Rapport mise à jour projet 2 MASB : *YAYA-OYE AKIBOU Abdel Moumin*

Le but de ce projet est d'identifier à quel virus ressemble le pathogène présent dans l'échantillon pulmonaire. En cherchant si il y des séquences semblables au virus tels que : grippe, rhinovirus, HIV et coronavirus. Il est donc question d'aligner les reads sur chacun de ces génomes et d'identifier sur quel génome, le plus de reads sont alignés. L'alignement consiste en une stratégie *seed and extend*. ***Seed*** : pour une extraction de k-mers du read puis recherche, sans tolérer d'erreur, dans le génome d'intérêt. ***Extend*** : une fois une graine (k-mer) trouvée, on aligne le read en entier sur cette région du génome.

Structures de données utilisées :

- ***Table de suffixe***: La table de suffixes est une table qui stocke les indices des suffixes triés dans l'ordre lexicographique. C'est-à-dire, pour une séquence de longueur n, la table de suffixes contient les positions de tous les suffixes de longueur n dans la séquence triés dans l'ordre lexicographique. La table de suffixes est utilisée pour la recherche de sous-chaînes dans une séquence en utilisant l'algorithme de recherche dichotomique et elle utilise une quantité d'espace raisonnable (mémoire) pour stocker la table de suffixes par rapport à d'autres structures de données, car elle ne stocke que les indices des suffixes et non les suffixes eux-mêmes. Nous permet donc d'indexer un génome et ainsi de pouvoir identifier si un k-mer est présent ou non dans ce génome sans le parcourir en entier.

  Inconvénient :
  - La construction de la table de suffixes peut être coûteuse en temps et en espace pour de très grandes séquences.
  - L'utilisation de la table de suffixes pour des séquences très répétitives peut entraîner une consommation élevée d'espace.
- ***Set*** : Il est utilisé pour stocker des k-mers uniques extraits à partir des séquences dans le fichier fastq.gz. C'est une structure de données non ordonnée et mutable qui ne peut contenir que des éléments uniques. Dans notre cas, il est utilisé pour stocker les k-mers uniques extraits afin d'éviter les doublons, ce qui réduit la complexité du traitement de ces k-mers. Son implémentation est basée sur une table de hachage, ce qui signifie qu'elle peut être utilisée pour rechercher rapidement si un élément est présent ou non. Mais, elle n'est pas optimale pour la récupération d'éléments dans un ordre particulier, car elle n'est pas ordonnée. Étant donné qu'on a utilisé une table des suffixes, ce n'est pas idéal pour notre recherche, raison pour laquelle son utilisation dans ce projet est limitée à cette étape.

  Les avantages de l'utilisation d'un set sont :

  - ✓ Les éléments sont uniques, ce qui est utile lorsque l'on ne veut pas de doublons dans la liste des k-mers extraits.
  - ✓ La recherche de la présence d'un élément est très efficace, ce qui est utile pour vérifier si un kmer est déjà présent dans la liste.

  Les inconvénients de l'utilisation d'un set sont :

  - ✓ Il n'est pas ordonné, donc il n'y a pas de garantie sur l'ordre dans lequel les éléments seront stockés dans la structure de données.
  - ✓ Il n'est pas optimisé pour récupérer des éléments dans un ordre particulier.

- **Dictionnaire** : Il a été choisi car il permet de stocker des éléments avec une clé et une valeur associée. Dans notre cas, la clé est le k-mer et la valeur est une liste des positions dans le génome où ce k-mer apparaît. Cette structure de données est idéale pour une recherche rapide de la position d'un k-mer dans le génome.

  Les avantages d'utiliser un dictionnaire sont :

  - ✓ La recherche d'un élément est rapide car elle se fait en temps constant en moyenne.
  - ✓ Les clés peuvent être de n'importe quel type immuable, y compris des chaînes de caractères, des nombres et des tuples.

  Les inconvénients sont :

  - ✓ Les dictionnaires prennent plus de mémoire que les listes pour stocker les mêmes données.
  - ✓ Les dictionnaires ne sont pas ordonnés, donc l'ordre des éléments n'est pas garanti.

  Dans notre cas, les avantages de la rapidité de la recherche l'emportent sur l'inconvénient de la mémoire supplémentaire nécessaire pour stocker les positions des k-mers.

- **Liste** : est utilisée pour stocker les séquences extraites du fichier FASTA. La raison d'avoir choisi une liste pour stocker ces séquences est que les listes sont des structures de données simples et efficaces pour stocker des collections d'objets. De plus, dans notre cas, nous avons besoin de stocker des objets de taille variable (les séquences peuvent avoir des longueurs différentes), ce qui est une caractéristique naturelle des listes.

  Les avantages d'utiliser une liste dans ce cas sont les suivants :

  - ✓ Elles permettent un accès rapide aux éléments individuels en utilisant des index entiers.
  - ✓ Elles sont dynamiques, ce qui signifie que leur taille peut être modifiée pendant l'exécution du programme.
  - ✓ Elles peuvent contenir des objets de taille variable.

  Cependant, il existe également des inconvénients à l'utilisation de listes. :

  - ✓ L'accès à des éléments de la liste en utilisant un index entier peut être lent pour les grandes listes.
  - ✓ Les opérations d'insertion et de suppression dans une liste peuvent être lentes pour les grandes listes, car cela nécessite de déplacer les éléments dans la liste.
  - ✓ Les listes peuvent prendre beaucoup de mémoire si elles contiennent de nombreux objets.

<u>Optimisation pour de grands jeux de données :</u>

- Utilisation du module re pour les expressions régulières pour valider les k-mers et éviter les erreurs de séquençage et en même temps pour faire office de filtrage des N.
- Utilisation des compréhensions de liste python pour éviter plusieurs boucle for.

Après avoir longuement consulté le code de mon programmes, et effectuer quelques recherches sur la manière dont l'optimisation pourrait se faire pour de grand jeux de donnée, il y a entre autre quelques approches possibles que j'ai envisagés mais que je n'ai pas pris le risque d'essayer en raison du temps et des ressources dont je dispose.

- Utilisation des ensembles (set) plutôt que des listes pour stocker les k-mers extraits étant donné que les ensembles ont une complexité de recherche beaucoup plus.
- Éviter de copier les séquences dans la mémoire chaque fois qu'une opération doit être effectuée dessus. Par exemple, dans la fonction ***generate_suffix_table***, la variable seq est une copie de la séquence en entrée, mais elle n'a pas besoin d'être stockée en mémoire. On pourrait essayer de concaténer le symbole de fin de chaîne de caractères ($) à la fin de la séquence en entrée pour obtenir la séquence de référence pour la table des suffixes.
- Utiliser des dictionnaires pour stocker les positions des k-mers dans le génome, au lieu de listes. Les dictionnaires ont une complexité de recherche beaucoup plus rapide .
- Ne pas parcourir le fichier FASTA entier à chaque fois que la fonction ***extract_sequences_from_fasta_file*** est appelée. La bibliothèque BioPython utilise une implémentation de générateur pour la lecture des séquences FASTA, ce qui permet de lire les séquences une à une, au lieu de toutes en même temps. On pourrait donc itérer sur l'objet SeqIO.parse(handle, "fasta") au lieu de stocker toutes les séquences dans une liste.
- Utiliser la bibliothèque parasail pour calculer en même temps le pourcentage d'identité au lieu de faire le calcul séparément.
- Utiliser des processus parallèles pour effectuer des tâches lourdes en calcul, comme l'extraction de k-mers et l'alignement de séquences. Cela permettra de distribuer la charge de travail sur plusieurs cœurs de processeur, ce qui peut considérablement accélérer l'exécution du programme (voir code multiprocessus).

Paramètre:

- Taille du k-mer: j'ai utilisé une taille des k-mers 11 comme par défaut pour blast qui se trouve être à mi-chemin entre une grande et petite taille, certes cela pourrait occasionner des problèmes de régions répètes mais au moins permet d'avoir des matchs pour tous les génomes normalement.

Sorties :

Pour **k = 11**

Reads du fichier SRR10971381_1.fastq.gz alignés sur le génome:

- HIV-1 :

Fichier de sortie : sortie_seq1_hiv_95%

Temps de calcul : 21272,86 secondes (soit 5 heures et 54 minutes) ➔
1930 reads alignés avec une identité > = 95%

- Grippe A :

Aucuns résultats obtenus pour une identité de 95%, avec une taille des k-mers de 11 .Je ne suis pas sur car, j'ai lancer sans nohup donc je ne sais la tâche s'est arreté entre temps mais avec un taille des k-mers de 5, il y a des sorties.

- Rhinovirus :

Fichier de sortie : sortie_seq_Rhinovirus1_95%.txt

Temps de calcul : 15787.48 secondes (soit 4 heures et 23 minutes) ➔ 257 reads alignés avec une identité > = 95%

- Alpha-coronavirus :

Reads du fichier SRR10971381_2.fastq.gz alignés sur le génome:

- HIV-1 :

Fichier de sortie : sortie_seq2_hiv_95%.txt

Temps de calcul : 0 reads alignés avec une identité > = 95%

- Grippe A : Pas pu lancer

- Rhinovirus :

Fichier de sortie : sortie_seq_Rhinovirus2_95%.txt

Temps de calcul : 31045.33 secondes (soit 8 heures et 37 minutes) ➔ 455 reads alignés avec une identité > = 95%

- Alpha-coronavirus :

Fichier de sortie : sortie_seq2_coronora_95%.txt

Temps de calcul : 33175.70 secondes (soit 9 heures et 12 minutes) ➔ 773 reads alignés avec une identité > = 95%

(Partie 2)

- Sars-Cov-2 :

Fichier de sortie : Pas aller au bout de la tâche, pour le moment, uniquement essayer avec corona.

Temps de calcul : Après 11h55, rien dans le fichier de sortie pour une taille de kmer de 11 et 95% de similarité pour le corona.