Chapitre 10 : Recherche textuelle

10.1 Principe

On a tous été un jour ou l'autre amenés à chercher une chaîne de caractères dans un documement texte `ctrl-F` ou encore *Édition > Chercher* c'est une fonctionnalité indispensable à tout logiciel manipulant du texte.
C'est ce que l'on appelle la recherche d'un **motif** dans un texte.

Dans le texte ci-dessous peut-on trouver le motif « met » ?

*Lorem ipsum dolor sit a**met**, consectetur adipiscing elit. Donec aliquam*
***met**us non dignissim fringilla. Nulla nisl purus, congue eget **met**us non,*
convallis pellentesque nulla. Morbi id tincidunt augue, id ullamcorper
erat. Nullam at orci erat. Donec fermentum semper mi, sed al

La réponse est oui, il apparaît 3 fois (ligne 1 colonne 23, ligne 2 colonne 0, ligne 2 colonne 61)

Bioinformatique

Les algorithmes de recherche textuelle sont notamment utilisés en bioinformatique.
La bioinformatique est une science à la jonction des mathématiques et des sciences informatiques d'une part, et des sciences de la vie (biochimie, biologie, microbiologie, écologie, épidémiologie) d'autre part.
Les domaines d'étude des sciences du vivant génèrent des quantités croissantes de données qu'il convient de stocker, traîter, visualiser, exploiter...
On peut citer par exemple le séquençage du génome humain. Le génome est l'ensemble du matériel génétique d'une espèce, codé dans son ADN ou dans son ARN pour les virus.
Le séquençage du génome humain a été achevé en 2003, grâce au Projet Génome Humain (PGH).
L'information génétique est souvent représentée sous forme de longues chaînes de caractères, composées des caractères A, T, G et C. Chacune de ces lettres représente une base azotée : Adénine, Thymine, Guanine et Cytosine pour l'ADN.
L'ARN est quant à lui composé des quatre bases Adénine, Guanine, Cytosine et Uracile.
Exemple : CTATTCAGCAGTC...

10.2 Algorithmes

a. Algorithme naïf

L'algorithme naïf est la façon la plus naturelle de rechercher un motif dans une chaîne de caractères.
On considère par exemple un motif de trois caractères dont on cherche à savoir s'il est présent ou non dans une chaîne de caractères.
On pose comme précondition que la longueur de la chaîne de caractères est supérieure ou égale à la longueur du motif.

- On compare le premier caractère du motif au premier caractère de la chaîne. Deux possibilités :
  - ⋆ les caractères correspondent, on passe alors au deuxième caractère du motif et de la chaîne
  - ⋆ les caractères ne correspondent pas, on compare alors le premier caractère du motif au second caractère de la chaîne

On répète ce procédé pour chaque caractère du motif et de la chaîne en faisant attention à ne pas déborder de la chaîne.

chaîne	A	G	G	T	C	G	A	T	G	T
motif		G	A	C						

Avant de passer à l'implémentation, aller à cette adresse `https://boyer-moore.codekodo.net/` et visualiser le déroulement de l'algorithme naïf.

Implémentation en Python : recopier et compléter ce script dont l'objectif est de retourner la ou les position(s) de la chaîne ou se trouve éventuellement le motif.

```
def recherche_textuelle_naif(motif, chaine):
    assert ..................., 'chaine trop courte'
    n = len(chaine)
    m = len(motif)
    i = 0
```

```
    resultat = []
    trouve = False
    while i < ......:
        j = 0
        while j < ....... and motif[j] == chaine[i+j]:
            j ......
        if j == m:
            ............
        i += 1
    return resultat
```

```
>>>recherche_textuelle_naif("AATC", "GTAATCAAATCTTGCCAATCAATC")
[2, 7, 16]
>>>recherche_textuelle_naif("GATACA", "GTAATCAAATCTTGCCAATCAATC")
[]
```

▶ Exercice 1

En utilisant la fonction `recherche_textuelle_naif` écrire une fonction `affichegras(motif, chaine)` qui affiche en gras le motif à chaque fois qu'il est présent dans la chaîne.

```
affichergras("AATC", "GTAATCAAATCTTGCCAATCTC")
```

GT**AATC**A**AATC**TTGCC**AATC**TC
On peut afficher un texte en gras dans une console à l'aide de deux caractères en UTF-8 ainsi

```
print('texte normal' + '\033[1m' + 'texte en gras' + '\033[0m')
```

b. Algorithme Boyer-Moore

L'obectif de l'algorithme de Boyer-Moore (B-M) simplifié est d'améliorer la méthode naïve qui ne déplace les comparaisons à chaque étape que d'un seul cran, alors qu'il est possible la plupart du temps de se déplacer de plusieurs crans. On doit pouvoir calculer dans chaque cas le décalage maximum qu'il est possible d'effectuer.

Voici une animation permet de mieux visualiser l'idée sous-jacente de cet algorithme.
Aller à l'adresse suivante : `https://boyer-moore.codekodo.net/` et visualiser l'algorithme de B-M

```
'''permet de connaitre le decalage a faire
en fonction de la correspondance ou pas '''

def correspondance(motif, chaine, adroite, p, i):
    # j varie de p-1 à 0 inclus en decroissant
    for j in range(p - 1, -1, -1):
        x = chaine[i + j]
        if x != motif[j]:
            if x in adroite.keys():
                decalage = max(1, j - adroite[x])
            else:
                decalage = 1
            return (False, decalage)
    return (True,0)
```

```python
def boyer_moore(motif, chaine):
    #trouver l'indice du caractere le plus a droite du motif
    #le stock dans un dictionnaire adroite
    adroite = {}
    for indice, lettre in enumerate(motif):
        adroite[lettre] = indice

    resultat = []
    n = len(chaine)
    p = len(motif)
    i = 0
    while i < n-p:
        ok, decalage = correspondance(motif,chaine, adroite, p, i)
        if ok:
            resultat.append(i)
            i = i + p
        else:
            i = i + decalage
    return resultat
```

On peut remarquer que l'on a bien, en fonction des cas, effectué plusieurs décalages en un coup, ce qui, au bout du compte, permet de faire moins de comparaisons que l'algorithme naïf. On peut aussi remarquer que plus le motif est grand, plus l'algorithme de Boyer-Moore est efficace.

▶ Exercice 2

Placer un compteur dans les boucles des fonctions `recherche_textuelle_naif` et `boyer_moore` pour afficher le nombre d'étapes sur plusieurs exemples comme ci-dessous.

```
>>> boyer_moore("AATC", "GTAATCAAATCTTGCCAATCTC")
9
[2, 7, 16]
>>>recherche_textuelle_naif("AATC", "GTAATCAAATCTTGCCAATCTC")
18
[2, 7, 16]
```

Sur cet exemple on compte 9 étapes pour l'algorithme B-M contre 18 pour l'algorithme naïf. Il est difficile d'évaluer la complexité de cet algorithme en moyenne, on peut par contre évaluer la complexité au pire et au meilleur des cas.