TD: recherche textuelle:

Exercice 1: recherche d'un caractère:
On s'interesse au programme suivant:

```
0 def recherche(texte, motif):
      for car in texte:
2         if car == motif:
              return True
4     return False

6 texte = 'azertyuiopqsdfghjklmwxcvbnazertyuiopqsdfghjklmwxcvbn'
  motif = 'g'
8
  print(recherche(texte,motif))
```

recherche_caractere.py

1. Expliquer ce qu'il permet de faire.
2. Que va renvoyer la ligne 9?
3. Ecrire une deuxième fonction qui peut renvoyer la place du caractère s'il a été trouvé.
4. Ecrire une troisième fonction qui peut renvoyer les places du caractère s'il a été trouvé plusieurs fois.

Exercice 2: recherche naïve d'un motif:
On peut procéder naïvement de la façon suivante pour chercher un motif dans un texte:

- on recherche la présence du premier caractère du motif dans le texte.
- si on le trouve, on vérifie si les caractères suivants du motif coïncident avec ceux du texte.
- si tous les caractères coïncident, le motif est trouvé, sinon, on reprend la recherche du premier caractère.

1. Ecrire une fonction de recherche naïve d'un motif dans un texte.
2. Il est possible d'écrire la fonction de la manière suivante:

```
0 def recherche_bis(texte, motif):
      n = len(texte)
2     m = len(motif)
      i = 0
4     j = 0
      sol = []    #pour stocker les solutions
6     while i < m and j < n:
```

```
        if motif[i] != texte[j]:
            j = j-i+1   #decalage d'un caractere
            i = 0       #reprise des comparaisons a partit du premier caractere du
    motif
        else:
            i += 1
            j += 1
        if i >= m:
            sol.append(j-m)
            i = 0
    return sol
```

recherche_naive_motif.py

Expliquer son fonctionnement ainsi que ses limites. On pourra par exemple discuter du cas où texte = 'abcababcabc' et motif = 'abcabc'.

Exercice 3: recherche d'un motif avec pré-traitement: méthode de Morris et Pratt:
Le but de cette méthode est de faire un pré-traitement du motif afin de gagner en efficacité sur la recherche. On propose le code suivant:

```
def morris_pratt(texte, motif):
    n = len(texte)
    m = len(motif)
    i = 0
    j = 0
    sol = []                    #pour stocker les solutions
    s = traitement(motif)       #traitement du motif (HP)
    while i < m and j < n:
        if i >= 0 and motif[i] != texte[j]:
            i = s[i]
        else:
            i += 1
            j += 1
        if i >= m:
            sol.append(j-m)
            i = 0
    return sol


def traitement(motif):
    m = len(motif)
    b = [-1]
    for j in range(1,m):
        i = b[j-1]
        while i >= 0 and motif[j-1] != motif[i]:
            i = b[i]
        b.append(i+1)
    return b
```

morris_pratt.py

1. Quelle est la partie commune aux codes des exercices 2 et 3?
2. Expliquer le rôle et le fonctionnement de la partie traitement.

Exercice 4: recherche d'un motif avec pré-traitement: méthode de Boyer et Moore:
Cet algorithme a la particularité de comparer texte et motif de droite à gauche en commençant par la fin du motif. Ainsi, il est possible de faire un pré-traitement interessant.

On peut en trouver un exemple à l'adresse `https://www.cs.utexas.edu/users/moore/best-ideas/string-searching/fstrpos-example.html`

On peut le coder de la manière suivante:

```
0  #dictionnaire des distances (version simple):
   def distances_v1(motif):
2      m = len(motif)
       dico = {i : m for i in 'azertyuiopqsdfghjklmwxcvbnAZERTYUIOPQSDFGHJKLMWXCVBN'}
4      for i in range(m-1):
           dico[motif[i]] = m-1-i
6      return dico


8
   d = distances_v1('exemple')
10 print(d)


12
   def boyer_moore(texte,motif):
14     m = len(motif)
       n = len(texte)
16     d = distances_v1(motif)
       s = m-1      #on commence face au dernier caractere du motif
18     sol = []
       while s < n:
20         i = m-1
           while i >= 0 and motif[i] == texte[s]:
22             s -= 1
               i -= 1
24         if i < 0:
               sol.append(s+1)
26         s += max(d[texte[s]],m-i)    #decalage
       return sol
```

boyer_moore.py

1. Expliquer le rôle de la fonction *distance()*. Que va renvoyer la ligne 10.
2. La fonction distance ne prend pas en compte l'intégralité des caractères possibles. Proposer une solution permettant de l'améliorer.
3. Expliquer le fonctionnement de *boyer_moore()*.

Exercice 5: efficacité des différents algorithmes:
Ces algorithmes peuvent évidemment être utilisés pour de la recherche en génétique.

1. Créer une chaine de 100 000 caractères comprenant de manière aléatoire les symboles A, T, G, C.
2. Rechercher dans cette liste le motif CAGCAG.
3. Comparer les temps d'exécution des différents algorithmes.