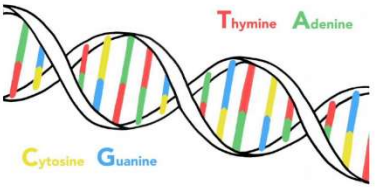
Recherche textuelle

1. La recherche textuelle

La plupart des applications comme celle sur laquelle vous lisez ces lignes, possèdent une fonction de recherche textuelle. En général pour y accéder on utilise la combinaison de touches **CTRL + F**.

Trouver les occurrences d'une chaîne de caractères (motif ou mot) dans un texte est un problème qui se rencontre fréquemment dans les éditeurs de texte.

En bio-informatique, on a recourt à la recherche textuelle pour la recherche de gènes dans l'ADN. L'ADN est composé de 4 nucléotides A (Adénine), G(Guanine), C(Cytosine) et T(Thymine) : on peut donc le considérer comme un texte d'un alphabet à 4 caractères et où le gêne recherché sera le motif.



Les algorithmes de recherche de chaîne de caractères sont également utilisés pour rechercher des pages web correspondantes à une recherche Internet.

2. La recherche textuelle avec Python

Python possède sa propre méthode de recherche, ce script affiche la présence ou non d'une occurrence (mot) dans un texte (phrase).

```
phrase = "Ceci n'est que la phrase qui sert d'exemple"
mot1 = "qui"
mot2 = "quiche"

def occurrence(mot, texte):
    if mot in texte:
        return True
    else:
        return False
```

✍ Tester les fonctions **occurrence(mot1, phrase)** et **occurrence(mot2, phrase)**.

```
>>> occurence(mot1, phrase)
>>> occurence(mot2, phrase)
```

Il peut être utile de rechercher l'indice d'un charactère dans un texte.

```
phrase = "Ceci n'est que la phrase qui sert d'exemple"

def indice(c, texte):
    for i in range(len(texte)):
        if texte[i] == c:
            return i
    return -1
```

✍ Tester les fonctions **indice('e', phrase)**, **indice('p', phrase)** et **indice('y', phrase)**.

```
>>> indice('e', phrase)
>>> indice('p', phrase)
>>> indice('y', phrase)
```

3. Algorithme naïf

Pour savoir si un mot est dans un texte, la méthode qui vient naturellement à l'esprit (on l'appelle méthode naïve) est la suivante : on parcourt le texte d'indice en indice depuis le début du texte en vérifiant à chaque pas si les lettres du mot coïncident.

```
Indice    0123456789
Texte     un exemple
Motif     em
Position  01
```

On place le motif (**'em'**) recherché au même niveau que les 2 premiers caractères du texte, le premier élément du motif ne correspond pas au premier élément du texte (**'u' ≠ 'e'**), on décale le motif d'un cran vers la droite.

```
Indice    0123456789
Texte     un exemple
Motif      em
Position   01
```

Le premier élément du motif (position 0) ne correspond pas à l'élément d'indice 1 du texte (**'n' ≠ 'e'**), on décale le motif d'un cran vers la droite et ainsi de suite…

```
Indice    0123456789
Texte     un exemple
Motif        em
Position     01
```

Le premier élément du motif (position 0) correspond à l'élément d'indice 3 du texte (**'e' = 'e'**) mais l'élément de position 1 du motif ne correspond pas à l'élément de position 4 du texte (**'x' ≠ 'm'**), on décale le motif d'un cran vers la droite et ainsi de suite…

```
Indice    0123456789
Texte     un exemple
Motif          em
Position       01
```

Le premier élément du motif (position 0) correspond à l'élément d'indice 5 du texte (**'e' = 'e'**) et l'élément de position 1 du motif correspond à l'élément de position 6 du texte (**'m' = 'm'**), le motif a été trouvé dans le texte.

✍ Écrire la fonction **rech_naive(motif, texte)** qui renvoie l'indice du motif dans le texte et -1 si le motif n'est pas dans le texte.

✍ Tester la fonction obtenue avec les assertions suivante :

```
assert(rech_naive('qui', "Ceci n'est que la phrase qui sert d'exemple") == 25)
assert(rech_naive('qu', "Ceci n'est que la phrase qui sert d'exemple") == 11)
assert( rech_naive('naif', "Ceci n'est que la phrase qui sert d'exemple") == -1)
```

4. Algorithme de Boyer-Moore-Horspool

4.1. Description du fonctionnement de l'algorithme

Dans la méthode naïve, à chaque étape on se décale d'un cran vers la droite. C'est en jouant sur ce décalage que l'on peut améliorer la méthode.

On recherche l'occurrence **CGGCTG** dans la séquence **ATAACAGGAGTAAATAACGGCTGGAGTAAATA**.
On aligne et on **teste l'occurrence par la droite** :

```
      x
CGGCTG
ATAACAGGAGTAAATAACGGCTGGAGTAAATA
```

Comme **G** et **A** ne correspondent pas et qu'il n'y a pas de **A** dans l'occurrence on décale l'occurrence de 6 rangs (la longueur de l'occurrence).

```
         x
      CGGCTG
ATAACAGGAGTAAATAACGGCTGGAGTAAATA
```

On est dans une situation similaire, et en deux étapes on obtient ce que la méthode naïve aurait fait en 12 étapes !

```
            x
         CGGCTG
ATAACAGGAGTAAATAACGGCTGGAGTAAATA
```

Dans cette situation, le **G** et le **C** ne correspondent pas mais il y a un **C** dans l'occurrence, on décalera donc l'occurrence de 2 rangs (place du premier **C** depuis la fin de l'occurrence).

```
               xo
            CGGCTG
ATAACAGGAGTAAATAACGGCTGGAGTAAATA
```

Cette fois-ci les **G** correspondent puis **T** et **G** ne correspondent pas, or il y a un **G** (avant le **C**) dans l'occurrence.
On décale donc de 3 rangs.

```
                  oooooo
                  CGGCTG
ATAACAGGAGTAAATAACGGCTGGAGTAAATA
```

On trouve une correspondance complète.

Pour continuer la recherche il suffit de la relancer un rang plus loin...
En appliquant à chaque étape un décalage adapté, on accélère grandement le processus.

4.2. Implémentation du prétraitement

Pour implémenter efficacement l'algorithme de Boyer-Moore-Horspool, il faut passer par un pré-traitement du motif pour facilement accéder au décalage à effectuer. On utilise un dictionnaire pour cela.

✍ Compléter ci-dessous la fonction **`calcul_decalage(motif)`** qui prend le motif en paramètre et qui retourne un dictionnaire associant à chaque lettre du préfixe du motif (texte situé avant la dernière lettre du motif ; exemple pour le motif souris, le préfixe est souri) le décalage à effectuer dans l'algorithme de Boyer-Moore-Horspool (décalage vers la gauche par rapport à la dernière lettre du motif).

```
def calcul_decalage(motif):
    decalage = {}
    . . . . . .
```

✍ Tester la fonction obtenue avec l'assertion suivante :

```
assert(calcul_decalage('Julien')) == {'J': 5, 'u': 4, 'l': 3, 'i': 2, 'e': 1}
```

✍ À l'aide de la fonction précédente, implémenter une fonction **`decale(lettre, motif)`** prenant en paramètres une lettre et un motif et retournant le décalage associé à la lettre du préfixe (ne pas oublier le décalage complet du motif si la lettre n'est pas présente dans le motif).

```
def decale(lettre, motif):
    decalage = calcul_decalage(motif)
    . . . . . .
```

✍ Tester la fonction obtenue avec les assertions suivante :

```
assert(decale('a','aababab') == 1)
assert(decale('b','aababab') == 2)
assert(decale('w','aababab') == 7)
```

4.3. Implémentation de l'algorithme

L'algorithme de Boyer-Moore-Horspool est le suivant :

```
fonction bmh(motif, texte):
    t ← longueur du texte
    m ← longueur du motif
    pos_deb_motif ← 0 (position du début du motif)
    pos_fin_motif ← 0 (position de la fin du motif)
    decalage ← calcul_decalage(motif)
    Tant que (pos_deb_motif + m) ≤ t: (le motif ne déborde pas du texte)
        pos_fin_motif ← m - 1
        Tant que texte[pos_deb_motif + pos_fin_motif] = motif[pos_fin_motif]: (correspondance
sur la dernière lettre du motif)
            pos_fin_motif = pos_fin_motif - 1 (on recule d'une lettre)
            si pos_fin_motif < 0: (toutes les lettres coïncident)
                retourner pos_deb_motif
        pos_deb_motif ← pos_deb_motif + decale(texte[pos_deb_motif + pos_fin_motif], motif)
(on décale le motif si pas correspondance)
    retourner -1 (motif non présent dans le texte)
```

✍ Implémenter la fonction **`bmh(motif, texte)`** retournant l'indice du début de la première occurrence du motif dans le texte.

✍ Tester la fonction obtenue avec les assertions suivante :

```
assert(bmh('aabba', 'aabbbababacaabbabaababababaab') == 11)
assert(bmh('xuv', 'aabbbababacaabbabaababababaab') == -1)
```