Université de Rennes 1

Master 2 Bio-informatique (BIS)

UE Algorithmique des séquences

2021 - 2022

Jacky AME et Charles BROTTIER

RAPPORT DÉVELOPPEUR

Représentation compressée et indexation d'un ensemble de k-mers d'un jeu de données de séquençage

Responsables d'UE :

Claire Lemaitre et Pierre Peterlongo

Sommaire

L'objet de ce rapport est d'expliquer la structure et le fonctionnement du programme SequencesToSPSS.py, que nous avons écrit en langage Python3 dans le cadre notre unité d'Algorithmique des séquences.

1 Structure du programme

1.1 Vue d'ensemble des modules

La figure 1 montre l'arborescence des différents modules qu'utilise le programme.

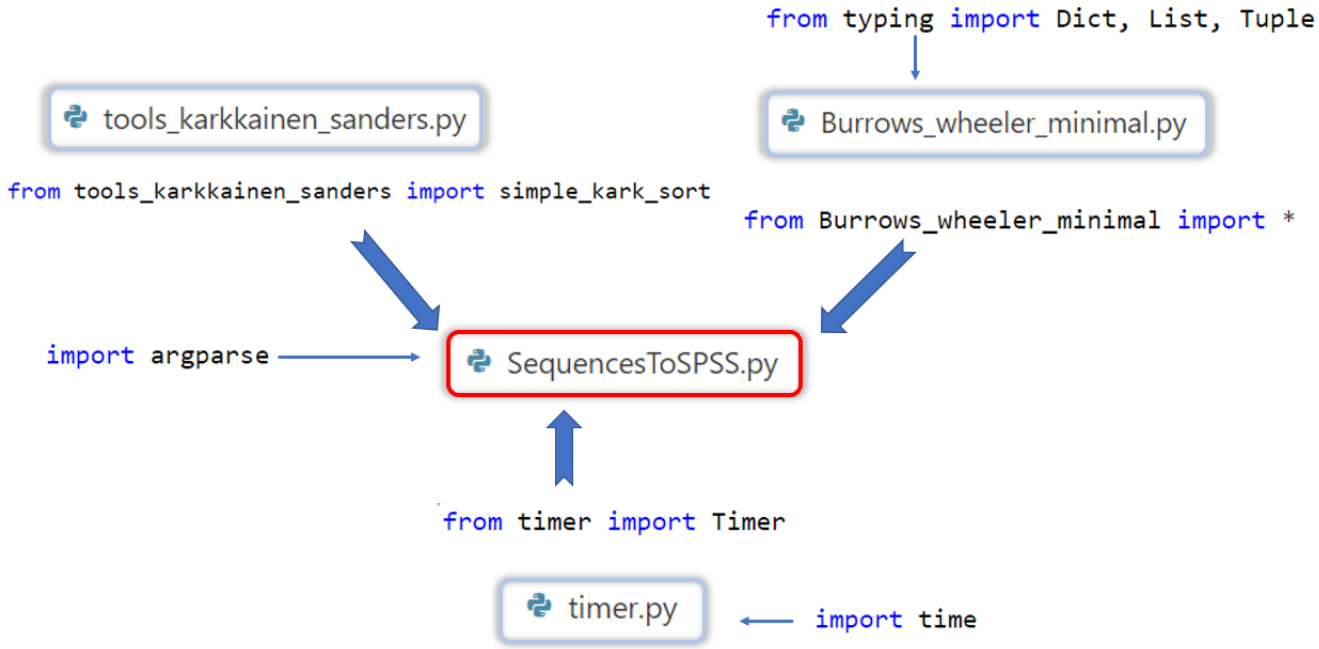


Figure 1 : Structuration du programme SequencesToSPSS.py. Les modules externes importés (timer.py, Burrows_wheeler_minimal.py et tools_krakkainen_sanders.py) sont en encadrés en bleu clair.

Le module timer.py sert à afficher, sur la sortie standard, les temps d'exécution des fonctions principales. Les modules tool_karkkainen_sanders.py et Burrows_Wheeler_minimal.py sont utilisés de concert pour l'indexation.

2 Description des fonctions

2.1 Fonctions ”outils”

```
def validate_dna_seq(seq: str) -> bool:
```

Prend une séquence ADN en format texte (type str) en argument, et renvoie un booléen : True si la séquence n’est pas vide **et** ne contient pas d’autres caractères que ’ACGT’, False sinon.

```
def rev_comp(seq: str) -> str:
```

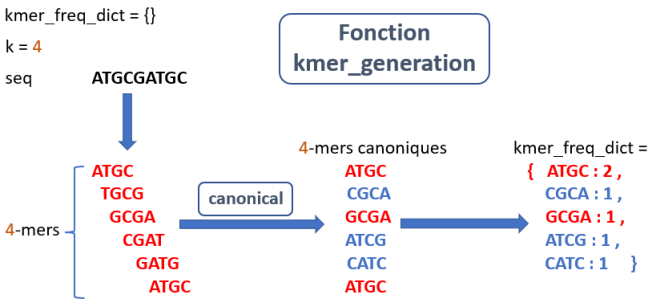
Prend une séquence ADN (str) en argument, et renvoie sa complémentaire inverse. Par exemple, la fonction avec pour argument ’ATCGAC’ renvoie ”GTCGAT”.

```
def canonical(kmer: str) -> str:
```

Prend une séquence texte (objet str) en argument, et renvoie sa forme canonique : celle qui vient la première, dans l’ordre lexicographique, entre la séquence d’origine et sa complémentaire inverse. Par exemple, le k-mer ’GAT’ ayant pour complémentaire inverse ’ATC’, qui est le premier (le plus ”petit”) lexicographiquement, la fonction **canonical()**, avec ”GAT” pour argument, renverra ”ATC”. Pour cela, elle utilise la fonction intégrée **min()**, qui renvoie l’objet à la valeur la plus basse.

```
def kmer_generation(seq: str, kmer_freq_dict: dict, k_size: int) -> dict:
```

Prend 3 arguments : une séquence ADN de type str (seq) ; un dictionnaire vide cible (kmer_freq_dict) destiné à contenir des k-mers (clés) associés à leurs fréquences (valeurs) ; et enfin la taille voulue des k-mers (k_size). Pour la séquence donnée, cette fonction renvoie ainsi un dictionnaire de k-mers **canoniques** (faisant appel à la fonction **canonical()**), non-redondants, chacun associé à sa fréquence.



2.2 Fonctions de chargement de données

```
def fasta_to_kmer(fasta_file: str, k_size: int, solid_t: int) -> set:
```

Prend en argument un fichier de séquences ADN (lectures) en format FASTA, la taille voulue des k-mers (k_size) et le seuil de solidité souhaité (solid_t). Selon ces 3 arguments et via la fonction **kmer_generation()**, **fasta_to_kmer()** construit un set de k-mers canoniques dont la fréquence est supérieure ou égale au seuil de solidité.

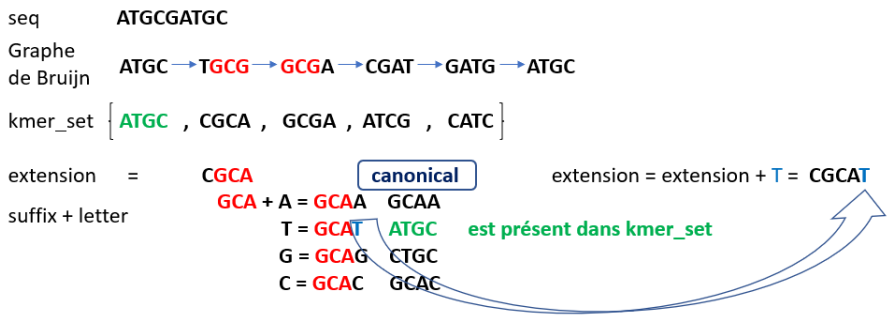
```
def fasta_simple_extract(fasta_file: str) -> str:
```

À partir d'un fichier FASTA passé en argument, cette fonction renvoie la concaténation directe de toutes les lectures qu'il contient.

2.3 Fonctions de construction de la SPSS

```
def kmer_forward_extension(extension: str, kmer_set: set, k_size: int):
```

Prend 3 arguments : un k-mer à étendre (extension, str), le set de k-mers canoniques (set) dont il est issu (kmer_set, set), et la taille des k-mers (k_size, int). Elle renvoie, d'une part, l'extension dans le sens *forward* ("à droite") de ce k-mer, à partir des k-mers canoniques présents dans le set, et, d'autre part, un set contenant les k-mers non-utilisés. Le principe d'extension des k-mers est illustré ci-dessous.




```
def unitig_generation(start: str, kmer_set: set, k_size: int):
```

Sur la base de la fonction précédente, celle-ci étend le k-mer fourni en argument en sens *forward* et *reverse* ("à gauche"). Il est considéré qu'étendre "à gauche" revient à étendre "à droite" le complémentaire inverse du k-mer considéré. Ainsi, ce dernier d'abord étendu à droite, l'extension produite est traitée par la fonction **rev_comp()**, puis étendue à droite à son tour. Est ainsi renvoyé un unitig maximal.

```
def all_unitigs(kmer_set: set, k_size: int) -> set:
```

À partir de l'ensemble des k-mers présents dans le set fourni en argument et de leur taille spécifiée, sont retournés tous les unitigs maximaux grâce à la fonction **unitig_generation()**. Les unitigs sont ainsi générés de façon gloutonne jusqu'à ce que le set soit vide. À chaque itération, le k-mer de départ est tiré au hasard par la méthode **.pop()**.

```
def unitig_compression(unitigs: set, ref_dict: dict, anchor_len: int) -> set:
```

Prend un set d'unitigs maximaux en argument et les compresse en recherchant entre eux des chevauchements suffixe-préfixe (”ancres”), avant de les ajouter à un nouveau set, renvoyé par la fonction. Le dictionnaire utilisé en argument sert à marquer les unitigs utilisés (déjà compressés) d'après la taille de l'ancre spécifiée (fixe).

```
def spss_generation(unitigs: set, k_size: int) -> str:
```

Prend un set d'unitigs maximaux en argument, ainsi que la taille des k-mers utilisée jusqu'ici. Cette fonction, de façon gloutonne et itérative, applique **unitig_compression()** en commençant par la taille d'ancre maximale possible (|k_size - 2|). À chaque itération, la taille de l'ancre est réduite de 1, afin de compresser progressivement tous les unitigs qui peuvent l'être sans ”perdre” de k-mers. Chaque séquence compressée est ainsi ajoutée à un set, et la fonction renvoie la concaténation de toutes les séquences compressées à l'issue de la dernière itération.

```
def spss_max_compression(spss: str) -> str:
```

Prend la SPSS générée précédemment en argument, et la compresse de façon maximale en remplaçant toute suite de caractères successifs identiques par le caractère en question, associé à son nombre d'occurrences. Par exemple, en passant la séquence 'AAAAATTTTGGGCC' en argument, la fonction renvoie 'A5T4G3C2'.

Note : Cette fonction est seulement utilisée à des fins de gain maximum d'espace de stockage, mais la séquence qu'elle renvoie n'est pas celle qui est utilisée pour les opérations de validation et d'indexation.

2.4 Fonctions de validation et d'indexation

```
def naive_fn_count(genome_kmers: set, spss: str) -> int:
def naive_fp_count(spss_kmers: set, genome: str) -> int:
```

Ces deux fonctions retournent respectivement le nombre de k-mers faux négatifs, c'est-à-dire tous ceux présents dans le génome de référence, mais pas dans la SPSS, et le nombre faux positifs, c'est-à-dire tous ceux présents dans la SPSS, mais pas dans le génome de référence. Pour cela, les k-mers et leurs inverse complémentaires sont recherchés. Chaque fonction prend en entrée la séquence à cribler, ainsi que l'ensemble des k-mers à y trouver (SPSS et k-mers du génome pour les faux négatifs, génome et k-mers de la SPSS pour les faux positifs). Ces fonctions utilisent simplement l'opérateur "not in" de python pour tester la présence/absence des k-mers, et renvoient le nombre d'occurrences comptées dans chaque cas.

```
def bwt_indexation(string: str):
```

Cette fonction indexe "virtuellement" la séquence fournie en argument. Pour cela, elle renvoie trois objets : la transformée de Burrows-Wheeler de la séquence (bwt, type str), le nombre d'occurrences de chaque caractère qu'elle contient (n_dict, un dictionnaire), et les rangs de chaque caractère à leurs positions respectives (ranks, une liste). Le tout forme une structure équivalente à un index (le *FM-index*) et permet une recherche accélérée de sous-séquences. L'indexation n'étant pas le sujet premier du présent rapport, sont reprises les fonctions des scripts tools_karkkainen_sanders.py et de Burrows_Wheeler_minimal.py afin de générer les objets mentionnés.

```
def indexed_fn_count(spss: str, genome_kmers: set) -> int:
def indexed_fp_count(genome: str, spss_kmers: set) -> int:
```

Équivalentes des fonctions **naive_fn_count()** et **naive_fp_count**, celles-ci utilisent en revanche le FM-index mis en place par la fonction **bwt_indexation()**, afin de réduire le temps de recherche des k-mers.