IFT2015 TP2 Qiang Ye (20139927) et Xiao Ju (1037977)

1. a) Veuillez regarder graph. py

b) Discuter des détails de votre implémentation en répondant aux questions suivantes:

• Quelle fonction de hachage avez-vous utilisé? et dans quelle mesure est-elle spécifique au problème?

Notre fonction de hachage utilisé comprend deux parties. La première partie est pour transformer la séquence de ADN en code de hachage. La deuxième partie est une fonction de compression pour adresser le codage dans la table hachage.

La fonction de code de hachage est spécifique au problème car elle prend une séquence ADN comme une clé et la transformer à un nombre entier unique. Plus spécifiquement, on transforme les quatre nucléotides en binaires différents : A = 00, T = 01, C = 10, G = 11. Pour une séquence ADN de n nucléotides, on aligne les deux bits de chaque nucléotide selon leurs positions pour créer un nombre binaire de 2n bits. Nous prenons le chiffre décimal correspondant comme le code de hachage final. Par exemple : une séquence ADN 'ATCGTA', sera représenté par '00' + '01' + '10' + '11' + '01' + '00' = 000110110100, code hachage est donc 436.

Il y aura de problème de collision si les séquences étudiées ne sont pas de même taille, par exemple, 'AAT' = '000001' et 'AAAT' = '00000001', les deux séquences vont donner le même chiffre décimal. Heureusement, dans notre TP, des k-mers pour créer le graph De Bruijn sont de même taille, alors, les chiffres seront uniques pour des k-mers différents.

La fonction de compression est une fonction de division pour passer des code hachage à des indices dans la table de hachage.

• Quelle stratégie d'adressage avez-vous utilisé?

La stratégie d'adressage utilisé est l'adresse ouvert avec sondage quadrique. L'utilisation de sondage quadrique est pour mieux distribuer les éléments et éviter de créer des 'cluster' dans la table.

c) Quel serait le principal gain d'utiliser une technique de hachage sensitif à la localité et de l'adressage linéaire pour ce problème?

L'utilisation de hachage sensitif à la localité et de l'adressage linéaire dans ce problème est plus efficace pour insérer des k-mers dans la table de hachage avec moins de collision, ce qui augmente aussi l'efficacité de la recherche des clés pour retrouver des k-mers.

2) a) Veuillez regarder main.ipynb

b) Bonus: Implanter une table de hachage plus rapide que celle de CPython (i.e. celle utilisée par set et dict) pour stocker des k-mers pour les opérations effectuées en 2a.

Notre table de hachage est aussi dynamique, mais il permet de définir la taille initiale de table, ce qui évite de 'resize' beaucoup de fois pour augmenter la taille de la table quand le nombre de k-mers est très grand. Par exemple, dans le problème de ce TP, on a plus que 5 millions de k-mers différents, si on initiale une table de petite taille, on aura besoin de 'resize' la table beaucoup de fois avant qu'elle puisse stocker tous les k-mers.

Le deuxième avantage de notre table de hachage est la fonction de hachage utilisée. Comme expliqué dans question 1 b), notre fonction de hachage est plus spécifique pour le problème des séquences ADN, elle donne des codes de hachage uniques pour des séquences différents, ce qui réduit aussi la collision dans la table.

3) a) Veuillez regarder main.ipynb.

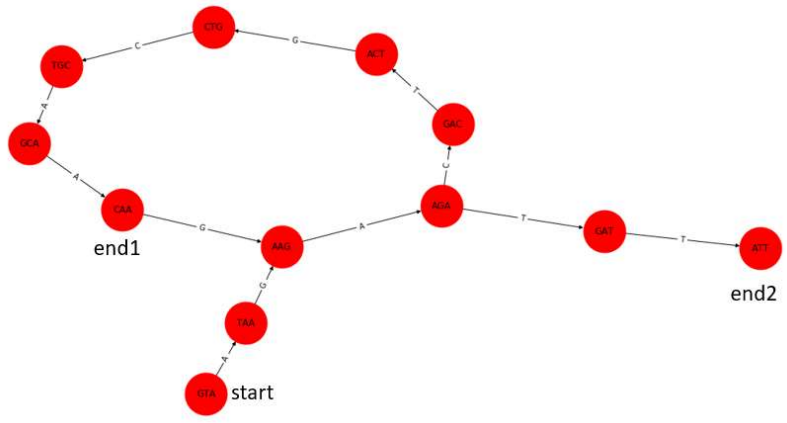
b) Dans votre rapport, répondez aux questions suivantes:

• Quelle stratégie de parcours avez-vous adoptée?

On utilise la stratégie de la recherche en profondeur (DFS). On commence par des nœuds qui n'ont pas de prédécesseurs (les nœuds 'start'), chercher les chemins pour aller aux nœuds accessibles qui n'ont pas de successeurs (les nœuds 'end'), s'il y'en a. Chaque chemin représente une séquence assemblée.

• Comment les boucles sont-elles gérées durant le parcours?

Lorsqu'il existe de boucle entre un nœuds 'start' et un nœuds 'end', il y a deux situations. On les montre dans la figure au-dessous.



La première est que la vraie séquence entre 'start' et 'end' passe par une partie de la boucle, mais fait pas un tour complet dans la boucle. Ce type de séquence peut être assemblé correctement par DFS, dans la figure, c'est le chemin direct entre 'start' et 'end2'.

La deuxième situation est que le séquence contenir la boucle au complet. C'est-à-dire le chemin commence par 'start', entre dans la boucle, fait au moins un tour dans la boucle, puis sort pour aller à 'end2'. Dans cette situation, le DFS vas couper la boucle à la position où il entre dans la boucle. On aura donc juste le chemin entre 'start' et 'end1', cette séquence obtenue peut être retrouvé dans la référence

de FASTA, mais elle manque une grande partie de séquence qui tourne dans la boucle et la partie connecte la boucle au 'end2'.

c) Quelle est l'influence du choix de k sur le résultat obtenu par le parcours?

Lorsque k est petit, on trouve plus de k-mer identiques (ou k-mer répétitive). Ce qui rend dans le graph, ce type de nœuds ont de dégrée élevée, et le parcours par ces nœuds donne des informations incorrectes.

Par conséquent, lorsque k-mer est long, il diminue le nombre de k-mers identiques, ce qui diminue le nombre de cycle dans le graph. Il augmente aussi la chance pour trouver des nœuds sans successeurs et des nœuds sans prédécesseurs, augmente la chance d'assembler correctement la séquence. Mais s'il y a des erreurs dans les données de FASTQ (ce qui arrive souvent), et k est trop haut, ils vont diminuer la chance pour trouver la vraie séquence.

4) a) Veuillez regarder main.ipynb

b) Peut-il exister des séquences qui ne peuvent pas être assemblée correctement dans la référence (avec k = 21)? Le cas échéant, en fournir un exemple.

Oui, il peut exister des séquences qui ne sont pas assemblées correctement. Comme expliqué dans la question 3b), lorsque la séquence contient des boucles, notre DFS ne peut pas l'assembler correctement. DFS va retourner le chemin direct entre 'start' et 'end2' sans contenir les autres nœuds dans la boucle.

Par exemple, dans la figure de question 3 b), la séquence assemblée pour 'start' et 'end2' est GTAAGATT. Si la vraie séquence comprend la boucle, elle doit être : GTAAGACTGCAAGATT, la partie rouge n'est pas assemblée correctement en raison de la coupe de cycle par DFS. Donc, dans la pratique, on aura besoin de l'assemblage secondaire pour corriger ce type de problème.

Un exemple dans FASTQ: une séquence de taille 77 :
'TATTATTACGAAGAAAAATACTCAAGCTTCTCACGGGACCTGATAGATCATCCGATGCGTTACATTCCCAAAATAGA'

c) Bonus: Proposer une structure de données appropriée pour effectuer une comparaison efficace des séquences assemblées avec celles de référence. La complexité doit être strictement inférieure à la méthode proposée en 4a.

str.find() method in python combines the advantages of two algorithm (Boyer Moore,Horspool)， it uses also bloom filter algorithm， make the str.find() method effective and fast .We have tried "sunday" algorithm， and the use of hash， unfortunately they are both less effective than python's method. The sunday algorithm and hash method used are showed below.

Another idea is to use the characteristic that DNA sequence only has four nucleotides ATCG. But we haven't found a good way to implement it yet.