Algorithmes pour l'étude de l'ADN

F.Gaudon

23 décembre 2010

Table des matières

1 Avant de commencer

Ce document contient une liste d'algorithmes permettant l'étude de l'ADN en lycée et les programmes correspondants.

Le langage de programation Python a été choisi car il est considéré par les développeurs professionels comme l'un des plus appropriés pour l'apprentissage de la programmation. Le langage de programmation du logiciel XCas a été choisi pour sa structure francisée et néanmoins très proche de celle des langages de programmation usuels (Python, C entre autres), ce qui en fait un logiciel particulièrement adapté à un public francophone.

Ce document suppose une connaissance des instructions fondamentales (entrées, sorties, boucles, conditions) du langage Python ainsi que de celles du logiciel XCas. Pour le langage Python, afin d'utiliser des caractères accentués dans les programmes, on n'oubliera pas de préciser l'encodage de caractères utilisé en plaçant au début de chaque programme la ligne suivante sous Windows

```
#-*- coding:Latin-1 -*-
```

et sous Linux ubuntu

```
#-*- coding:Utf-8 -*-
```

2 Génération aléatoire d'un code ADN

L'algorithme suivant génère un code ADN aléatoire de longueur l spécifiée par l'utilisateur.

Entrées :
l : nombre
Début traitement
  $code$ prend la valeur "" ;
  pour k ***allant de*** 1 ***à*** l **faire**
    a prend la valeur d'un nombre aléatoire entier entre 1 et 4 ;
    si $a = 0$ **alors**
      $code$ prend la valeur de $code + "A"$;
    fin
    si $a = 1$ **alors**
      $code$ prend la valeur de $code + "T"$;
    fin
    si $a = 2$ **alors**
      $code$ prend la valeur de $code + "C"$;
    fin
    si $a = 3$ **alors**
      $code$ prend la valeur de $code + "G"$;
    fin
  fin
Fin
Sorties : $code$

Programmes correspondants :

XCas :

```
Saisir("longueur : ",l);
code:="";
pour k de 1 jusque l faire
  a:=alea(4);
  si a==0 alors code:=code+"A";fsi;
  si a==1 alors code:=code+"T";fsi;
  si a==2 alors code:=code+"C";fsi;
  si a==3 alors code:=code+"G";fsi;
fpour;
afficher("Code ADN : ",code);
```

Python :

```
from random import *
l=input("Longueur du code : ")
code=""
for k in range(0,l):
    a=randint(1,4)
    if a==1:
        code=code+"A"
    if a==2:
        code=code+"T"
    if a==3:
        code=code+"C"
    if a==4:
        code=code+"G"
print ("code ADN : "+code)
```

3 Génération du code complémentaire à un code ADN donné

Entrées :
code : chaîne de caractères composée de A, T, C ou G.
Début traitement
  codecompl prend la valeur "" ;
  n prend la valeur de la longueur de *code*
  pour k ***allant de*** *1* ***à*** n **faire**
    si $code[k] =$ "A" **alors**
      codecompl prend la valeur $codecompl +$ "T" ;
    fin
    si $code[k] =$ "T" **alors**
      codecompl prend la valeur $codecompl +$ "A" ;
    fin
    si $code[k] =$ "C" **alors**
      codecompl prend la valeur $codecompl +$ "G" ;
    fin
    si $code[k] =$ "G" **alors**
      codecompl prend la valeur $codecompl +$ "C" ;
    fin
  fin
Fin
Sorties : *codecompl*

Programmes correspondants :

XCas :

```
code:="AATCGGGA";
codecompl:="";
n:=length(code);
pour k de 1 jusque n faire
  si code[k-1]=="A" alors ...
    ...codecompl:=codecompl+"T";fsi;
  si code[k-1]=="T" alors ...
    ...codecompl:=codecompl+"A";fsi;
  si code[k-1]=="C" alors ...
    ...codecompl:=codecompl+"G";fsi;
  si code[k-1]=="G" alors ...
    ...codecompl:=codecompl+"C";fsi;
fpour;
afficher(codecompl);
```

Python :

```
code="AATCGGGA"
codecompl=""
n=len(code)
for k in range(1,n):
    if code[k-1]=="A":
        codecompl=codecompl+"T"
    if code[k-1]=="T":
        codecompl=codecompl+"A"
    if code[k-1]=="C":
        codecompl=codecompl+"G"
    if code[k-1]=="G":
        codecompl=codecompl+"C"
print("code complementaire : ",codecompl)
```

Remarque :
Dans l'algorithme on utilise $code[k]$ pour désigner le k^{e} caractère de la chaîne de caractère *code* alors que dans les programmes on utilise $code[k-1]$: cela vient du fait que pour les langages Python et XCas, le premier caractère d'une chaîne de caractères est le caractère de rang 0.

4 Algorithme CGR de représentation de l'ADN

Nous ne décrirons pas ici l'algorithme CGR. On en trouvera une description très claire de l'algorithme CGR à l'adresse suivante :
http ://www.mathom.fr/mathom/FeteDeLaScience/FS2007/complements.html
Le programme suivant trace la représentation de l'ADN donnée par l'algorithme CGR :
Python :

```
from Tkinter import*

def tracer(sequence):
    "Tracé"
    taille=len(sequence)
    x1=205
    y1=205
    for car in sequence:
        if car=="A":
            x2=10
            y2=10
        if car=="C":
            x2=410
            y2=10
        if car=="G":
            x2=410
            y2=410
        if car=="T":
            x2=10
            y2=410
        can1.create_line(x1,y1,x2,y2,fill='white')
        x1=(x1+x2)/2
        y1=(y1+y2)/2
        can1.create_oval(x1-3,y1-3,x1+3,y1+3,fill='blue')

sequence=raw_input("Entrer la séquence ADN à représenter : ")
fen1=Tk()
can1=Canvas(fen1,height=500,width=500,background='black')
can1.pack(side=LEFT)
can1.create_line(10,10,410,10,fill='white')
can1.create_line(410,10,410,410,fill='white')
can1.create_line(410,410,10,410,fill='white')
can1.create_line(10,410,10,10,fill='white')
bou1=Button(fen1,text='Quitter',command=fen1.quit)
bou1.pack(side=BOTTOM)
bou2=Button(fen1,text='Tracer',command=tracer(sequence))
bou2.pack()
fen1.mainloop()
fen1.destroy()
```