Quentin Baert
Alexandre Verkyndt

AeA - TP 02

Identification de microARN

Réalisation

Le travail était à diviser en 3 sous tâches :

- Sous tâche 1 : Écriture d'un simulateur de structures tige/boucle
- Sous tâche 2 : Écriture d'un algorithme de recherche des structures tige/boucle dans une séquence génomique
- Sous tâche 3 : Exploration des données biologiques à la recherche des pré-microARN

Seule la première sous tâche a été réalisée en entière.
La seconde a été commencée, la détection de structure tige/boucle symétriques a notamment été codée.
La troisième n'a pas été abordée.

Utilisation

- Pour générer une structure tige/boucle depuis quelques paramètres

```
java -jar structGenerator.jar <tailleSequence> <nb> <tailleTerm>

Avec :
    <tailleSequence> : taille de la structure à générer
    <nb>             : nombre de boucles internes
    <tailleTerm>     : taille de la boucle terminale
```

Exemple :

```
$ java -jar structGenerator.jar 100 32 8
((((((..((((((..(((...(((...(((...(((...(((((((((.........)))))))..)))))..)))).)))..)))).))))
)).))))..)))...))))
AAGCUCGACAGGUAUCUCAUGUAUCUUUUAUGAUAAAUCAUCAGGUUGCGAUCCGAUGGCCUGAGGUUUUGUCCGGGAUAGA
GAACCUGCGGAGACGUUU
```

Architecture

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

- Le package `adn` contient toutes les classes permettant de générer une structure tige/boucle
- Le package `algo` contient la classe qui permet de déterminer si une sequence génomique est une structure tige/boucle symétrique
- le package `main` contient la classe main qui a servie à la génération du `.jar` fourni
- le package `test` contient les tests

Algorithmes

Simulation des structures tige/boucle

Le but de cette sous tâche est la réalisation d'un simulateur produisant aléatoirement des structures tige/boucle correspondant aux pré-microARN qui devront être détectés par l'algorithme de recherche des pré-microARN.

Pour ce faire, nous avons défini une classe `StructTigeBoucle`. Cette classe représente les contraintes que devra respecter la structure qui va être générée. Certaines de ces contraintes sont des constantes qui ne peuvent être modifiées, d'autres sont des paramètres que l'on peut donner à la construction.
Si certains paramètres ne sont pas spécifiés, des valeurs aléatoires sont définies. Ces valeurs sont comprises dans les intervalles respectants les contraintes. Une classe de test a été codée pour s'assurer que la génération des objets StructTigeBoucle soit fiable.

Ensuite, un objet `PréMicArnGenerator` a été codé. Celui-ci prend en paramètre un objet `StructTigeBoucle`. Ce générateur pourra ensuite générer toute sorte de pré-micro ARN respectant les contraintes apportées.

Génération :

- Le générateur commence par établir une structure de base : il place la boucle terminale dont la taille est passée en paramètre. Il encadre la boucle terminal du bon nombre d'appariements (également passé en paramètre).

- Ensuite il va généré une liste de chaînes de un à trois points en considérant le nombre de nucléotides manquants pour atteindre la taille voulue.
- Enfin un algorithme de placement va essayer d'insérer dans la structure de base les différentes chaînes de la liste. Ces insertions ne doivent pas couper les appariements n'importe comment : la règle des trois appariements successifs doit être respectée. De plus deux chaînes de points ne peuvent se trouver l'une à côté de l'autre pour éviter d'avoir plus de trois points côtes à côtes.

Une fois ces trois étapes réalisées, nous avons un "squelette" constitué de parenthèses et de points tel que :

```
(((.(((..(((...((((.(((.((((((.(((........)))...)))...)))..)))..)))))))..)))))))
```

De là nous appliquons un algorithme qui va remplacer les parenthèses deux à deux avec des appariements de nucléotides et les points par des nucléotides non compatibles de façon à obtenir :

```
UCCUCACGACACCCCGCCCUUACCCGGUACCCGGUUGACUGUGGGUCUAUCUCCGGCCUAGGGGGUGUGCGGUGGGG
```

Tout le code du générateur a été commenté et documenté.

Détection d'une structure tige/boucle symétrique

La détection de structure tige/boucle symétrique se fait grâce à deux pointeurs : l'un pointant sur le début de la séquence, l'autre sur la fin.
Tant que les deux pointeurs ne se sont pas rencontrés, l'un avance et l'autre recule sur la séquence. Lors de ce parcours, il s'agit ensuite de vérifier que la séquence pour l'instant parcourue remplie toutes les conditions pour être considérée comme une structure tige/boucle.

Pour voir des exemples de détection, se référer au package `test` et à la classe `TestDetectionTigeBoucleSymetrique`.