

Zliczanie k-merów

Przypomnienie: co to są k-mery?

- Ciągłe bądź nieciągłe fragmenty danej sekwencji
- Ponadto wyróżniamy k-mery pozycyjne i niepozycyjne

A	C	T	A	A	C	G
A	C C	T T T	A A A	A A A	C C	G

A	C	T	A	A	C	G
A	C C	- T T	A - A A	A - A	C -	G

Problem: efektywne zliczanie k-merów w sekwencjach na podstawie danego alfabetu

$$\text{hash}(s_1 \dots s_n) = (\text{code}(s_1) \cdot P^{n-1} + \dots + \text{code}(s_{n-1}) \cdot P + \text{code}(a_n)) \mod M$$

- Rozwiązanie zaimplementowane w większości w C++11 (przy pomocy *Rcpp*)
- Zastosowane rozwiązanie opiera się na idei haszowania
- Zrównoleglenie algorytmu przy pomocy pakietu *RcppParallel*
- *Spostrzeżenie: k-mery pozycyjne możemy zliczać tak samo jak zwykłe k-mery:
Należy jedynie dodać wartość pozycji do funkcji haszującej*

Zastosowane ulepszenia w porównaniu z poprzednią wersją

- Rozdzielenie implementacji zliczania k-merów ciągłych od zliczania k-merów nieciągłych
- Dla k-merów ciągłych zastosowanie zliczania przy pomocy algorytmu *Rolling Window Hash*



Inne ulepszenia

- **Optymalizacja 1: nie przetwarzamy przedziałów, w których nie ma ani jednego k-meru zawierającego wszystkie symbole z danego alfabetu**

Inne ulepszenia

- Optymalizacja 1: nie przetwarzamy przedziałów, w których nie ma ani jednego k-meru zawierającego wszystkie symbole z danego alfabetu
- **Optymalizacja 2: zrównoleglone generowanie wszystkich k-merów**

Inne ulepszenia

- Optymalizacja 1: nie przetwarzamy przedziałów, w których nie ma ani jednego k-meru zawierającego wszystkie symbole z danego alfabetu
- Optymalizacja 2: zrównoleglone generowanie wszystkich k-merów
- **Refaktoring kodu 1: możliwość podania ciągów o elementach dowolnego typu (o ile podamy odpowiednie funkcje konwertujące)**

Inne ulepszenia

- Optymalizacja 1: nie przetwarzamy przedziałów, w których nie ma ani jednego k-meru zawierającego wszystkie symbole z danego alfabetu
- Optymalizacja 2: zrównoleglone generowanie wszystkich k-merów
- Refaktoring kodu 1: możliwość podania ciągów o elementach dowolnego typu (o ile podamy odpowiednie funkcje konwertujące)
- **Refaktoring kodu 2: zastosowanie standardu C++17**

Inne ulepszenia

- Optymalizacja 1: nie przetwarzamy przedziałów, w których nie ma ani jednego k-meru zawierającego wszystkie symbole z danego alfabetu
- Optymalizacja 2: zrównoleglone generowanie wszystkich k-merów
- Refaktoring kodu 1: możliwość podania ciągów o elementach dowolnego typu (o ile podamy odpowiednie funkcje konwertujące)
- Refaktoring kodu 2: zastosowanie standardu C++17
- **Refaktoring kodu 3: dla każdego etapu rozwiązania jest pomocnicza funkcja z wrapperami do R**
Dzięki temu: lepsze, jednostkowe przetestowanie kodu

To nie jest koniec moich przygód!

Kolejne zadania:

- Integracja z pakietem *tidysq*
Dzięki temu: optymalizacja pamięci
- Implementacja liniowej wersji algorytmu dla k-merów niespójnych
- ... *sky is the limit* ...

Dziękuję za uwagę ;)