tidysq: efektywne przetwarzanie sekwencji biologicznych w R
Dokumentacja interfejsów

22 stycznia 2021

Mateusz Bąkała, Dominik Rafacz

promotor: Michał Burdukiewicz

wersja 1.0.0

Spis treści

1 Wstęp

Niniejszy dokument zawiera opis wszystkich publicznych interfejsów udostępnianych przez pakiet. Spełniają one wszystkie wymagania funkcjonalne wyspecyfikowane w dokumentacji biznesowej. W pierwszej podsekcji przedstawiony jest interfejs R, w drugiej interfejs C++

2 Interfejs R

Wszystkie elementy tego interfejsu to funkcje wywoływane z poziomu konsoli R. Wyspecyfikowane są warunki na typy przyjmowanych parametrów oraz typy zwracanych parametrów oraz możliwe sytuacje wyjątkowe. Niewyszczególnione są błędy spowodowane podaniem parametrów o nieodpowiednich typach – zachodzą one z domysłu. Jeśli któraś funkcja odpowiada bezpośrednio danemu wymaganiu funkcjonalnemu, jest to podane w tabeli. Funkcje, dla których zaznaczono, że są metodami, mają określony kontrakt dla pierwszego argumentu jako obiektu klasy `sq`.

Funkcja `alphabet()`

Zwraca atrybut `alphabet` podanego obiektu.

Parametry:

- `x : [sq]` – obiekt, którego atrybut ma zostać wyciągnięty

Wyjście:

`[alfabet]` – alfabet stanowiący atrybut podanego obiektu

Metoda `as.character()`

Metody

1. ```
   as.character.sq(x, ...,
                   NA_letter = getOption("tidysq_NA_letter"))
   ```

2. ```
   as.character.sq_alphabet(x, ...)
   ```

Zwraca ciąg napisów stanowiący tekstową reprezentację obiektu
1) konwertuje każdą *Sekwencję* do napisu, sklejając bez separatora wszystkie *Litery* w oryginalnej kolejności; braki danych zastępowane są przez wskazaną literę
2) ekstrahuje wektor *Liter* z całego obiektu

Parametry wejściowe

- `x : [sq lub alfabet]` – obiekt, który ma ulec konwersji
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[wektor napisów]` – reprezentacja obiektu w postaci ciągu napisów

Metoda `as.matrix()`

Metody

1. ```
   as.matrix.sq(x, ...)
   ```

1) Zwraca macierz stanowiącą alternatywną reprezentację *Sekwencji*. Ekstrahuje każdą *Literę* do oddzielnej komórki, tworząc oddzielny wiersz dla każdej *Sekwencji*; krótsze sekwencje dopełnia brakami danych (`NA`).

Parametry wejściowe
   ```

- `x : [sq]` – obiekt, który ma ulec konwersji
- `... : [dowolne]` – dodatkowe argumenty do metod

Wyjście

`[macierz]` – reprezentacja obiektu w postaci macierzy

Funkcja generyczna `as.sq()`

Metody

1. `as.sq.character(x, ...)`
2. `as.sq.default(x, ...)`

Zwraca zaimportowany obiekt klasy `sq`
1) konstruuje obiekt przy użyciu funkcji `sq()`
2) importuje obiekt przy użyciu funkcji `import_sq()`

Parametry wejściowe

- `x : [sq]` – obiekt, który ma ulec konwersji
- `... : [dowolne]` – dodatkowe argumenty do metod

Wyjście

`[sq lub tibble]` – zaimportowany obiekt

Funkcja generyczna `bite()`

Metody

1. ```
   bite.sq(x, indices, ...,
           NA_letter = getOption("tidysq_NA_letter"),
           on_warning = getOption("tidysq_on_warning"))
   ```

1) Ekstrahuje zdefiniowany podciąg elementów z wszystkich *Sekwencji*.

Parametry wejściowe

- `x : [sq]` – obiekt z oryginalnymi *Sekwencjami*
- `indices : [wektor liczb całkowitych]` – indeksy elementów do wyciągnięcia z *Sekwencji* bądź — przeciwnie — do pominięcia przy wyciąganiu
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych
- `on_warning : [napis]` – specyfikacja zachowania w razie wystąpienia wyjątku

Wyjście

`[sq]` – obiekt z *Sekwencjami* stanowiącymi podciągi elementów oryginalnych *Sekwencji* określone poprzez wektor indeksów

Możliwe sytuacje wyjątkowe

- podanie indeksu o wartości większej niż długość sekwencji

Odpowiadające wymaganie

Zmiana kolejności ciągu *Liter*
   ```

Funkcja generyczna `complement()`

Metody

1. `complement.sq_dna_bsc(x, ..., NA_letter = getOption("tidysq_NA_letter"))`
2. `complement.sq_dna_ext(x, ..., NA_letter = getOption("tidysq_NA_letter"))`
3. `complement.sq_rna_bsc(x, ..., NA_letter = getOption("tidysq_NA_letter"))`
4. `complement.sq_rna_ext(x, ..., NA_letter = getOption("tidysq_NA_letter"))`

1–4) Tworzy komplementarne nici DNA/RNA.

Parametry wejściowe

- `x : [sq]` – obiekt z oryginalnymi *Sekwencjami* typu DNA bądź RNA
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[sq]` – obiekt tego samego *Typu*, co *Sekwencje* wejściowe

Odpowiadające wymaganie

Uzyskanie komplementarnych *Sekwencji* DNA/RNA

Funkcja generyczna `export_sq()`

Metody

1. `export_sq.sq_ami_bsc(x, export_format, name = NULL, ...)`
2. `export_sq.sq_ami_ext(x, export_format, name = NULL, ...)`
3. `export_sq.sq_dna_bsc(x, export_format, name = NULL, ...)`
4. `export_sq.sq_dna_ext(x, export_format, name = NULL, ...)`
5. `export_sq.sq_rna_bsc(x, export_format, name = NULL, ...)`
6. `export_sq.sq_rna_ext(x, export_format, name = NULL, ...)`

1–2) Eksportuje *Sekwencje* do jednego z następujących formatów: `ape::AAbin`, `Biostrings::AAString`, `Biostrings::AAStringSet`, `seqinr::SeqFastaAA`.
3–4) Eksportuje *Sekwencje* do jednego z następujących formatów: `ape::DNAbin`, `Biostrings::DNAString`, `Biostrings::DNAStringSet`, `seqinr::SeqFastadna`.
5–6) Eksportuje *Sekwencje* do jednego z następujących formatów: `Biostrings::RNAString`, `Biostrings::RNAStringSet`.

Parametry wejściowe

- `x : [sq]` – obiekt z *Sekwencjami* zakodowanymi w formacie udostępnianym przez *Pakiet*
- `export_format : [napis]` – specyfikacja docelowego formatu w postaci `pakiet::klasa`
- `name : [wektor napisów]` – nazwy sekwencji

- `...` : `[dowolne]` – dodatkowe argumenty do metod

Wyjście

`[dowolne]` – obiekt klasy specyfikowanej poprzez format eksportu

Odpowiadające wymaganie

Eksport obiektów do innych pakietów

Funkcja generyczna `find_invalid_letters()`

Metody

1. ```
   find_invalid_letters.sq(x, dest_type, ...,
                           NA_letter = getOption("tidysq_NA_letter"))
   ```

1) Zwraca listę, której każdy element jest wektorem napisów; każdy element zawiera *Litery* występujące w odpowiadającej *Sekwencji*, a nie wchodzące w skład *Standardowego Alfabetu* docelowego typu.

Parametry wejściowe

- `x` : `[sq]` – obiekt z *Sekwencjami* zakodowanymi w formacie udostępnianym przez *Pakiet*
- `dest_type` : `[napis]` – jeden z *Typów Standardowych*
- `...` : `[dowolne]` – dodatkowe argumenty do metod
- `NA_letter` : `[napis]` – *Litera* reprezentująca brak danych

Wyjście

`[lista]` – lista wektorów napisów o długości równej wejściowemu obiektowi `sq`

Odpowiadające wymaganie

Wyszukanie *Liter* niezgodnych z określonym

Funkcja generyczna `find_motifs()`

Metody

1. ```
   find_motifs.sq(x, name, motifs, ...,
                  NA_letter = getOption("tidysq_NA_letter"))
   ```

1) Wyszukuje wszystkie wystąpienia podanych *Motywów* w podanych *Sekwencjach*.

Parametry wejściowe

- `x` : `[sq]` – obiekt *Sekwencji*
- `name` : `[wektor napisów]` – nazwy sekwencji
- `motifs` : `[wektor napisów]` – *Motywy* do wyszukania
- `...` : `[dowolne]` – dodatkowe argumenty do metod
- `NA_letter` : `[napis]` – *Litera* reprezentująca brak danych

Wyjście

`[tibble]` – ramka danych o następujących kolumnach:

- `names` : `[wektor napisów]` – nazwa *Sekwencji*, w której *Motyw* został odnaleziony

- `found : [sq]` – podsekwencja odpowiadająca znalezionemu *Motywowi*
- `sought : [wektor napisów]` – znaleziony *Motyw*
- `start : [wektor liczb naturalnych]` – indeks początku (inkluzywnego) podsekwencji w oryginalnej *Sekwencji*
- `end : [wektor liczb naturalnych]` – indeks końca (inkluzywnego) podsekwencji w oryginalnej *Sekwencji*

Odpowiadające wymaganie

Wyszukiwanie *Motywów*

Funkcja `get_sq_lengths()`

Zwraca długości *Sekwencji*.

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji*

Wyjście

`[wektor liczb naturalnych]` – wektor długości sekwencji

Funkcja `get_standard_alphabet()`

Zwraca *Standardowy Alfabet* odpowiadający danemu *Typowi*.

Parametry wejściowe

- `type : [napis]` – jeden z *Typów Standardowych*

Wyjście

`[alfabet]` – predefiniowany *Alfabet* dla podanego *Typu*

Funkcja `get_tidysq_options()`

Zwraca podzbiór wszystkich globalnych opcji zawierający te pochodzące z *Pakietu*.

Wyjście

`[lista]` – nazwana lista opcji pochodzących z *Pakietu*

Operator `%has%`

Zwraca wektor logiczny wskazujący, czy *Sekwencja* zawiera wszystkie *Motywy*.

Operandy

- `x : [sq]` – obiekt *Sekwencji*
- `y : [wektor napisów]` – *Motywy* do wyszukania

Wyjście

`[wektor wartości logicznych]` – ‘prawda‘, jeśli odpowiadająca *Sekwencja* zawiera wszystkie podane *Motywy*, ‘fałsz‘ w przeciwnym wypadku

Odpowiadające wymaganie

Sprawdzanie zawierania *Motywów*

Funkcja generyczna `import_sq()`

Metody

1. `import_sq.AAbin(object, ...)`
2. `import_sq.DNAbin(object, ...)`
3. `import_sq.alignment(object, ...)`
4. `import_sq.AAString(object, ...)`
5. `import_sq.DNAString(object, ...)`
6. `import_sq.RNAString(object, ...)`
7. `import_sq.BString(object, ...)`
8. `import_sq.AAStringSet(object, ...)`
9. `import_sq.DNAStringSet(object, ...)`
10. `import_sq.RNAStringSet(object, ...)`
11. `import_sq.BStringSet(object, ...)`
12. `import_sq.XStringSetList(object, ...)`
13. `import_sq.SeqFastaAA(object, ...)`
14. `import_sq.SeqFastadna(object, ...)`
15. `import_sq.list(object, ..., separate = TRUE)`

1–2) Importuje *Sekwencje* aminokwasów bądź DNA z obiektu z pakietu `ape`.
3) Importuje *Sekwencje* niewiadomego *Typu* z macierzy z pakietu `ape`.
4–7) Importuje pojedynczą *Sekwencję* odpowiedniego *Typu* (aminokwasowego, DNA, RNA bądź *Nieokreślonego*) z obiektu z pakietu `Biostrings`.
8–11) Importuje wektor *Sekwencji* odpowiedniego *Typu* (aminokwasowego, DNA, RNA bądź *Nieokreślonego*) z obiektu z pakietu `Biostrings`.
12), 15) Dla każdego elementu listy importuje tenże, a następnie zwraca listę zaimportowanych elementów.
13–14) Importuje *Sekwencje* aminokwasów bądź DNA z obiektu z pakietu `seqinr`.

Parametry wejściowe

- `object : [dowolne]` – importowany obiekt
- `... : [dowolne]` – dodatkowe argumenty do metod
- `separate : [wartość logiczna]` – flaga określająca, czy w przypadku list obiektów wynikowe ramki danych powinny być połączone w jedną

Wyjście

`[tibble]` – ramka danych o następujących kolumnach:

- `name : [wektor napisów]` – nazwy *Sekwencji*
- `sq : [sq]` – zaimportowane *Sekwencje*

Odpowiadające wymaganie

Import obiektów z innych pakietów

Funkcja `is_empty_sq()`

Sprawdza, które *Sekwencje* są puste (innymi słowy, mają długość równą 0).

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji*

Wyjście

`[wektor wartości logicznych]` – wektor określający, czy dana *Sekwencja* jest pusta

Funkcja `is.sq()` i pochodne

1. `is.sq(x)`
2. `is.sq_ami_bsc(x)`
3. `is.sq_ami_ext(x)`
4. `is.sq_dna_bsc(x)`
5. `is.sq_dna_ext(x)`
6. `is.sq_rna_bsc(x)`
7. `is.sq_rna_ext(x)`
8. `is.sq_unt(x)`
9. `is.sq_atp(x)`
10. `is.sq_ami(x)`
11. `is.sq_dna(x)`
12. `is.sq_rna(x)`

1) Sprawdza, czy obiekt należy do klasy `sq`.
2–9) Sprawdza, czy obiekt ma wskazany *Typ*.
10–12) Sprawdza, czy obiekt ma wskazany *Typ Standardowy* z pominięciem rozszerzoności *Alfabetu*.

Parametry wejściowe

- `x : [sq]` – sprawdzany obiekt

Wyjście

`[wartość logiczna]` – wynik sprawdzania

Metoda `print()`

Metody

1. ```
   print.sq(x, ...,
            max_sequences = getOption("tidysq_print_max_sequences"),
            use_color = getOption("tidysq_print_use_color"),
            NA_letter = getOption("tidysq_NA_letter"),
            letters_sep = NULL)
   ```

   1) Wypisuje na konsolę sformatowany obiekt klasy `sq`.

Parametry wejściowe

- `x : [sq]` – wektor wyświetlanych *Sekwencji*
- `... : [dowolne]` – dodatkowe argumenty do metod
- `max_sequences : [liczba naturalna]` – maksymalna liczba wyświetlanych *Sekwencji*
- `use_color : [wartość logiczna]` – flaga określająca, czy używać kolorów przy wypisywaniu *Sekwencji*
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych
- `letters_sep : [napis]` – ciąg znaków umieszczany pomiędzy kolejnymi literami *Sekwencji*

Funkcja `random_sq()`

Generuje *Sekwencje* zgodnie z podanymi parametrami.

Parametry wejściowe

- `n : [liczba naturalna]` – liczba *Sekwencji*
- `len : [liczba naturalna]` – długość *Sekwencji*
- `alphabet : [napis lub wektor napisów]` – nazwa typu *Sekwencji* bądź wektor dostępnych *Liter*
- `sd : [liczba naturalna]` – odchylenie standardowe długości *Sekwencji*
- `use_gap : [wartość logiczna]` – flaga określająca, czy w przypadku *Standardowych Alfabetów* mają być wykorzystywane znaki `*` oraz `-`

Wyjście

`[sq]` – wygenerowane sekwencje

Funkcja `read_fasta()`

Wczytuje *Sekwencje* z pliku FASTA wraz z ich nazwami.

Parametry wejściowe

- `file_name : [napis]` – ścieżka do pliku w formacie FASTA
- `alphabet : [napis lub wektor napisów]` – nazwa typu *Sekwencji* bądź wektor dostępnych *Liter*
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych
- `safe_mode : [wartość logiczna]` – flaga określająca, czy *Alfabet* może ulec zmianie w celu zagwarantowania wczytania wszystkich *Liter*
- `on_warning : [napis]` – specyfikacja zachowania w razie wystąpienia wyjątku
- `ignore_case : [wartość logiczna]` – flaga określająca, czy duże i małe znaki mają być traktowane jako ta sama *Litera*
   ```

Wyjście

`[tibble]` – zaimportowany obiekt w postaci ramki danych z następującymi kolumnami:

- `name : [wektor napisów]` – nazwy *Sekwencji*
- `sq : [sq]` – wczytane *Sekwencji*

Odpowiadające wymaganie

Wczytanie pliku FASTA

Funkcja generyczna `remove_ambiguous()`

Metody

1. ```
   remove_ambiguous.sq_ami_ext(x, by_letter = FALSE, ...,
                               NA_letter = getOption("tidysq_NA_letter"))
   ```

2. ```
   remove_ambiguous.sq_dna_ext(x, by_letter = FALSE, ...,
                               NA_letter = getOption("tidysq_NA_letter"))
   ```

3. ```
   remove_ambiguous.sq_rna_ext(x, by_letter = FALSE, ...,
                               NA_letter = getOption("tidysq_NA_letter"))
   ```

4. ```
   remove_ambiguous.sq_ami_bsc(x, by_letter = FALSE, ...,
                               NA_letter = getOption("tidysq_NA_letter"))
   ```

5. ```
   remove_ambiguous.sq_dna_bsc(x, by_letter = FALSE, ...,
                               NA_letter = getOption("tidysq_NA_letter"))
   ```

6. ```
   remove_ambiguous.sq_rna_bsc(x, by_letter = FALSE, ...,
                               NA_letter = getOption("tidysq_NA_letter"))
   ```

1–3) Usuwa *Sekwencje* zawierające *Litery Niejednoznaczne* bądź same takie *Litery*.
4–6) Zwraca oryginalny obiekt bez dokonywania zmian.

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji* o *Typie Standardowym*
- `by_letter : [wartość logiczna]` – flaga określająca, czy usunięciu ma ulegać pojedyncza *Litera*, czy cała *Sekwencja*
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[sq]` – obiekt o analogicznym *Typie Podstawowym* bez *Liter Niejednoznacznych*

Odpowiadające wymaganie

Usunięcie *Liter* z *Alfabetu Rozszerzonego*

Funkcja generyczna `remove_na()`

Metody

1. ```
   remove_na.sq(x, by_letter = FALSE, ...,
                NA_letter = getOption("tidysq_NA_letter"))
   ```

   1) Usuwa *Sekwencje* zawierające braki danych bądź same takie wartości.

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji*
- `by_letter : [wartość logiczna]` – flaga określająca, czy usunięciu ma ulegać pojedyncza *Litera*, czy cała *Sekwencja*
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[sq]` – obiekt o tym samym *Typie* bez braków danych

Odpowiadające wymaganie

Usunięcie braków danych

Funkcja generyczna `reverse()`

Metody

1. ```
   reverse.sq(x, ...,
              NA_letter = getOption("tidysq_NA_letter"))
   ```

   1) Odwraca kolejność elementów (*Liter*) w *Sekwencjach*.

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji*
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[sq]` – obiekt o tym samym *Typie*

Odpowiadające wymaganie

Odwrócenie kolejności ciągu *Liter*

Funkcja `sq()`

Tworzy *Sekwencje* z wektora napisów.

Parametry wejściowe

- `x : [wektor napisów]` – reprezentacja *Sekwencji* biologicznych w postaci napisów
- `alphabet : [napis lub wektor napisów]` – nazwa typu *Sekwencji* bądź wektor dostępnych *Liter*
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

- `safe_mode : [wartość logiczna]` – flaga określająca, czy *Alfabet* może ulec zmianie w celu zagwarantowania wczytania wszystkich *Liter*
- `on_warning : [napis]` – specyfikacja zachowania w razie wystąpienia wyjątku
- `ignore_case : [wartość logiczna]` – flaga określająca, czy duże i małe znaki mają być traktowane jako ta sama *Litera*

Wyjście

`[sq]` – wektor *Sekwencji* poddanych *Pakowaniu*

Funkcja generyczna `sq_type()`

Metody

1. `sq_type.sq(x, ...)`

1) Zwraca *Typ* podanego obiektu.

Parametry wejściowe

- `x : [sq]` – sprawdzany obiekt
- `... : [dowolne]` – dodatkowe argumenty do metod

Wyjście

`[napis]` – tekstowa reprezentacja *Typu*

Funkcja generyczna `sq_type<-`

Metody

1. `‘sq_type<-.sq‘(x, value)`

1) Przypisuje obiektowi nowy *Typ*

Parametry wejściowe

- `x : [sq]` – modyfikowany obiekt
- `value : [napis]` – jeden z *Typów Standardowych*

Wyjście

`[sq]` – zmodyfikowany argument `x` metody

Funkcja `sqapply()`

Aplikuje dowolną funkcję do tekstowej reprezentacji *Sekwencji*.

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji*
- `fun : [funkcja]` – aplikowana operacja
- `... : [dowolne]` – dodatkowe argumenty do metod
- `single_string` – flaga określająca, czy *Sekwencja* powinna być traktowana jako pojedynczy napis
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[lista]` – wyniki wykonania *Operacji* dla każdej *Sekwencji* z osobna

Odpowiadające wymaganie

Aplikacja dowolnej funkcji do *Sekwencji*

Funkcja generyczna `substitute_letters()`

Metody

1. ```
   substitute_letters.sq(x, encoding, ...,
                         NA_letter = getOption("tidysq_NA_letter"))
   ```

1) Zastępuje wszystkie wystąpienia *Liter* określonych w nazwach argumentu `encoding` poprzez odpowiadające im *Litery* w tymże wektorze.

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji*
- `encoding : [wektor napisów]` – słownik podstawień *Liter* w miejsce innych *Liter*
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[sq]` – sekwencja o zmodyfikowanym, *Nietypowym Alfabecie*

Odpowiadające wymaganie

Zastąpienie *Liter* innymi

Funkcja generyczna `translate()`

Metody

1. ```
   translate.sq_dna_bsc(x, table = 1, ...,
                        NA_letter = getOption("tidysq_NA_letter"))
   ```

2. ```
   translate.sq_rna_bsc(x, table = 1, ...,
                        NA_letter = getOption("tidysq_NA_letter"))
   ```

1–2) Odkodowuje *Sekwencje* aminokwasów na podstawie trzyliterowych kodonów DNA bądź RNA.

Parametry wejściowe

- `x : [sq]` – obiekt typu DNA/RNA o *Standardowym Alfabecie*
- `table : [liczba naturalna]` – numer tabeli kodonów używanej przy translacji
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[sq]` – *Sekwencje* aminokwasów bez *Liter Niejednoznacznych*

Odpowiadające wymaganie

Translacja DNA/RNA do aminokwasów
   ```

Funkcja generyczna `typify()`

Metody

1. ```
   typify.sq(x, dest_type, ...,
             NA_letter = getOption("tidysq_NA_letter"))
   ```

   1) Nadaje obiektowi klasy `sq` wybrany *Typ Standardowy*.

Parametry wejściowe

- `x : [sq]` – wektor *Sekwencji*
- `dest_type : [napis]` – jeden z *Typów Standardowych*
- `... : [dowolne]` – dodatkowe argumenty do metod
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Wyjście

`[sq]` – obiekt o docelowym *Typie*

Funkcja `write_fasta()`

Zapisuje *Sekwencje* na dysku w postaci pliku FASTA.

Parametry wejściowe

- `x : [sq]` – obiekt *Sekwencji*
- `name : [wektor napisów]` – nazwy sekwencji
- `file : [napis]` – ścieżka do pliku, do którego sekwencje mają zostać zapisane
- `width : [liczba naturalna]` – maksymalna szerokość sekwencji wewnątrz pliku
- `NA_letter : [napis]` – *Litera* reprezentująca brak danych

Odpowiadające wymaganie

Zapis do pliku FASTA

Operator `==`

Zwraca wektor logiczny wskazujący, czy odpowiadające sobie *Sekwencje* są równoważne.

Operandy

- `e1 : [sq]` – obiekt *Sekwencji*
- `e2 : [sq lub wektor napisów]` – obiekt *Sekwencji*

Wyjście

`[wektor wartości logicznych]` – ‘prawda‘, jeśli i-te sekwencje obydwu obiektów są sobie równoważne, w przeciwnym razie ‘fałsz‘
   ```

Metody pakietu `vctrs`

Oprócz powyższych funkcji i metod zaimplementowane zostały również liczne metody dla podklas `sq` oraz klasy `sq_alphabet` dla poniższych funkcji generycznych pochodzących z pakietu `vctrs`:

- `vec_cast()` – zwraca przekazany obiekt o docelowej klasie
- `vec_ptype_abbr()` – zwraca krótką nazwę *Typu* obiektu
- `vec_ptype_full()` – zwraca pełną nazwę *Typu* obiektu
- `vec_ptype2()` – zwraca wspólny prototyp obydwu obiektów

Ich uwzględnienie umożliwia wykorzystanie sporej części funkcjonalności oferowanych przez `vctrs`, między innymi funkcję `vec_c()`, łączącą dowolną liczbę obiektów w sposób lepiej zdefiniowany i bardziej przewidywalny niż dla bazowej funkcji `c()`.

3 Interfejs C++

Elementy tego interfejsu znajdują się w przestrzeni nazw `tidysq`. W obrębie tej przestrzeni znajdują się przestrzenie:

- `util` – zawiera funkcje i obiekty pomocnicze, niezwiązane bezpośrednio z funkcjonalnością,
- `ops` – zawiera szablony klasowe związane z *Operacjami* wykonywanymi na *Sekwencjach*,
- `io` – zawiera klasy i funkcje związane z odczytem i zapisem z dysku,
- `constants` – zawiera definicje stałych obiektów.

W kolejnych podpunktach zaprezentowane są udostępniane interfejsy wraz z podaniem przestrzeni nazw z pominięciem przestrzeni tidysq, która występuje na początku wszystkich nazw.

Podstawowe typy proste

- `LenSq` – Alias na typ całkowitoliczbowy, oznacza długość *Sekwencji* bądź liczbę *Sekwencji*.
- `ElementPacked` – Alias na typ o rozmiarze bajta, oznacza spakowaną wartość bitową.
- `ElementRaws` – Alias na typ o rozmiarze bajta, oznacza niespakowaną wartość bitową.
- `ElementInts` – Alias na typ całkowitoliczbowy bez znaku, oznacza niespakowaną wartość bitową.
- `ElementStrings` – Alias na ciąg znaków, oznacza niespakowaną wartość w postaci ciągu znaków.
- `ElementStringSimple` – Alias na znak, oznacza niespakowaną wartość przy *Alfabecie Prostym*.
- `ElementStringMultichar` – Alias na ciąg znaków, oznacza niespakowaną wartość w postaci ciągu znaków przy *Alfabecie Wieloznakowym*.
- `AlphSize` – Alias na typ całkowitoliczbowy bez znaku, oznacza *Rozmiar Alfabetu*.
- `LetterValue` – Alias na typ całkowitoliczbowy bez znaku, onzacza wartość danej *Litery* w *Alfabecie*.
- `Letter` – Alias na ciąg znaków, oznacza *Literę*.
- `SimpleLetter` – Alias na znak, oznacza *Literę Prostą*.
- `SqType` – Typ wyliczeniowy, mogący przyjąć wartość odpowiadającą *Typowi Alfabetu*. Możliwe wartości prezentuje tabela 1

DNA_BSC	*Podstawowy Alfabet DNA*
DNA_EXT	*Rozszerzony Alfabet DNA*
RNA_BSC	*Podstawowy Alfabet RNA*
RNA_EXT	*Rozszerzony Alfabet RNA*
AMI_BSC	*Podstawowy Alfabet Aminokwasowy*
RNA_EXT	*Rozszerzony Alfabet Aminokwasowy*
ATP	*Alfabet Nietypowy*
UNT	*Alfabet Nieokreślony*

Tabela 1: Wartości typu wyliczeniowego `SqType` oraz odpowiadające im `Typy Alfabetu`.

Abstrakcyjne typy mapujące

Są to typy, których zadaniem jest zgrupowanie innych typów w celu wykorzystania możliwości metaprogramowania.

Ponieważ główne typy używane w bibliotece są wzorcami klasowymi (jak np. `Sequence`, `Sq`), które mogą wspierać wiele różnych backendów jeśli chodzi o przechowywanie danych, wymagają one podania jako parametry typu struktur z odpowiednimi typami członkowskimi. Poniżej opisane są warunki, jakie muszą spełniać typy mapujące, jeśli użytkownik chce zapewnić własne typy. W kolejnej podsekcji opisane są typy dołączone razem z biblioteką.

Dwie klasy są podstawowymi typami mapującymi:

- `InternalType` – klasa określająca backend wektorowy Sekwencji oraz wektora Sekwencji. Klasa `X_IT` rozszerzająca tę klasę musi posiadać publiczne typy członkowskie spełniające warunki określone w tabeli 2.

Typ członkowski	Wymaganie dotyczące typu
`SequenceType`	Alias na `Sequence<X_IT>`.
`SequenceElementType`	Alias na `ElementPacked`
`SequenceContentStorageType`	Typ wektorowy obiektów typu `SequenceElementType`.
`SequenceContentAccessType`	Typ zwracany przez użycie operatora `[]` na `SequenceContentStorageType`.
`SequenceContentConstAccessType`	Typ zwracany przez użycie operatora `[]` na `SequenceContentStorageType` w stałym kontekście.
`SqType`	Alias na `Sq<X_IT>`.
`SqContentStorageType`	typ wektorowy, którego każdy element odpowiada obiektowi typu SequenceType.
`SqContentAccessType`	Typ zwracany przez użycie operatora `[]` na `SqContentStorageType`.
`SqContentConstAccessType`	Typ zwracany przez użycie operatora `[]` na `SqContentStorageType` w stałym kontekście.

Tabela 2: Typy członkowskie, które musi posiadać klasa `X_IT` rozszerzająca klasę `InternalType`, oraz wymagania dotyczące tych typów.

- `ProtoType` – klasa określająca formę przechowywania Niespakowanej Sekwencji oraz wektora Niespakowanych Sekwencji. Klasa `Y_PT` rozszerzająca tę klasę musi posiadać publiczny typ członkowski `ProtoSequenceElementType` oznaczający element Niespakowanej Sekwencji.

Do mapowania wykorzystywany jest wzorzec klasowy `util::TypeBinder`. Posiada on dwa parametry typu: `INTERNAL` oraz `PROTO`. Aby móc korzystać z pozostałych obiektów występujących w bibliotece, które posiadają parametry typu `INTERNAL` oraz `PROTO` o wartościach odpowiednio `X_IT` i `Y_PT` (spełniające wymagania odpowiednio `InternalType` oraz `ProtoType`), należy zapewnić specjalizację typu `util::TypeBinder<X_IT, Y_PT>`. Specjalizacja ta powinna posiadać publiczne typy członkowskie, spełniające warunki określone w tabeli 3

Oprócz tego w bibliotece występuje wzorzec klasowy `util::UniversalTypeBinder`, parametryzowany typami `INTERNAL` oraz `PROTO` analogicznie jak `util::TypeBinder`, a także dodatkowe paramatry logiczne `PACKED` oraz `CONST`. Nie wymaga on jednak specjalizacji przy dodawaniu nowego backendu. Posiada on uogólnione definicje typów zaprezentowane w tabeli 4.

Specjalizacje i implementacje typów mapujących

W pakiecie występują gotowe rozszerzenia typów mapujących, obejmujące podstawowe przypadki:

Typ członkowski	Wymaganie dotyczące typu
`ProtoSequenceType`	Alias na `ProtoSequence<X_IT, Y_PT>`.
`ProtoSequenceContentStorageType`	typ wektorowy przechowujący Niespakowane Sekwencje w formie Y_PT.
`ProtoSequenceContentAccessType`	Typ zwracany przez użycie operatora `[]` na `ProtoSequenceContentStorageType`.
`ProtoSequenceContentConstAccessType`	Typ zwracany przez użycie operatora `[]` na `ProtoSequenceContentStorageType` w stałym kontekście.
`ProtoSqType`	Alias na `ProtoSq<X_IT, Y_PT>`.
`ProtoSqListConstructorType`	typ wektorowy, który może zostać podany jako lista inicjalizacyjna w konstruktorze i może być przekazany, aby skonstruować obiekt typu backendu `ProtoSq<X_IT, Y_PT>`.
`ProtoSqContentStorageType`	typ wektorowy, który jest backendową reprezentacją obiektu `ProtoSq<X_IT, Y_PT>`.
`ProtoSqContentAccessType`	Typ zwracany przez użycie operatora `[]` na `ProtoSqContentStorageType`.
`ProtoSqContentConstAccessType`	Typ zwracany przez użycie operatora `[]` na `ProtoSqContentStorageType` w stałym kontekście.

Tabela 3: Typy członkowskie, które musi posiadać specjalizacja wzorca klasowego `TypeBinder<X_IT, Y_IT>`, oraz wymagania dotyczące tych typów.

- `InternalType`:
  - `STD_IT` – backendem są wektory `std::vector` z biblioteki standardowej. Zalecane do używania w każdym przypadku, w którym nie jest konieczna bezpośrednia interakcja z R.
  - `RCPP_IT` – backendem są wrapery wektorowe z biblioteki `Rcpp` (`Rcpp::RawVector`, `Rcpp::IntegerVector`, itd.). Zalecane do używania wyłącznie kiedy to konieczne (np. podczas przekazywania danych bezpośrednio z R bądź do R).
- `ProtoType`:
  - `RAWS_IT` – elementy niespakowanej Sekwencji to wartości bitowe (`ElementRaws`).
  - `INTS_IT` – elementy niespakowanej Sekwencji to nieujemne wartości całkowitoliczbowe (`ElementInts`).
  - `STRINGS_IT` – elementy niespakowanej Sekwencji to napisy (`ElementStrings`).
  - `STRING_IT` – Niespakowana Sekwencja jest przechowywana w postaci ciągu znaków (`ElementStringSimple` lub `ElementStringMultichar`).

Stałe

Biblioteka zawiera szereg wartości stałych, związanych głównie z pewnymi pojęciami biologicznymi.

- `const short constants::BUFF_SIZE` – określa rozmiar bufora odczytu z dysku.
- `const char constants::NEW_SEQUENCE_CHAR` – znak oznaczający nową *Sekwencję* podczas odczytu pliku *FASTA*.
- `const char constants::NEW_LINE_CHAR` – znak końca linii podczas odczytu z pliku *FASTA*.
- `const Letter constants::DEFAULT_NA_LETTER` – domyślna litera do reprezentacji wartości braku danych.
- `const bool constants::DEFAULT_IGNORE_CASE` – domyślna wartość określająca, czy podczas wczytywania *Sekwencji* z napisów należy igrorować wielkość liter.
- `const std::unordered_map<SqType, const std::vector<Letter>> constants::STANDARD_LETTERS` – słownik mapujący *Typ Standardowy Sekwencji* na wektor odpowiadających *Liter Standardowych*.
- `const internal::AmbiguousDict constants::AMBIGUOUS_AMINO_MAP` – słownik mapujący *Litery Niejednoznaczne* z *Standardowego Rozszerzonego Alfabetu* reszt aminokwasowych na *Litery*.

Typ członkowski	Definicja typu
`ProtoOrNotSequenceType`	Odpowiadający parametrom `INTERNAL` oraz `PROTO` typ `ProtoSequenceType` dla `PACKED == false` lub `SequenceType` w przeciwnym przypadku.
`ProtoOrNotSequenceContentStorageType`	Odpowiadający parametrom `INTERNAL` oraz `PROTO` typ `ProtoSequenceContentStorageType` dla `PACKED == false` lub `SequenceContentStorageType` w przeciwnym przypadku.
`ProtoOrNotSequenceContentAccessType`	Odpowiadający parametrom `INTERNAL` oraz `PROTO` typ `ProtoSequenceContentAccessType` dla `PACKED == false` lub `SequenceContentAccessType` w przeciwnym przypadku. Jeśli `CONST == true` zamiast powyższych typów wartością jest analogicznie `ProtoSequenceContentConstAccessType` lub `SequenceContentConstAccessType`.
`ProtoOrNotSqType`	Odpowiadający parametrom `INTERNAL` oraz `PROTO` typ `ProtoSqType` dla `PACKED == false` lub `SqType` w przeciwnym przypadku.
`ProtoOrNotSqContentStorageType`	Odpowiadający parametrom `INTERNAL` oraz `PROTO` typ `ProtoSqContentStorageType` dla `PACKED == false` lub `SqContentStorageType` w przeciwnym przypadku.
`ProtoOrNotSqContentAccessType`	Odpowiadający parametrom `INTERNAL` oraz `PROTO` typ `ProtoSqContentAccessType` dla `PACKED == false` lub `SqContentAccessType` w przeciwnym przypadku. Jeśli `CONST == true` zamiast powyższych typów wartością jest analogicznie `ProtoSqContentConstAccessType` lub `SqContentConstAccessType`.

Tabela 4: Typy członkowskie wzorca klasowego `UniversalTypeBinder<INTERNAL, PROTO, PACKED, CONST>`.

- `const internal::AmbiguousDict constants::AMBIGUOUS_DNA_MAP` – słownik mapujący *Litery Niejednoznaczne* z *Standardowego Rozszerzonego Alfabetu* DNA na *Litery*.
- `const internal::AmbiguousDict constants::AMBIGUOUS_RNA_MAP` – słownik mapujący *Litery Niejednoznaczne* z *Standardowego Rozszerzonego Alfabetu* RNA na *Litery*.
- `const internal::ComplementTable constants::BSC_COMPLEMENT_TABLE` – słownik mapujący wartość *Litery* ze *Standardowego Podstawowego Alfanbetu* DNA bądź RNA na wartość komplementarną.
- `const internal::ComplementTable constants::EXT_COMPLEMENT_TABLE` – słownik mapujący wartość *Litery* ze *Standardowego Rozszerzonego Alfanbetu* DNA bądź RNA na wartość komplementarną.
- `const internal::CodonTable constants::CODON_TABLE_1` – słownik mapujący wartości licznowe odpowiadające kodonom (trójkom *Liter* z *Podstawowego Alfabetu* DNA) na *Litery* z *Alfabetu* aminokwasowego według biologicznych zasad translacji tabeli nr 1.
- `const std::unordered_map<int, const internal::CodonTable> constants::CODON_DIFF_TABLES` – słownik mapujący numer tabeli translacji na tabelę translacji zawierającą różnicę względem tabeli nr 1.
- `const std::unordered_map<int, const internal::CodonTable> constants::AMB_CODON_DIFF_TABLES` – słownik mapujący numer tabeli translacji zawierającej niedoprecyzowane mapowanie na tabelę translacji zawierającą różnicę względem tabeli nr 1.
- `constants::WarningLevel` – typ wyliczeniowy zawierające możliwe poziomy raportowania o sytuacjach wyjątkowych w R.
- `const constants::WarningLevel constants::DEFAULT_WARNING_LEVEL` – domyślny poziom raportowania.

- `const std::unordered_map<std::string, WarningLevel> WARNING_LEVEL_NAMES` – słownik mapujący słowną nazwę poziomu raportowania na poziom raportowania.

Klasa `Alphabet`

`Alphabet` – klasa reprezentująca *Alfabet*, to znaczy mapowanie z wartości liczbowej (indeksu) na znak bądź ciąg znaków (*Literę*)

Typy członkowskie

- `const_iterator` – iterator po parach (wartość liczbowa odpowiadająca *Literze*, *Litera*) przechowywanych w *Alfabecie*

Konstruktory

Sygnatury:

1. ```
   Alphabet(
       const std::vector<Letter> &letters,
       const SqType &type,
       const Letter &NA_letter = constants::DEFAULT_NA_LETTER,
       const bool ignore_case = constants::DEFAULT_IGNORE_CASE)
   ```

2. ```
   Alphabet(
       const SqType &type,
       const Letter &NA_letter = constants::DEFAULT_NA_LETTER,
       const bool ignore_case = constants::DEFAULT_IGNORE_CASE)
   ```

3. ```
   Alphabet(
       const std::vector<Letter> &letters,
       const Letter &NA_letter = constants::DEFAULT_NA_LETTER,
       const bool ignore_case = constants::DEFAULT_IGNORE_CASE)
   ```

Tworzy nowy obiekt *Alfabetu*
1) tworzy *Alfabet* z jawnym podaniem *Liter* i *Typu*
2) tworzy *Alfabet Standardowy* z podaniem *Typu Standardowego*
3) tworzy *Alfabet*, dopasowując najmniejszy *Typ Standardowy* zawierający wszystkie podane *Litery* i rozszerzając wektor *Liter* o pozostałe występujące w dopasowanym *Typie*; w przeciwnym razie ustawia typ *Nieokreślony*

Parametry:

- `letters` – wektor liter zawartych w *Alfabecie*
- `type` – element wyliczeniowej klasy reprezentujący *Typ* obiektu
- `NA_letter` – litera reprezentująca brak danych
- `ignore_case` – flaga określająca, czy duże i małe znaki traktowane są jako ta sama *Litera*

Metody

- ```
  LetterValue size() const
  ```

  Zwraca liczbę Liter w Alfabecie.

- ```
  const Letter &operator[](LetterValue index) const
  ```

  Zwraca referencję na *Literę* o podanej wartości. W przypadku niewystępowania *Litery* o podanej wartości, zwracana jest referencja na `NA_letter`.

- ```
  const SimpleLetter &get_simple_letter(LetterValue index) const
  ```

  Zwraca referencję na *Literę Prostą* (jednoznakową) o podanej wartości. W przypadku niewystępowania *Litery Prostej* o podanej wartości, zwracana jest referencja na `NA_letter`.

- ```
  const LetterValue &NA_value() const
  ```

  Zwraca referencję na wartość (indeks) odpowiadający znakowi brakowi wartości.
  ```

- `const Letter &NA_letter() const`

  Zwraca referencję na `NA_letter`.

- `const SqType &type() const`

  Zwraca referencję na *Typ Alfabetu*.

- `const AlphSize &alphabet_size() const`

  Zwraca referencję na liczbę bitów używaną do zakodowania jednej *Litery*.

- `bool is_simple() const`

  Informuje, czy dany *Alfabet* zawiera wyłącznie jednoznakowe *Litery*.

- `bool ignores_case() const`

  Informuje, czy dany *Alfabet* traktuje wielkie i małe litery jako tę samą *Literę*.

- ```
  const_iterator begin() const
  const_iterator cbegin() const
  ```

  Zwraca iterator wskazujący na początek wektora *Liter*.

- ```
  const_iterator end() const
  const_iterator cend() const
  ```

  Zwraca iterator wskazujący na koniec wektora *Liter*.

- ```
  bool operator==(const Alphabet &other) const
  bool operator!=(const Alphabet &other) const
  ```

  Porównuje równość mapowań z wartości (indeksu) na *Literę* oraz wartości `NA_letter`.

- ```
  LetterValue match_value(const ElementRaws &letter) const
  LetterValue match_value(const ElementInts &letter) const
  ```

  Zwraca przekazaną wartość, jeśli zawiera się ona w wektorze dozwolonych indeksów. W przeciwnym wypadku zwraca indeks odpowiadający znakowi brakowi wartości.

- `LetterValue match_value(const ElementStringSimple &letter) const`

  Zwraca wartość (indeks) odpowiadającą przekazanej *Literze Prostej*. Jeśli takowa nie istnieje, zwraca indeks odpowiadający znakowi brakowi wartości.

- `LetterValue match_value(const Letter &letter) const`

  Zwraca wartość (indeks) odpowiadającą przekazanej *Literze*. Jeśli takowa nie istnieje, zwraca indeks odpowiadający znakowi brakowi wartości.

- `bool contains(const Letter &letter) const`

  Sprawdza, czy *Alfabet* zawiera wskazaną *Literę*.

Klasa `ProtoSequence`

`ProtoSequence` – klasa reprezentująca rozpakowaną postać *Sekwencji*.

Parametry wzorca

- `INTERNAL` – podklasa typu `InternalType`,
- `PROTO` – podklasa typu `ProtoType`.

Typy członkowskie

- `ElementType` – typ pojedynczego elementu *Sekwencji*,
- `ContentStorageType` – typ wewnętrznego formatu przechowywania danych,

- `AccessType` – typ zwracany w wyniku indeksacji obiektu,
- `ConstAccessType` – typ zwracany w wyniku indeksacji obiektu w stałym kontekście.

Konstruktory

Sygnatury:

1. `ProtoSequence(const ContentStorageType &content)`
2. `ProtoSequence(const LenSq length)`
3. `ProtoSequence()`
4. `ProtoSequence(const std::initializer_list<ElementType> &list)`

Tworzy nowy obiekt *Sekwencji* niepoddanej *Pakowaniu*
1) tworzy *Sekwencję* o zawartości wziętej z podanego obiektu
2) tworzy *Sekwencję* o podanej długości bez inicjalizacji zawartości
3) tworzy pustą *Sekwencję* o długości 0
4) tworzy *Sekwencję*, przekazując listę inicjalizującą do konstruktora zawartości

Parametry:

- `content` – wektor `Rcpp` albo obiekt klasy `string` zawierający *Sekwencję*
- `length` – długość wektora przechowującego *Spakowaną Sekwencję*
- `list` – lista inicjalizująca zawierająca *Sekwencję*

Metody

- ```
  AccessType operator[](const LenSq index)
  ConstAccessType operator[](const LenSq index) const
  ```

  Zwraca element znajdujący się w miejscu wektora wskazanym przez indeks.

- ```
  LenSq size() const
  ```

  Zwraca liczbę elementów, inaczej długość *Sekwencji*.

- ```
  const ContentStorageType &content() const
  ```

  Zwraca referencję na zawartość *Sekwencji*.

- ```
  bool operator==(const ProtoSequence<INTERNAL, PROTO> &other) const
  bool operator!=(const ProtoSequence<INTERNAL, PROTO> &other) const
  ```

  Porównuje równość zawartości.

- ```
  template<bool SIMPLE>
  ProtoSequenceInputInterpreter<INTERNAL, PROTO, SIMPLE> content_interpreter(
          const Alphabet &alphabet) const
  ```

  Tworzy interpreter zawartości przy podanym *Alfabecie*.

Metody wyspecjalizowane

- Dostępna w sytuacji, kiedy `PROTO == STRING_PT`:

  ```
  ProtoSequence<INTERNAL, PROTO>& operator+=(const Letter &letter)
  ProtoSequence<INTERNAL, PROTO>& operator+=(const SimpleLetter &letter)
  ```

  Wydłuża zawartość o przekazany element (*Literę* bądź *Literę Prostą*).
  ```

Klasa ProtoSq

`ProtoSq` – klasa reprezentująca wektor obiektów typu `ProtoSequence`.

Parametry wzorca

- `INTERNAL` – podklasa typu `InternalType`,
- `PROTO` – podklasa typu `ProtoType`.

Typy członkowskie

- `ElementType` – typ pojedynczego elementu *Sekwencji*,
- `ContentStorageType` – typ wewnętrznego formatu przechowywania danych.

Konstruktory

Sygnatury:

1. `ProtoSq(const ContentStorageType &content, const Alphabet &alphabet)`
2. `ProtoSq(const LenSq length, const Alphabet &alphabet)`
3. `ProtoSq(const ContentStorageType &content, const SqType &type)`
4. `ProtoSq(const LenSq length, const SqType &type)`

Tworzy wektor *Sekwencji* niepoddanych *Pakowaniu*
1) tworzy wektor z jawnym podaniem *Sekwencji* oraz *Alfabetu*
2) tworzy wektor o zadanej długości z podanym *Alfabetem*
3) tworzy wektor zapełniony podanymi *Sekwencjami* z *Alfabetem* odpowiadającym podanemu *Typowi*
4) tworzy wektor o zadanej długości z *Alfabetem* odpowiadającym podanemu *Typowi*

Parametry:

- `content` – wektor *Sekwencji* nie poddanych pakowaniu
- `length` – liczba *Sekwencji*
- `alphabet` – *Alfabet* używany przy *Pakowaniu Sekwencji*
- `type` – *Typ Standardowy* odpowiadający docelowemu *Alfabetowi Standardowemu*

Metody

- ```
  ProtoSequenceProxy<INTERNAL, PROTO> operator[](const LenSq index)
  ProtoSequenceConstProxy<INTERNAL, PROTO> operator[](const LenSq index) const
  ```

  Zwraca proxy z obiektem typu `ProtoSequence` pod wskazanym indeksem.

- ```
  LenSq size() const
  ```

  Zwraca liczbę *Sekwencji*.

- ```
  Alphabet& alphabet()
  const Alphabet &alphabet() const
  ```

  Zwraca referencję na *Alfabet*.

- ```
  const SqType &type() const
  ```

  Zwraca referencję na *Typ*.

- ```
  bool operator==(const ProtoSq<INTERNAL, PROTO> &other) const
  bool operator!=(const ProtoSq<INTERNAL, PROTO> &other) const
  ```

  Porównuje równość alfabetu oraz zawieranych obiektów typu `ProtoSequence`.

- ```
  Sq<INTERNAL_OUT> pack() const
  ```

  Wykonuje *Pakowanie* i zwraca *Spakowany* obiekt typu `Sq`.

Klasa Sequence

`Sequence` – klasa reprezentująca *Spakowaną Sekwencję*.

Parametry wzorca

- `INTERNAL` – podklasa typu `InternalType`.

Typy członkowskie

- `ElementType` – typ pojedynczego elementu *Sekwencji*,
- `ContentStorageType` – typ wewnętrznego formatu przechowywania danych,
- `AccessType` – typ zwracany w wyniku indeksacji obiektu,
- `ConstAccessType` – typ zwracany w wyniku indeksacji obiektu w stałym kontekście,
- `iterator` – typ iteratora po elementach,
- `const_iterator` – typ iteratora po elementach w stałym kontekście.

Konstruktory

Sygnatury:

1. `Sequence(const ContentStorageType &content, const LenSq original_length)`
2. `Sequence(const LenSq content_length, const LenSq original_length)`
3. `Sequence()`

Tworzy nowy obiekt *Spakowanej Sekwencji*
1) tworzy *Spakowaną Sekwencję* o zawartości wziętej z podanego obiektu
2) tworzy *Spakowaną Sekwencję* o podanej długości bez inicjalizacji zawartości
3) tworzy pustą *Spakowaną Sekwencję* o długości 0

Parametry:

- `content` – wektor `Rcpp` albo obiekt klasy `string` zawierający *Spakowaną Sekwencję*
- `content_length` – długość wektora przechowującego *Spakowaną Sekwencję*
- `original_length` – ilość elementów (*Liter*) *Spakowanej Sekwencji*

Metody

- `LetterValue operator[](const std::pair<LenSq, AlphSize> &index) const`

  Zwraca wartość odpowiadającą elementowi znajdującemu się na i-tym miejscu *Sekwencji*, gdzie i jest zdefiniowane przez indeks.

- ```
  AccessType operator()(const LenSq index)
  ConstAccessType operator()(const LenSq index) const
  ```

  Zwraca element znajdujący się w miejscu wektora wskazanym przez indeks.

- ```
  iterator begin(const AlphSize& alph_size)
  const_iterator begin(const AlphSize& alph_size) const
  const_iterator cbegin(const AlphSize& alph_size) const
  ```

  Zwraca iterator wskazujący na początek wektora zawartości.

- ```
  iterator end(const AlphSize& alph_size)
  const_iterator end(const AlphSize& alph_size) const
  const_iterator cend(const AlphSize& alph_size) const
  ```

  Zwraca iterator wskazujący na koniec wektora zawartości.
  ```

- `LenSq original_length() const`

  Zwraca długość *Sekwencji*.

- `LenSq size() const`

  Zwraca liczbę elementów wektora zawartości.

- `ContentStorageType content() const`

  Zwraca zawartość *Spakowanej Sekwencji*.

- ```
  bool operator==(const Sequence<INTERNAL> &other) const
  bool operator!=(const Sequence<INTERNAL> &other) const
  ```

  Porównuje równość zawartości.

- `void trim(const LenSq packed_length, const Alphabet &alphabet)`

  Skraca wektor zawartości do długości wyznaczonej przez długość *Spakowanej Sekwencji*.

Klasa `Sq`

`Sq` – klasa reprezentująca wektor obiektów typu `Sequence`.

Parametry wzorca

- `INTERNAL` – podklasa typu `InternalType`.

Typy członkowskie

- `ElementType` – typ pojedynczego elementu *Sekwencji*,
- `ContentStorageType` – typ wewnętrznego formatu przechowywania danych. kontekście.

Konstruktory

Sygnatury:

1. `Sq(const ContentStorageType &content, const Alphabet &alphabet)`
2. `Sq(const LenSq length, const Alphabet &alphabet)`
3. `Sq(const LenSq length, const SqType &type)`
4. `Sq(const Alphabet &alphabet)`
5. `Sq(const SqType &type)`

Tworzy wektor *Spakowanych Sekwencji*
1) tworzy wektor z jawnym podaniem *Spakowanych Sekwencji* oraz *Alfabetu*
2) tworzy wektor o zadanej długości z podanym *Alfabetem*
3) tworzy wektor o zadanej długości z *Alfabetem* odpowiadającym podanemu *Typowi*
4) tworzy pusty wektor z podanym *Alfabetem*
5) tworzy pusty wektor z *Alfabetem* odpowiadającym podanemu *Typowi*

Parametry:

- `content` – wektor *Spakowanych Sekwencji*
- `length` – liczba *Sekwencji*
- `alphabet` – *Alfabet* użyty przy *Pakowaniu Sekwencji*
- `type` – *Typ Standardowy* odpowiadający docelowemu *Alfabetowi Standardowemu*

Metody
  ```

- `SequenceProxy<INTERNAL> operator[](const LenSq index)`

  Zwraca proxy z obiektem typu `Sequence` pod wskazanym indeksem.

- `LenSq size() const`

  Zwraca liczbę *Sekwencji*.

- `Alphabet& alphabet()`
  `const Alphabet &alphabet() const`

  Zwraca referencję na *Alfabet*.

- `const SqType &type() const`

  Zwraca referencję na *Typ*.

- `void push_back(const ElementType &sequence)`

  Wstawia przekazaną *Sekwencję* na ostatnie miejsce wektora.

- `bool operator==(const Sq<INTERNAL> &other) const`
  `bool operator!=(const Sq<INTERNAL> &other) const`

  Porównuje równość alfabetu oraz zawieranych obiektów typu `ProtoSequence`.

- `template<typename INTERNAL_OUT, typename PROTO_OUT>`
  `ProtoSq<INTERNAL_OUT, PROTO_OUT> unpack() const`

  Wykonuje *Rozpakowanie* i zwraca rozpakowany obiekt typu `ProtoSq`.

- `template<typename INTERNAL_OUT, typename PROTO_OUT>`
  `ProtoSq<INTERNAL_OUT, PROTO_OUT> unpack(const LenSq from, const LenSq to) const`

  Wykonuje *Rozpakowanie* na wskazanym podciągu *Sekwencji* i zwraca rozpakowany obiekt typu `ProtoSq`.

Klasa `ProtoSequenceInputInterpreter`

`ProtoSequenceInputInterpreter` – klasa zajmująca się iteracją po elementach niespakowanej *Sekwencji*, szczególnie istotna w przypadku *Alfabetu Wieloznakowego*.

Parametry wzorca

- `INTERNAL` – podklasa typu `InternalType`,
- `PROTO` – podklasa typu `ProtoType`,
- `SIMPLE` – wartość logiczna, określająca czy *Alfabet*, zgodnie z którym ma być interpretowana *Sekwencja*, jest *Prosty*.

Typy członkowskie

- `ContentStorageType` – typ wewnętrznego formatu przechowywania danych,
- `ElementType` – typ pojedynczego elementu *Sekwencji*,
- `ContentConstIteratorType` – typ iteratora po elementach w stałym kontekście.

Metody

- `LetterValue get_next_value()`

  Zwraca wartość odpowiadającą indeksowi odczytanej następnej *Litery* w *Alfabecie*.

- `ElementType get_next_element()`

  Zwraca następną odczytaną *Literę*. Jeśli takowa nie występuje w *Alfabecie*, zwraca `NA_letter`.

- `ElementType get_or_extract_next_element()`

  Zwraca następną odczytaną *Literę*. Jeśli takowa nie występuje w *Alfabecie*, wstawia ją do niego.

- `bool reached_end() const`

  Informuje, czy iterator po zawartości dotarł do jej końca.

- `LenSq interpreted_letters() const`

  Zwraca liczbę odczytanych *Liter*.

Klasy pośredniczące (proxy)

`ProtoSequenceProxy`, `ProtoSequenceConstProxy`, `SequenceProxy` i `SequenceConstProxy` są to wzorce klasowe zwracane podczas indeksacji obiektów `ProtoSq` oraz `Sq` (odpowiednio do nazw). Obiekty z członem *Const* w nazwie powstają podczas indeksacji w kontekście stałym. Każdy obiekt proxy jest pośrednikiem w dostępie do danej sekwencji. Ich istnienie umożliwia wyabstrahowanie interfejsu od backendu.

Obiekty proxy mogą być jawnie bądź niejawnie rzutowane do obiektu `Sequence` lub `ProtoSequence` odpowiadającemu rodzajowi obiektów przechowywanemu w indeksowanym obiekcie. Jawne rzutowanie następuje przez wywołanie metody `get()`. Obiekt wynikowy pozwala uzyskać dostęp do odpowiedniej *Sekwencji* przez referencję. Jeśli kontekst był stały, referencja ta jest niemodyfikowalna, w niestałym kontekście jest modyfikowalna – typ tej referencji jest określony przez odpowiedni typ `util::UniversalTypeBinder<INTERNAL, PROTO, PACKED, CONST>::ProtoOrNotSqContentAccessType`.

Obiekty proxy mogą być porównywane, przy czym porównywanie polega na porównaniu elementów, do których dostęp zapewniają.

Klasa `util::ResultWrapper`

`ResultWrapper` – klasa opakowująca rezultat w celu umożliwienia równoczesnego przekazania informacji o błędzie bądź ostrzeżenia.

Parametry wzorca

- `RESULT_TYPE` – dowolny typ.

Konstruktory

Sygnatury:

1. `ResultWrapper(ResultType result, const MessageType &message)`

Tworzy nowy obiekt opakowujący rezultat *Operacji* oraz potencjalne ostrzeżenie o błędzie
1) tworzy `ResultWrapper` z jawnym podaniem rezultatu oraz treści ostrzeżenia

Parametry:

- `result` – wynik działania *Operacji*
- `message` – ostrzeżenie bądź jego brak rzucone w trakcie działania *Operacji*

Metody

- `ResultType result() const`

  Zwraca wartość rezultatu wykonanej *Operacji*.

- `bool has_message() const`

  Informuje, czy *Operacja* zwróciła informację o błędzie.

- `std::string message_text() const`

  Wyciąga wartość tekstową pola `message_`, zawierającą treść informacji o błędzie.

Klasa `ops::OperationVectorToVector`

`ops::OperationVectorToVector` jest abstrakcyjnym funktorem, reprezentującym *Operację*, przekształcającą jeden wektor w inny wektor, element po elemencie. Jest ona głównie wykorzystywana w funkcji `sqapply`.

Parametry wzorca

- `VECTOR_IN` – typ wektora wejściowego,
- `ELEMENT_IN` – typ elementu wektora wejściowego,
- `VECTOR_OUT` – typ wektora wyjściowego,
- `VECTOR_OUT` – typ elementu wektora wejściowego.

Metody

- `virtual bool may_return_early(const VECTOR_IN &vector_in)`

  Sprawdza, czy wektor wejściowy spełnia warunek wczesnego powrotu danej *Operacji*.

- `virtual VECTOR_OUT return_early(const VECTOR_IN &vector_in)`

  Zwraca wektor wyjściowy bez wykonywania *Operacji* element po elemencie.

- ```
  virtual VECTOR_OUT initialize_vector_out(const VECTOR_IN &vector_in, const LenSq from,
      const LenSq to) = 0
  virtual VECTOR_OUT initialize_vector_out(const VECTOR_IN &vector_in)
  ```

  Inicjalizuje wektor wyjściowy. Jeśli podane są `from` i `to`, wektor jest inicjalizowany tylko dla wartości odpowiadających zakresowi wektora wejściowego od `from` włącznie do `to` włącznie.

- `virtual ELEMENT_OUT initialize_element_out(const ELEMENT_IN &element_in) = 0`

  Inicjalizuje element wyjściowy na podstawie elementu wejściowego.

- `virtual void operator() (const ELEMENT_IN &element_in, ELEMENT_OUT &element_out) = 0`

  Wykonuje *Operację* na elemencie wyjściowym w miejscu, na podstawie elementu wejściowego.

- `virtual ELEMENT_OUT operator() (const ELEMENT_IN &element_in)`

  Wykonuje *Operację* na elemencie wejściowym i zwraca element wyjściowy.

Klasa `ops::OperationSqToSq`

`ops::OperationSqToSq` jest abstrakcyjnym funktorem, reprezentującym *Operację*, przekształcającą jeden wektor *Sekwencji* w inny wektor *Sekwencji*. Klasa ta rozszerza publicznie klasę `ops::OperationVectorToVector<Sq, Sequence, Sq, Sequence>` z odpowiednimi parametrami wzorca.

Parametry wzorca

- `INTERNAL_IN` – typ `InternalType` wektora wejściowego.
- `INTERNAL_OUT` – typ `InternalType` wektora wyjściowego.

Metody

Klasa dziedziczy po klasie bazowej, zapewniając ponadto domyślne implementacje dla metod.

```
Sq<INTERNAL_OUT> initialize_vector_out(const Sq<INTERNAL_IN> &sq_in, LenSq from, LenSq to)
Sequence<INTERNAL_OUT> initialize_element_out(const Sequence<INTERNAL_IN> &sequence_in)
```

- `virtual Alphabet map_alphabet(const Alphabet &alphabet_in) const`

  Mapuje *Alfabet* obiektu wejściowego na *Alfabet* obiektu wyjściowego.

Funkcja `sqapply`

Jest to kluczowa funkcja z punktu widzenia funkcjonalności pakietu. Służy ona aplikowaniu *Operacji* do wektorów.

Są one parametryzowane przez parametry wzorca analogiczne do parametrów `ops::OperationVectorToVector`.

1. ```
   VECTOR_OUT sqapply(const VECTOR_IN &vector_in,
                      ops::OperationVectorToVector<VECTOR_IN, ELEMENT_IN, VECTOR_OUT,
       ELEMENT_OUT> &operation)
   ```

2. ```
   VECTOR_OUT sqapply(const VECTOR_IN &vector_in,
                      ops::OperationVectorToVector<VECTOR_IN, ELEMENT_IN, VECTOR_OUT,
       ELEMENT_OUT> &&operation)
   ```
   ```

3. ```
   VECTOR_OUT sqapply(const VECTOR_IN &vector_in,
                      ops::OperationVectorToVector<VECTOR_IN, ELEMENT_IN, VECTOR_OUT,
       ELEMENT_OUT> &operation,
                      const LenSq from,
                      const LenSq to)
   ```

4. ```
   VECTOR_OUT sqapply(const VECTOR_IN &vector_in,
                      ops::OperationVectorToVector<VECTOR_IN, ELEMENT_IN, VECTOR_OUT,
       ELEMENT_OUT> &&operation,
                      const LenSq from,
                      const LenSq to)
   ```

   1), 2) Funkcja aplikuje *Operację* do każdego elementu wektora wejściowego.

   3), 4) Funkcja aplikuje *Operację* do każdego elementu wektora wejściowego od `from` włacznie do `to` wyłącznie.

Zaimplementowane *Operacje*

W bibliotece po klasach `ops::OperationVectorToVector` oraz `ops::OperationSqToSq` dziedziczy szereg *Operacji*, odpowiadającej poszczególnym funkcjonalnościom pakietu. Są one tutaj zaprezentowane pokrótce – szczegóły ich działania opisane są przy odpowiadających funkcjach na poziomie R. Ponadto dla każdej z tych klas istnieją odpowiadające im funkcje o analogicznych nazwach (dla *Operacji* `OperationXxxxYyyyZzzz` funkcja nazywa się `xxxx_yyyy_zzzz`), które na wejściu przyjmują odpowiedni wektor i aplikują do niego odpowiednią *Operację*. Funkcje te mają też przeciążone wersje, gdzie na wejściu przyjmowany jest tylko pojedynczy element i pojedynczy element jest zwracany.

- `OperationApplyRFunction` – analogia dla `sqapply` na poziomie R.
- `OperationComplement` – analogia dla `complement` na poziomie R.
- `OperationFindInvalidLetters` – analogia dla `find_invalid_letters` na poziomie R.
- `OperationHas` – analogia dla `%has%` na poziomie R.
- `OperationRandomSq` – analogia dla `random_sq` na poziomie R.
- `OperationRemoveAmbiguous` – analogia dla `remove_ambiguous` na poziomie R.
- `OperationRemoveNA` – analogia dla `remove_NA` na poziomie R.
- `OperationReverse` – analogia dla `reverse` na poziomie R.
- `OperationSubstituteLetters` – analogia dla `substitute_letters` na poziomie R.
- `OperationTranslate` – analogia dla `translate` na poziomie R.
- `OperationTypify` – analogia dla `typify` na poziomie R.

Kilka z funkcji nie posiada bezpośrednich odpowiedników na poziomie R. W poniższym fragmencie są opisane bardziej szczegółowo:

1. `OperationBite` oraz `OperationSkip` – pierwsza z tych funkcji działa tak jak `bite` na poziomie R, z różnicą co do indeksowania. Do tej funkcji można przekazać wektor wyłącznie nieujemnych indeksów. Indeksacja jest też przesunięta o 1 w dół względem R – wynika to z faktu, że w C++ występuje indeksacja od 0 w przeciwieństwie do R. Jeżeli użytkownik chce skorzystać z funkcjonalności negatywnej indeksacji jak w R, należy użyć `OperationSkip` z podaniem odpowiednich indeksów z odwróconym znakiem i pomniejszonych o 1.

2. `OperationPack` oraz `OperationUnpack` – te *Operacje* są odpowiedzialne za bitowe *Pakowanie* oraz *Rozpakowywanie* i nie są dostępne bezpośrednio z poziomu R. Są jednak wywoływane pośrednio np. podczas konstruowania wektora *Sekwencji* (`sq`) czy podczas wyświetlania go na ekran (`print.sq`).

3. `OperationRemoveOnCondition` – *Operacja* ta jest uogólnieniem działania `remove_ambiguous` oraz `remove_NA`. Wektor obiektów `sq`, na którym jest ona wykonywana, jest pozbawiony albo elementów w *Sekwencjach* spełniających określony warunek (bycie wartością `NA` przy `remove_NA` bądź bycie *Literą Wieloznaczną* przy `remove_ambiguous`) albo *Sekwencje* posiadające co najmniej jeden element spełniajacy ten warunek są zupełnie usuwane.

Funkcje `io::sample_fasta`, `io::read_fasta` oraz `io::write_fasta`

Funkcja `io::sample_fasta` służy do odczytania *Alfabetu Sekwencji* z danego pliku i zwraca *Alfabet*.

Funkcja `io::read_fasta` odczytuje *Sekwencje* z podanego pliku oraz zwraca obiekt `internal::NamedSqibble<INTERNAL>` z danymi *Sekwencjami* oraz ich nazwami.

Funkcja `io::write_fasta` zapisuje podane *Sekwencje* do pliku wraz z podanymi nazwami pod podaną ścieżką.

Funkcje `io::read_fasta` oraz `io::write_fasta` wymagają podania parametru typu `INTERNAL` będącego klasą spełniającą wymogi `InternalType`, która odpowiada parametrowi `INTERNAL` zapisywanego bądź odczytywanego obiektu `Sq`.

1. ```
   Alphabet io::sample_fasta(const std::string &file_name,
                             const LenSq sample_size = constants::BUFF_SIZE,
                             const Letter &NA_letter = constants::DEFAULT_NA_LETTER,
                             const bool ignore_case = constants::DEFAULT_IGNORE_CASE)
   ```

2. ```
   internal::NamedSqibble<INTERNAL> io::read_fasta(const std::string &file_name,
                                                   const Alphabet &alphabet)
   ```

3. ```
   void io::write_fasta(const Sq<INTERNAL> &sq,
                        const std::vector<std::string> &names,
                        const std::string &file_name,
                        const unsigned int &width)
   ```

Parametry:

- `file_name` – pełna ścieżka do pliku wejściowego z *Sekwencjami* w formacie *FASTA* bądź wyjściowego.
- `sample_size` – rozmiar próbkowania, liczba pierwszych znaków z pliku, na podstawie którego ma być zgadnięty *Alfabet*.
- `NA_letter` – *Litera* reprezentująca brak danych.
- `ignore_case` – wartość logiczna, określająca czy wielkość liter ma znaczenie.
- `alphabet` – *Alfabet Sekwencji* z odczytywanego pliku.
- `sq` – wektor zapisywanych *Sekwencji*.
- `names` – wektor nazw zapisywanych *Sekwencji*.
- `width` – maksymalna liczba znaków w linii wyjściowego pliku.

Działanie funkcji jest analogiczne do funkcji `read_fasta` i `write_fasta` w interfejsie `R`, przy czym na poziomie `C++` funkcja `read_fasta` wymaga jawnego podania *Alfabetu*, co może wymagać ręcznego wywołania funkcji *sample_fasta*.

Funkcja `find_motifs`

`find_motifs` jest funkcją odpowiadającą funkcji o tej samej nazwie z *Wysokopoziomowego API*. Wymaga podania parametru typu `INTERNAL` będącego klasą spełniającą wymogi `InternalType`, która odpowiada parametrowi `INTERNAL` przekazywanego obiektu `Sq` oraz zwracanych motywów.

```
internal::MotifFrame<INTERNAL> find_motifs(const Sq<INTERNAL> &sq,
                                           const std::vector<std::string>& names,
                                           const std::vector<std::string>& motifs)
```

Parametry:

- `sq` – wektor przeszukiwanych *Sekwencji*.
- `names` – wektor nazw przeszukiwanych *Sekwencji*.
- `motifs` – wektor poszukiwanych *Motywów*.

Funkcja zwraca znalezione motywy razem ze szczegółowymi informacjami o ich lokalizacji wewnątrz wejściowych *Sekwencji*.
```

Funkcje integracji

Poniższe funkcje służą do opakowywania i rozpakowywania obiektów opisywanych w tym dokumencie i zaimplementowanych w bibliotece w odpowiednie typy z biblioteki `Rcpp`, aby umożliwić przekazywanie tych obiektów pomiędzy R a C++.

Importowanie z R:

1. ```
   Alphabet import_alphabet_from_R(const Rcpp::StringVector &letters,
                                   const Letter &NA_letter = constants::DEFAULT_NA_LETTER,
                                   const bool &ignore_case = constants::DEFAULT_IGNORE_CASE)
   ```

2. ```
   Sq<RCPP_IT> import_sq_from_R(const Rcpp::List &sq,
                                const Letter &NA_letter = constants::DEFAULT_NA_LETTER)
   ```

3. ```
   ProtoSq<RCPP_IT, PROTO> import_proto_sq_from_R(
           const typename ProtoSq<RCPP_IT, PROTO>::ContentStorageType &proto,
           const Rcpp::StringVector &alphabet,
           const Letter &NA_letter = constants::DEFAULT_NA_LETTER,
           const bool &ignore_case = constants::DEFAULT_IGNORE_CASE)
   ```

1) Importuje *Alfabet*.
2) Importuje obiekt *Sq*.
3) Importuje obiekt *ProtoSq* z określonym `ProtoType`.

Eksportowanie do R:

1. ```
   Rcpp::StringVector export_to_R(const Alphabet &alphabet)
   ```

2. ```
   Rcpp::List export_to_R(const Sq<RCPP_IT> &sq))
   ```

3. ```
   template<typename INTERNAL, typename PROTO>
   typename ProtoSq<INTERNAL, PROTO>::ContentStorageType export_to_R(
           const ProtoSq<INTERNAL, PROTO> &proto_sq))
   ```

4. ```
   Rcpp::DataFrame export_to_R(const internal::NamedSqibble<RCPP_IT> &sqibble))
   ```

5. ```
   Rcpp::DataFrame export_to_R(const internal::MotifFrame<RCPP_IT> &found_motifs)
   ```

1–5) Eksportuje odpowiednie obiekty C++ z biblioteki do odpowiednich typów `Rcpp`, które mogą być przekazane do *Wysokopoziomowego API*.

Parametry:

- `sq` – obiekt klasy `sq` posiadający poprawny atrybut `alphabet`
- `proto` – obiekt typu odpowiedniego do skonstruowania `ProtoSq`
- `letters` – wektor *Liter* posiadający atrybut `type` będący poprawną nazwą *Typu Alfabetu*
- `NA_letter` – *Litera* reprezentująca brak danych
- `ignore_case` – wartość logiczna określająca, czy różnice w rozmiarze *Liter* powinny być ignorowane

Konwersje R-owych wektorów:

1. ```
   std::vector<std::string> convert_string_vector(const Rcpp::StringVector &vector)
   ```

2. ```
   std::string convert_to_scalar(const Rcpp::StringVector &vector,
                                 const unsigned int index = 0)
   int convert_to_scalar(const Rcpp::IntegerVector &vector,
                         const unsigned int index = 0)
   bool convert_to_scalar(const Rcpp::LogicalVector &vector,
                          const unsigned int index = 0)
   ```

3. ```
   LenSq convert_sample_size(const Rcpp::NumericVector &sample_size)
   ```
   ```

1) Importuje wektor napisów.
2) Importuje pojedynczą wartość, ekstrahując element wektora znajdujący się na wskazanej przez indeks pozycji, domyślnie zerowej (pierwszej w terminach R-owych).
3) Wyciąga pierwszą wartość wektora oraz zamienia ewentualną wartość `Inf` na `R_XLEN_T_MAX`.

Parametry:

- `vector` – wektor R-owy do zaimportowania
- `index` – indeks wyciąganego elementu
- `sample_size` – wektor długości 1 z rozmiarem próbki (potencjalnie o wartości `Inf`)

Konwersje C++-owych wektorów:

1. ```
   Rcpp::StringVector convert_string_vector(const std::vector<std::string> &vector)
   ```

1) Importuje wektor napisów.

Parametry:

- `vector` – C++-owy wektor napisów

Pozostałe funkcje użytkowe

Obsługa ostrzeżeń:

1. ```
   void handle_warning_message(const std::string &message,
                               const constants::WarningLevel &warning_level)
   ```

2. ```
   void handle_warning_message(const std::string &message,
                               const std::string &warning_level)
   ```

1–2) Obsługuje ostrzeżenie w konsoli R, wyświetlając podany komunikat na podanym poziomie.

Parametry:

- `message` – wiadomość dla użytkownika z informacją o sytuacji wyjątkowej
- `warning_level` – poziom ostrzeżenia w formie wartości typu wyliczeniowego albo nazwy poziomu

Odczytywanie alfabetu *Sekwencji*:

1. ```
   template<typename INTERNAL>
   Alphabet obtain_alphabet(
       const typename TypeBinder<INTERNAL, STRING_PT>::ProtoSqListConstructorType &x,
       const LenSq sample_size,
       const Letter &NA_letter = constants::DEFAULT_NA_LETTER,
       const bool ignore_case = constants::DEFAULT_IGNORE_CASE)
   ```

1) Uzyskuje rzeczywisty *Alfabet* na podstawie przekazanych *Niespakowanych Sekwencji*.

Parametry:

- `x` – lista niespakowanych *Sekwencji*
- `sample_size` – liczba pierwszych znaków *Sekwencji*, na podstawie których ma być określony *Alfabet*
- `NA_letter` – *Litera* reprezentująca brak danych
- `ignore_case` – wartość logiczna określająca, czy różnice w rozmiarze *Liter* mają być ignorowane

Konwersje mapa–wektor:

1. ```
   template<typename T>
   std::vector<T> convert_set_to_vector(const std::set<T> &set)
   ```
   ```

2. `std::vector<Letter> convert_map_to_vector(`
   `const std::unordered_map<LetterValue, const Letter> &value_to_letter)`

1) Konwertuje podany zbiór na wektor o kolejności zgodnej z kolejnością iteratora po zbiorze.
2) Konwertuje mapę (wartość liczbowa odpowiadająca *Literze*, *Litera*) na wektor *Liter* zgodny z kolejnością wartości liczbowych.

Parametry:

- `set` – zbiór dowolnych elementów
- `value_to_letter` – mapa par (wartość liczbowa odpowiadająca *Literze*, *Litera*)

4 Instrukcja użytkowania

W poniższej sekcji przedstawiono podstawowe sposoby korzystania z Systemu.

Wykorzystywanie pakietu R

Aby korzystać z pakietu, należy wprowadzić do konsoli nazwę funkcji poprzedzoną nazwą pakietu i operatorem zakresu '::', np.:

```
tidysq::sq(c("ATGATAGCTAGTG", "ATCGATAGC", "CCCTAGTGA")) -> sequences
tidysq::reverse(sequences)
tidysq::bite(sequences, 2:4)
```

Alternatywą jest załączanie przestrzeni nazw pakietu do ścieżki przeszukiwań za pomocą jednorazowego wywołania polecenia 'library':

```
library(tidysq)

sq(c("ATGATAGCTAGTG", "ATCGATAGC", "CCCTAGTGA")) -> sequences
reverse(sequences)
bite(sequences, 2:4)
```

Wprowadzenie w podstawowe funkcjonalności pakietu dla nowych użytkowników można znaleźć w winiecie pod adresem: `http://biogenies.info/tidysq/articles/quick-start.html`. Można też uzyskać do niej dostęp z poziomu środowiska `RStudio` wywołując polecenie

```
vignette("quick-start", "tidysq")
```

Pomoc na temat używania pakietu, struktury obiektów oraz szczegółową dokumentację funkcji można znaleźć wywołując w konsoli '?topic', gdzie 'topic' oznacza temat, na jaki chce się uzyskać pomoc.

Załączenie biblioteki dynamicznej C++

Aby załączyć bibliotekę we własnym pakiecie, należy wykonać następujące kroki:

1. Dołączyć bibliotekę i pakiet `Rcpp`. Najprostszy sposób to zainstalowanie pakietu `usethis` oraz wywołanie `usethis::use_rcpp()` i postępowanie zgodnie z krokami podanymi w konsoli. Szczegółowe informacje na temat dołączania `Rcpp` można znaleźć pod linkiem: `http://dirk.eddelbuettel.com/code/rcpp/Rcpp-package.pdf`.
2. Dodać `tidysq` do sekcji `Imports` oraz `LinkingTo` w pliku *DESCRIPTION*.
3. W przypadku korzystania z pakietu `roxygen2`, należy w dokumentacji dodać rejestrację biblioteki dynamicznej przy rejestracji biblioteki `Rcpp`:

   ```
   #' @useDynLib tidysq, .registration = TRUE
   ```

   W przypadku niekorzystania z pakietu `roxygen2`, należy dodać następującą linię w pliku *NAMESPACE*:

   ```
   useDynLib(tidysq, .registration = TRUE)
   ```

Aby skorzystać z biblioteki w danym pliku źródłowym, należy w nim dodać kod:

```
// [[Rcpp::depends(tidysq)]]
#include <tidysq.h>
```

Od tego miejsca w pliku źródłowym można korzystać z przestrzeni nazw `tidysq` i funkcji i obiektów w niej dostępnych.