Quentin Baert
Alexandre Verkyndt

AeA - TP 01

Recherche de motifs répétés dans un génome

Réalisation

Toutes les étapes de réalisation ont été effectuées. Il est donc possible de partir d'un fichier `fasta` et d'en tirer un *DotPlot*. Les résultats de l'algorithme de recherche de motif sont stockés dans un fichier `.dat`.

Deux algorithmes de recherche de motif ont été implémentés :

- l'algorithme naïf
- l'algorithme KMP

Sont fournis :

- les sources commentées du programme
- les tests des différentes classes et algorithmes implémentés
- la documentation du programme au format HTML
- un `jar` éxécutable qui permet d'appliquer l'algorithme de recherche de motif *KMP* sur un fichier `fasta`
- un script shell qui permet de passer d'un fichier `.dat` à un *DotPlot* au format `png` en utilisant `gnuplot`
- un dossier `generate` dans lequel se trouve un fichier `.dat` déjà généré ainsi que le *DotPlot* correspondant

Utilisation

Voici la marche d'execution à suivre :

- Pour générer un fichier `.dat` :

```
java -jar plot.jar <chemin> <code> <length>

Avec :
    <chemin> : chemin vers le fichier fasta qui contient la sequence ADN/ARN

    <code>   : un code entre 0 et 7 pour determiner les motifs à considérer
        0 : motif seul
        1 : motif seul + reverse
        2 : motif seul + complement
        3 : motif seul + reverse complement
        4 : motif seul + reverse + complement
        5 : motif seul + reverse + reverse complement
        6 : motif seul + complement + reverse complement
        7 : motif seul + reverse + complement + reverse complement

    <length> : taille des motifs a considérer
```

- Pour passer d'un fichier `.dat` à un *DotPlot* au format `png` :

```
./dotplot.sh <chemin-vers-le-fichier-dat>
```

Les fichiers `.dat` et `.png` ainsi générés le seront dans le répertoire courant.

Architecture

```
src
  |_ adn
  |_ algo
  |_ main
  |_ plot
test
  |_ adn
  |_ algo
```

- Le package `adn` contient toutes les classes permettant de représenter les motifs d'ADN ou d'ARN
- Le package `algo` contient les deux instances respectivent de l'algorithme de recherche naïf et de l'algorithme KMP ainsi que des classes qui représentent les entrées et sorties de ces algorithmes
- Le package `plot` contient la classe qui permet d'appliquer un algorithme à une entrée et d'obtenir une sortie écrite dans un fichier `.dat` par la suite
- le dossier `test` contient les tests des classes comprises dans les packages de mêmes noms

Algorithmes

Algorithme naïf

L'algorithme naïf ne possède aucune particularité dans son implémentation : une fenêtre de la taille du motif à chercher se déplace caractère par caractère sur toute la séquence et un test est effectué à chaque déplacement de la fenêtre. Si la fenêtre correspond au motif recherché, sa position est ajoutée à la sortie.
On peut toutefois remarquer l'utilisation d'une méthode `acceptFor()` qui determine si une fenêtre correspond au motif recherché ou à son complément, son reverse, ou reverse-complément en fonction de l'entrée sur laquelle s'applique l'algorithme. Le recours à cette méthode permet d'effectuer un seul parcours de séquence au prix d'un test un peu plus couteux à chaque déplacement de fenêtre.

Algorithme KMP

Pour l'algorithme KMP, nous avons aussi fait en sorte de ne faire qu'un seul parcours global de la sequence.
Pour cela, les pretraitement de toutes les formes du motif demandées par l'entrée sont calculés avant de commencer le parcours et une liste tient à jour les indexes des différentes fenêtre (une fenêtre par forme de motif).
L'algorithme consiste donc à faire avancer les différentes fenêtres en fonction du pretraitement du motif et ce jusqu'à la fin de la séquence. Si une des fenêtre correspond à son motif associé, son indice est ajouté à la sortie de l'algorithme.

Construction du *DotPlot*

Pour construire le *DotPlot*, l'algorithme KMP est appliqué sur une séquence. Pour cela, une fenêtre parcours toute la séquence caractère par caractère et lance une recherche sur le motif présent dans la fenêtre.
Afin d'optimiser ce parcours, la recherche est uniquement lancé sur la séquence à partir de l'indice du début de la fenêtre (le motif n'est pas présent avant sinon la recherche aurait déjà été faite) et un ensemble de tous les motifs recherché permet de ne pas relancer une recherche sur un moitf déjà rencontré.

L'exemple présent dans le dossier `generate` a été généré depuis la séquence `chromosome13_NT_009952.fasta` pour les motifs de taille 10.
La génération du fichier `.dat` a pris environ 7 minutes.