Recherche de Motif

Algorithme naïf :

Cet algorithme étant très rapide et simple à implémenter, nous avons préféré commencer par celui-ci afin d'avoir rapidement un programme fonctionnel. Il parcours simplement le texte à analyser et vérifie si le morceau actuel correspond à la chaîne recherchée. Si c'est le cas, l'indice de la position dans le texte sera ajouté à la liste d'occurrence du mot recherché.

Une fois cet algorithme implémenté et testé, nous nous sommes lancé dans l'implémentation de l'algorithme de Boyer-Moore (décrit plus bas). Nous pouvions rapidement vérifier les résultats de celui-ci en les comparant à ceux de la recherche naïve.

Algorithme de Boyer-Moore :

Cette implémentation se fait en deux parties distinctes :

- Tout d'abord, On fait appel à la fonction *remplirBonSuffixe* qui se charge de remplir la table des bons suffixes du motif passé en paramètre. C'est elle qui va déterminer le nombre de lettres à shifter lors de la recherche de motif.

  Pour un motif m de taille n, on procède à chaque indice de 0 à n-1 de telle façon : La case d'indice x compris entre 0 et n-1 est égale à une valeur v. Concrètement, cela veut dire que nous avons trouvé le mot m[x-1,n-1] dans une séquence mais que m[x] n'apparaît pas dans la dite séquence à sa bonne place. Dans ce cas il faut se décaler de v indices sur la droite dans la séquence pour espérer trouver une occurrence du motif.

  L'algorithme fonctionne ainsi pour un motif m de longueur n passé en paramètre :

  Pour chaque sous-motif du mot (c'est à dire pour chaque m[i,n-1], pour i = n-1…0) :

  On doit chercher une occurrence du sous-motif mais qui ne doit pas être égal à m[i-1,n-1] à l'intérieur même du motif.

  Si on en trouve une alors on écrit dans *bonsuffixe*[i-1] le nombre de déplacement pour se rendre à cette occurrence à partir de i.

  Sinon *bonsuffixe*[i-1] vaut n moins la longueur de plus long bord du motif.

- L'algorithme principal *chercherMotif* fait appel à *remplirBonSuffixe* dans un premier temps. Puis on commence à chercher les occurrences du motif m dans la séquence s.

  i vaut 0 au début.

  On compare s[x+i] et m[x], x décroissant de n-1 à 0 (n étant la taille du motif).

  Dès que l'on a une différence de caractères pour un des x, alors i = i + *bonsuffixe*[x].

Si l'on trouve une occurrence du motif dans la séquence alors on ajoute son indice dans la liste des occurrences et i = i + *bonsuffixe*[i].

On recommence les comparaisons jusqu'à la fin de la séquence entrée en paramètre.

A la fin on retourne cette liste.

Utilisation du programme :

Pour tester ce programme simplement, il suffit d'exécuter le script bash 'rechercheMotif.bash' et de lui préciser :

- le chemin vers le fichier .fasta à analyser
- la taille N des motif à rechercher

Il est également possible de lui préciser des options de recherche (facultatives) :

- -r pour rechercher les mots de taille N ainsi que leur inverse
- -c pour rechercher les mots de taille N ainsi que leur complémentaire
- -rc pour rechercher les mots de taille N ainsi que leur inverse-complémentaire

Vous pouvez aussi lui demander d'effectuer la recherche avec l'algorithme naïf grâce à l'option

--naif

Détection de pré-micro-ARN

La détection de Pré-Micro-ARN est basée sur la programmation dynamique. Une classe *MaxParenthesage* se charge de compter le nombre maximal d'appariements d'une séquence génomique passée en constructeur, la fonction déclenchant l'algorithme remplie dynamiquement et simultanément trois tableaux :

- Un tableau comptant le nombre d'appariements successifs (Un Pré-Micro-ARN devant avoir des séquences d'appariements supérieures à trois).

- Un tableau comptant le nombre de nucléotides successifs ignorés (Un Pré-Micro-ARN ne pouvant en avoir plus de trois en suivant hors boucle interne).

- Un tableau comptant le nombre d'appariements au total, en tenant compte des deux tables ci-dessus.

Le remplissage se fait diagonalement de gauche à droite. Pour remplir la case *tab*[i][j], il faut faire le supérieur des cases *tab*[i+1][j], *tab*[i][j-1] et tab[i+1][j-1]. Si la valeur supérieure est *tab*[i+1][j-1] alors c'est que l'on trouve un appariement, la case *tab*[i][j] prend la valeur *tab*[i+1][j-1]+1, on affecte 0 à *tab_points*[i][j] et *tab_parents*[i][j] vaut *tab_parents*[i+1][j-1].

Sinon, c'est qu'on doit ignorer une des nucléotides :

On prend pour valeur de *tab*[i][j] la valeur maximale entre *tab*[i+1][j] et *tab*[i][j] et on met à jour les autres tables, *tab_parents*[i][j] vaut 0 et *tab_points*[i][j] vaut soit *tab_points*[i+1][j]+1 soit *tab_points*[i][j+1].

Cependant, il faut tenir compte assez souvent des valeurs situées dans les tables *tab_points* et *tab_parents* avant d'ajouter bêtement un. Par exemple, pour remplir tab[i][j], si la valeur maximale des trois possibilités est *tab*[i][j-1] mais que la valeur contenue dans t*ab_parents*[i][j-1] est inférieure à 3 (signifie qu'on à pas le bon compte d'appariements successifs) ou que celle contenue dans *tab_points*[i+1][j] est égale à 3 (signifie que l'on ne peut plus ignorer de nucléotide de ce côté), alors on prend comme valeur supérieure celle de *tab*[i+1][j] (cela veut dire que c'est la seule issue possible pour pouvoir espérer trouver un pré-Micro-ARN).

Une fois les tables remplient on est capable de dire si la séquence sur laquelle on a appliqué l'algorithme contient ou non un pré-Micro-ARN et on retourne les indices de début et fin.

Pour trouver tout les pré-Micro-ARN dans le chromosome 13, l'idée est de parcourir 100 par 100 le gène et d'appliquer la fonction précédente. Si l'on en trouve un, alors on ajoute ses coordonnées dans une liste que l'on renvoie à la fin. Nous avons décidé de renvoyer à chaque fois les plus gros pré-Micro-ARN lors du renvoi des coordonnée de la fonction *contientPreMicroARN*.

On peut accéder à l'intégralité de la liste avec la fonction *getAllPreMicroARN*. Afin de connaître les Pré-Micro-ARN contenant un Micro-ARN pouvant s'hybrider avec un ARN-Messager, on prend le complémentaire des 7 premiers nucléotides du messager et on applique la recherche de Boyer-Moore pour ce motif sur chaque élément de la liste renvoyée par *getAllPreMicroARN*. L'hybridation avec le messager est possible si l'indice renvoyé par la recherche de Boyer-Moore est compris entre 9 et 14.

Comment tester :

javac -d bin src/*/*.java

java -cp bin detection.DetectionPreMiRNA <fasta-ARNmessager>