Как работает на практике?

- Медленно, но большего сложно ожидать.

Зачем нужен?

- Работает не настолько медленно, чтобы его нельзя было использовать.
- С его помощью можно тестировать более умные подходы.

III этап - хороший в теории алгоритм:



III этап - хороший в теории алгоритм:

LCP за O(1)

=

suffix array + LCP array + sparse table

III этап - хороший в теории алгоритм:

Что делает?

- Считая расстояние, каждый раз переходит к следующему несовпадающему символу за O(1).

Как работает в теории?

- $O(rate \cdot n^2 + n \cdot \log(n))$, rate - допустимая доля несовпавших.

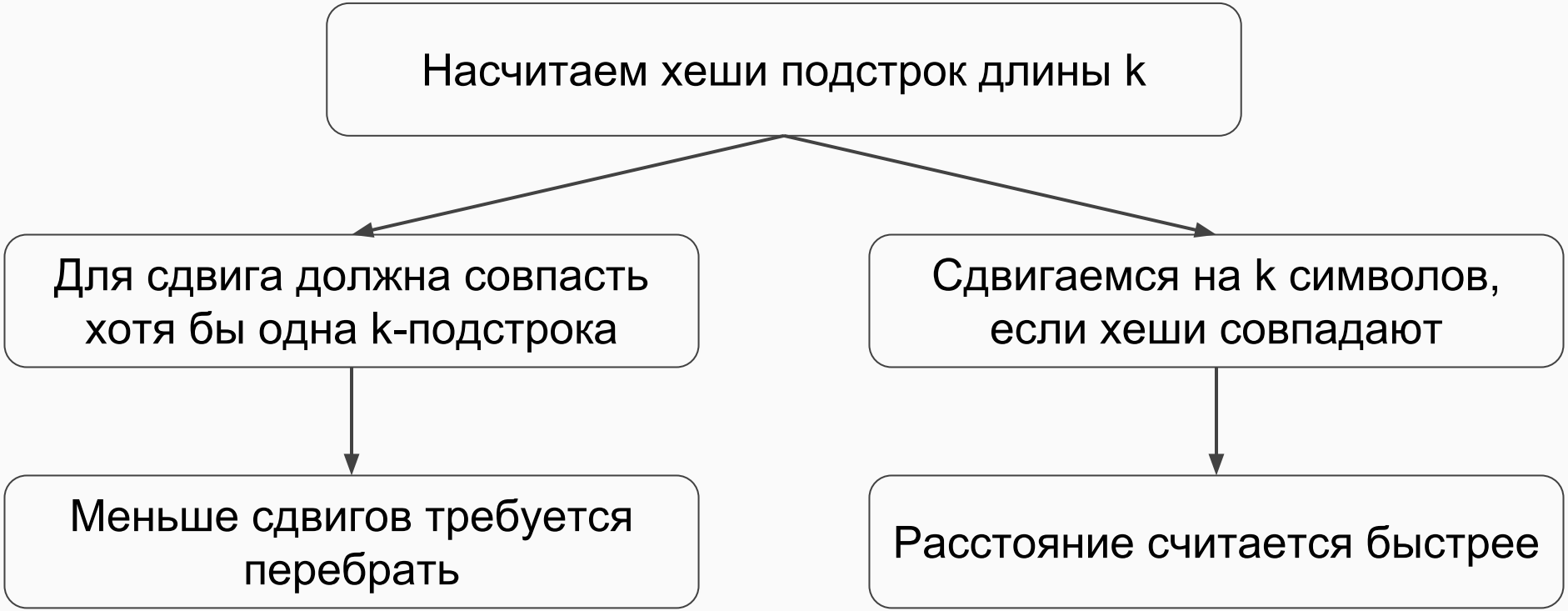
Как работает на практике?

- Из-за большой константы при построении всего в 2-2.5 раза быстрее наивного алгоритма.

Зачем нужен?

- Из теории казалось, что будет работать лучше.
- При уменьшении rate ускоряется до выигрыша в 4 раза.

IV этап - хороший на практике алгоритм:



IV этап - хороший на практике алгоритм:

Что делает?

- Отсекает часть сдвигов при помощи хешей. При подсчете расстояния может сдвигаться больше, чем на один символ.

Как работает в теории?

- $O(n^2)$, возможно, можно оценить данные эвристики, но этого не требовалось.

Как работает на практике?

- Работает уже в 10-11 раз быстрее наивного алгоритма. Как и предыдущий алгоритма, при уменьшении rate ускоряется.

Зачем нужен?

- Работает ощутимо быстрее предыдущих.

Выводы:

+ Написанной программой можно пользоваться, она встроена в проект.
+ Получен один из первых опытов написания кода, которым будут пользоваться и который будут читать какие-то люди помимо меня.
+ Проверены на практике разные подходы к решению.
+ Существенно улучшенно время работы.
- Программа не распараллеленна.
- Рассматривалась возможность некоторой исследовательской деятельности в конце проекта, но на нее не хватило времени.

Но где же все это найти?

Репозиторий с проектом находится по адресу:

https://github.com/egor-bogomolov/ig_repertoire_constructor

Спасибо за внимание!