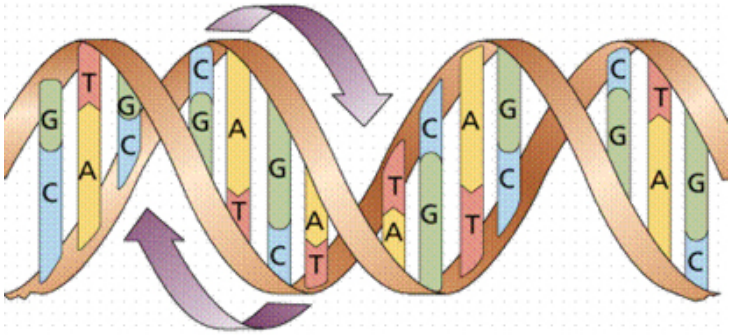
FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

Informatique (INFO-F-206) – Projet
Séquences ADN et répétitions en tandem

Gwenaël Joret

30 octobre 2013

1 Introduction



En bio-informatique, une *séquence ADN* est une séquence de lettres provenant de l'alphabet {`A`, `C`, `G`, `T`} (pour *Adénine*, *Cytosine*, *Guanine*, et *Thymine*), comme par-exemple

`ACGACTGACTAC`

Notons que les séquences ADN que nous allons considérer dans ce projet n'ont pas nécessairement d'interprétation biologique, *toute* séquence utilisant les lettres {`A`, `C`, `G`, `T`} sera en effet appelée séquence ADN. Les séquences ADN provenant de la biologie ont la particularité qu'elles contiennent souvent des *répétitions en tandem*, des motifs qui se répètent à l'identique et qui sont adjacents dans la séquence.

Formellement, une répétition en tandem est une séquence ADN de longueur paire et non nulle telle que la première moitié de la séquence est égale à sa seconde moitié. (Remarquez que la simple répétition d'une lettre, comme `CC`, est un cas particulier d'une répétition en tandem). On dit qu'une séquence ADN σ *contient* une répétition en tandem s'il existe une sous-séquence de lettres consécutives de σ qui soit une répétition en tandem. La séquence ci-dessus contient par-exemple la répétition en tandem `GACTGACT`. D'autre part, on peut vérifier que la séquence suivante

`CTATCACGCTACAGCGTATGCAGACGTCAGTCTATGCAGACATGACGATGCATCAGTAGACATGTCTAGCTCGATGAGTCGATGCAG`

ne contient *pas* de répétition en tandem.

Les répétitions en tandem étant fréquentes en biologie—elles représentent par-exemple 5% de la composition totale du génome humain—on peut se demander si celles-ci sont tout simplement inévitables en général, c-à-d si toute séquence ADN *arbitraire* suffisamment longue contient nécessairement une répétition en tandem. Cette question a été étudiée, dans un vocabulaire légèrement différent, il y a déjà plus de 100 ans par le mathématicien norvégien Axel Thue. Celui-ci a en particulier montré qu'il existe des séquences ADN arbitrairement longues ne contenant aucune répétition en tandem !

Le résultat de Thue en a fasciné plus d'un (essayez en effet de construire une séquence ADN de longueur 100 à la main sans répétition en tandem...), et diverses méthodes ont été proposées tout au

long du 20ème siècle pour générer de longues séquences ne contenant pas de répétition en tandem. Très récemment, trois informaticiens polonais, Jarosław Grytczuk, Jakub Kozik et Piotr Micek, ont proposé en 2011 un algorithme particulièrement simple et élégant pour produire de telles séquences. C'est cet algorithme que nous vous proposons d'implémenter dans ce projet. Une particularité de cet algorithme est qu'il est *probabiliste*, c-à-d qu'il effectue des choix aléatoires pour s'aider à construire la séquence désirée. Son temps d'exécution est également aléatoire, deux exécutions de l'algorithme avec les même paramètres pourront ainsi prendre des temps différents.

2 L'algorithme

Supposons que l'on souhaite construire une séquence ADN sans répétition en tandem qui soit de longueur n (le paramètre n sera spécifié par l'utilisateur de votre programme). L'algorithme de Grytczuk, Kozik, et Micek peut être alors informellement décrit comme suit : Initialement, on démarre avec une séquence ADN σ vide. On va ensuite *essayer* de faire grandir σ en lui ajoutant des lettres une à une, chaque lettre étant choisie uniformément au hasard dans notre alphabet {`A, C, G, T`}. Cependant, si l'ajout d'une lettre donne lieu à l'apparition d'une répétition en tandem, on efface alors de σ la seconde partie de la-dite répétition (la taille de notre séquence σ *diminuera* donc dans ce cas !). On continue à faire ces essais d'ajouts de lettres aléatoires tant que la taille de la séquence σ est inférieure à la taille n désirée. Une fois cette taille atteinte, on arrête simplement l'algorithme.

Avant de passer à une description plus formelle de l'algorithme, considérons l'exemple suivant illustrant une exécution possible de l'algorithme pour $n = 6$:

$\sigma = \varnothing$	*(initialement σ est vide)*
	(on choisit au hasard la lettre `C`*)*
$\sigma =$ `C`	*(on ajoute* `C`*)*
	(on choisit au hasard la lettre `T`*)*
$\sigma =$ `CT`	*(on ajoute* `T`*)*
	(on choisit au hasard la lettre `A`*)*
$\sigma =$ `CTA`	*(on ajoute* `A`*)*
	(on choisit au hasard la lettre `T`*)*
$\sigma =$ `CTAT`	*(on ajoute* `T`*)*
	(on choisit au hasard la lettre `A`*)*
$\sigma =$ `CTATA`	*(on ajoute* `A`*)*
	(répétition en tandem `TATA` *créée ! on efface la seconde moitié* `TA`*)*
$\sigma =$ `CTA`	
	(on choisit au hasard la lettre `A`*)*
$\sigma =$ `CTAA`	*(on ajoute* `A`*)*
	(répétition en tandem `AA` *créée ! on efface la seconde moitié* `A`*)*
$\sigma =$ `CTA`	
	(on choisit au hasard la lettre `G`*)*
$\sigma =$ `CTAG`	*(on ajoute* `G`*)*
	(on choisit au hasard la lettre `C`*)*
$\sigma =$ `CTAGC`	*(on ajoute* `C`*)*
	(on choisit au hasard la lettre `G`*)*
$\sigma =$ `CTAGCG`	*(on ajoute* `G`*)*

Observons que, puisque nous prenons soin de maintenir le fait que σ ne contienne pas de répétition en tandem, si l'ajout d'une lettre à σ fait apparaître une telle répétition, alors celle-ci doit

nécessairement contenir la lettre que nous venons d'ajouter, et est par conséquent un *suffixe* de la séquence σ. Cette observation facilitera notre tâche de vérification : il suffira en effet de passer en revue chaque suffixe de longueur paire de σ, et de vérifier qu'aucun d'eux n'est une répétition en tandem.

Décrivons maintenant l'algorithme de manière plus précise :

1. Initialiser σ comme séquence vide
2. Répéter tant que σ est de longueur $< n$:
   (a) Choisir un entier aléatoire j uniformément distribué entre 1 et 4
   (b) Ajouter la $j$ème lettre de l'alphabet à la fin de σ
   (c) Soit $\sigma = x_1 x_2 \dots x_t$, où x_i est la $i$ème lettre de σ
   (d) Soient $p = 1$ et *arret* = faux
   (e) Tant que $2p \leq t$ et *arret* = faux :
      i. Si $x_{t-2p+1} x_{t-2p+2} \dots x_t$ est une répétition en tandem :
         A. Effacer les p dernières lettres de σ
         B. Poser *arret* = vrai
      ii. Sinon :
         A. Incrémenter p de 1
3. Retourner σ

3 Énoncé du travail

On vous demande d'écrire un programme mettant en œuvre l'algorithme décrit ci-dessus. De plus, votre programme prendra le paramètre n en ligne de commande (voir ci-dessous), et affichera également le contenu de la séquence σ au début de chacune des itérations de la boucle principale.

Pour cela, vous devrez utiliser les structures du langage Python vues en cours. En particulier, on vous demande :

— de diviser le programme en fonctions élémentaires,
— de commenter de manière pertinente les différentes parties du programme (notamment en utilisant les « `docstrings` », voir syllabus, Chapitre 3, « Documenter ses fonctions »),
— d'utiliser le générateur de nombres pseudo-aléatoires et les paramètres en ligne de commande, comme décrits ci-dessous.

3.1 Génération de nombre pseudo-aléatoires

Le fonctionnement d'un ordinateur étant fondamentalement déterministe, il est illusoire de vouloir générer des nombres réellement aléatoires. Il existe néanmoins des méthodes déterministes pour générer des suites de nombres qui semblent satisfaire aux critères d'une suite aléatoires. On parle alors de nombres *pseudo-aléatoires*.

En Python 3, on peut utiliser le module `random`, en le précisant en début de programme grâce à l'instruction `import` :

```
import random
```

La fonction `random.randint (a, b)` renvoie un nombre entier pseudo-aléatoire entre `a` et `b`, où `a` et `b` sont tous les deux inclus. Voici un petit exemple en mode interactif :

```
>>> for i in range(5):
...   print(random.randint(3,13))
...
6
7
11
13
12
```

3.2 Paramètres en ligne de commande

L'interpréteur Python peut être utilisé de différentes manières. La manière la plus simple est d'utiliser l'outil IDLE (cf. cours).

Il est cependant possible d'invoquer l'interpréteur Python à partir d'un terminal. Sous Linux (sur les ordinateurs mis à votre disposition au bâtiment NO), il est possible de faire apparaître une fenêtre terminal via le menu en bas à gauche : *Système* → *Terminal*. Il suffit alors de taper la commande

```
python3 fichier.py
```

où `fichier.py` est le nom de votre fichier source, pour exécuter votre programme. *(NB. Veillez à appeler l'exécutable correspondant à la version 3 de python. Sur certains systèmes, celui-ci s'appelle simplement* `python` *au lieu de* `python3`*).*

Exécuter un programme Python via le terminal permet de passer des paramètres au programme, d'une manière similaire au mécanisme de passage de paramètres à une fonction. Le début de votre programme devra contenir la ligne suivante :

```
import sys
```

Lors de l'invocation de l'interpréteur Python via le terminal, des valeurs peuvent être ajoutées après le nom du programme (ici `projetINFOF206.py`), par exemple :

```
python3 projetINFOF206.py 25
```

Ces valeurs peuvent être récupérées par le programme, dans la liste appelée `sys.argv`. Dans l'exemple précédent, l'instruction Python `print (sys.argv)` affichera les éléments suivants :

```
['projetINFOF206.py', '25']
```

Chaque élément de la liste `sys.argv` est de type `str`. Ils peuvent être convertis vers d'autres types à l'aide des fonctions de conversion de types, comme `int()` ou `float()`.

Dans ce travail, on vous demande de passer un paramètre en ligne de commande, soit la taille n désirée pour la séquence ADN sans répétition en tandem. Si votre programme s'appelle `projetINFOF206.py`, la commande précédente devrait lancer l'algorithme avec $n = 25$.

Pour en savoir plus. L'article original de Grytczuk, Kozik, et Micek est disponible à l'adresse suivante : `http://arxiv.org/abs/1103.3809`. Notez que sa lecture n'est aucunement nécessaire à la réalisation de ce projet. L'article contient entre autre une analyse du nombre d'itérations nécessaires *en moyenne* pour que l'algorithme produise une séquence de longueur n (sans entrer dans les détails, ce nombre moyen est linéaire en n).

Personnes de contact

Les assistants en charge du projet sont Liran Lerman (`liran.lerman@ulb.ac.be`) et Cédric Ternon (`cedric.ternon@ulb.ac.be`). Vous pouvez vous adresser à ceux-ci, au titulaire du cours, ou aux élèves-assistants qui organisent les guidances informatiques pour toute question que vous auriez. (Notez que votre assistant de TP n'est pas nécessairement en charge du projet).

4 Consignes pour la remise du projet

À respecter scrupuleusement !
Le projet ne sera pas testé à la main, un traitement automatisé sera appliqué. Pour que votre projet soit pris en compte par ce traitement automatisé, il est indispensable que vous respectiez les instructions ci-dessous à la lettre.

LES PROJETS QUI NE RESPECTENT PAS CES INSTRUCTIONS SERONT CONSIDÉRÉS COMME « NON REMIS ».

1. Ce projet est INDIVIDUEL ! Il est bien entendu acceptable d'en discuter entre collègues, mais deux copies trop similaires seront considérées comme frauduleuses ;
2. Toute forme de plagiat sera également sévèrement sanctionnée. A ce sujet, consulter la page des bibiothèques de l'ULB : `http://www.bib.ulb.ac.be/fr/aide/eviter-le-plagiat/`. La notion de plagiat s'applique naturellement aux programmes ou morceaux de programmes.
3. Votre travail devra être remis au plus tard le **lundi 1 décembre 2014 avant 15h** sous deux formats :
   — sous forme d'un fichier projetINFOF206.py sur l'Université Virtuelle
   — sur papier, au secrétariat étudiants du département d'Informatique (local 2.N8.104 - Plaine), une version imprimée du code source de chacun des fichiers soumis sur l'UV. Chaque feuille reprendra **le nom et le prénom** de l'étudiant, ainsi que son **numéro de matricule** ;
4. Les dates et heures de remises sont à comprendre au sens strict. Aucun retard ne sera toléré ;
5. Veuillez numéroter vos feuilles clairement et ne remettez pas de feuilles volantes ;
6. Votre projet doit être **dactylographié**. Les projets écrits à la main ne seront **pas corrigés** (0/10) ;
7. Votre code doit être **commenté de manière pertinente**. Veillez par ailleurs à bien soigner sa présentation ;
8. En dehors des cas où l'énoncé le spécifie explicitement, l'utilisation de librairies n'est pas autorisée ;
9. L'ensemble du projet sera à réaliser en Python 3 ;
10. Une mini-défense du projet de quelques minutes sera organisée lors de la dernière séance de TP, durant laquelle vous devrez répondre à quelques questions simples sur le fonctionnement et la structure de votre programme.