#### Zliczanie k-merów

#### Przypomnienie: co to są k-mery?

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

$\mathbf{A}$	$\mathbf{C}$	$  \mathbf{T}  $	$\mathbf{A}$	$\mathbf{A}$	$\mathbf{C}$	$\mathbf{G}$
A	С	Т				
	С	$\mid T \mid$	A			
		$\mid T \mid$	A	A		
			A	A	$\mathbf{C}$	
				A	$\mathbf{C}$	G

$oldsymbol{A}$	$\mathbf{C}$	$oxed{\mathbf{T}}$	A	A	$\mathbf{C}$	$\mathbf{G}$
A	С	_	A			
	$\mathbf{C}$	$\mid T \mid$	_	A		
		$\mid T \mid$	A	_	С	
			A	A	_	G

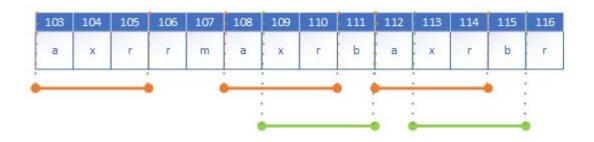
# Problem: efektywne zliczanie k-merów w sekwencjach na podstawie danego alfabetu

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

- 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



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

- 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

- 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)

- 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

- 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;)