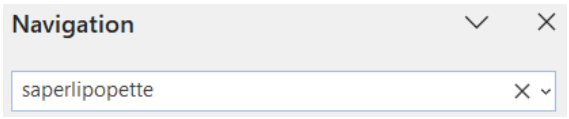
Chapitre 21 - Recherche textuelle

Dans la plupart des éditeurs de texte, il existe une fonctionnalité *Recherche* qui permet de rechercher la présence d'un motif dans un document.

Pour rechercher par exemple la présence du motif « *saperlipopette* » dans le texte ci-contre, on lance un « Ctrl F » et on saisi dans la fenêtre de dialogue :



Lorem ipsum dolor sit amet consectetur adipisicing elit. Beatae libero nesciunt illo consequatur doloribus adipisci, animi corporis molestiae ipsa perferendis ullam voluptate expedita magnam explicabo sint iste aut cumque vitae. Lorem ipsum dolor sit amet consectetur adipisicing elit. Beatae libero nesciunt illo consequatur doloribus adipisci, animi corporis molestiae ipsa perferendis ullam voluptate expedita magnam explicabo sint iste aut cumque vitae. Lorem ipsum dolor sit amet consectetur adipisicing elit. Beatae libero nesciunt illo consequatur doloribus adipisci, animi corporis molestiae ipsa perferendis ullam voluptate expedita saperlipopette magnam explicabo sint iste aut cumque vitae.

On découvre dans ce chapitre un algorithme développé par Robert S. Boyer et J Strother Moore en 1977. Il permet de réaliser une recherche avec une complexité faible. Avant de le présenter, on s'intéresse à l'algorithme de recherche naïf.

1- ALGORITHME NAIF :

Exemple :
```
texte = "GCATCGCAGAGAGTATACAGTACG"
motif = "GCAGAGAG"
```

index ***i***	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
texte	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
Motif	**G**	**C**	**A**	**G**	**A**	**G**	**A**	**G**																
index ***k***	0	1	2	3	4	5	6	7																

⇨ Code naïf avec 2 boucles for :

```
len(texte) =24    len(motif) =8    len(texte) - len(motif) =16
```

```
def recherche_naive(texte, motif):
    '''
    renvoie la liste des indices (éventuellement vide)
    des occurrences de la chaîne motif dans la chaîne texte.
    '''
    indices = []
    for i in range(len(texte) - len(motif) + 1):
        n = 0
        for k in range(len(motif)) :
            if texte[k+i] == motif[k]: n = n + 1
        if n == len(motif) :
            indices.append(i)
    return indices
```

Nombres de comparaisons : On obtient 136 comparaisons : (16 + 1) x 8 = 136

index *i*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
texte	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
Motif	**G**	**C**	**A**	**G**	**A**	**G**	**A**	**G**																
index *k*	0	1	2	3	4	5	6	7																

⇨ Code naïf avec 1 boucle for et une boucle while :

`len(texte)` = 24 `len(motif)` = 8 `len(texte) - len(motif)` = 16

```
def recherche_naive(texte, motif):
    '''
    renvoie la liste des indices (éventuellement vide)
    des occurrences de la chaîne motif dans la chaîne texte.
    '''
    indices = []
    for i in range(len(texte) - len(motif) +1):
        k = 0
        while k < len(motif) and texte[k+i] == motif[k]:
            k = k + 1
        if k == len(motif) :
            indices.append(i)
    return indices
```

Nombres de comparaisons : On obtient 31 comparaisons.

index *i*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
texte	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
Motif	**G**	**C**	**A**	**G**	**A**	**G**	**A**	**G**																
index *k*	0	1	2	3	4	5	6	7																

⇨ Code naïf en effectuant les comparaisons en commençant par la fin du motif :

`len(texte)` = 24 `len(motif)` = 8 `len(texte) - len(motif)` = 16

```python
def recherche_naive(texte, motif):
    '''
    renvoie la liste des indices (éventuellement vide)
    des occurrences de la chaîne motif dans la chaîne texte.
    '''
    indices = []
    i = 0
    while i <= len(texte) - len(motif):
        k = len(motif) - 1
        while k >= 0 and texte[i+k] == motif[k]:
            k = k-1
        if k == -1:
            indices.append(i)
        i = i + 1
    return indices
```

Nombres de comparaisons : On obtient 34 comparaisons.

⇨ Complexité de ce type d'algorithme : Dans ce type d'algorithme, si la taille du texte est n et la taille du motif m, la complexité en $\mathcal{O}((n-m) \times m)$.

2- ALGORITHME DE BOYER-MOORE :

Exemple :
```
texte = "GCATCGCAGAGAGTATACAGTACG"
motif = "GCAGAGAG"
```

index ***i***	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
texte	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
Motif	**G**	**C**	**A**	**G**	**A**	**G**	**A**	**G**																
index ***k***	0	1	2	3	4	5	6	7																

⇨ Code naïf en effectuant les comparaisons en commençant par la fin du motif :

Exemple d'exécution avec le motif de l'exemple :

```python
def dico_lettres(motif):
    dico = {}
    for i in range(len(motif)-1):
        dico[motif[i]] =  i
    return dico
```

```
>>> dico_lettres("GCAGAGAG")
{'G': 5, 'C': 1, 'A': 6}
```

G	**C**	**A**	**G**	**A**	**G**	**A**	**G**
0	1	2	3	4	5	6	7

```python
def BM(texte, motif):
    '''
    renvoie la liste des indices (éventuellement vide)
    des occurrences de la chaîne motif dans la chaîne texte.
    '''
    dico = dico_lettres(motif)
    indices = []
    i = 0
    while i <= len(texte) - len(motif):
        k = len(motif)-1
        while k >= 0 and texte[i+k] == motif[k]:
            k = k - 1
        if k == -1:
            indices.append(i)
            decalage = len(motif)
        else:
            if texte[i+k] in dico and dico[texte[i+k]] < k :
                decalage =  k - dico[texte[i+k]]
            else:
                decalage = k + 1
        i = i + decalage
    return indices
```

```
>>> dico_lettres("GCAGAGAG")
{'G': 5, 'C': 1, 'A': 6}
```

len(motif)-1 =

index *i*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
texte	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G

Motif index *k*

G	**C**	**A**	**G**	**A**	**G**	**A**	**G**
0	1	2	3	4	5	6	7

decalage = k - dico[texte[i+k]]

index *i*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
texte	G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G

Motif index *k*

G	**C**	**A**	**G**	**A**	**G**	**A**	**G**
0	1	2	3	4	5	6	7

decalage = k - dico[texte[i+k]]

Motif index *k*

G	**C**	**A**	**G**	**A**	**G**	**A**	**G**
0	1	2	3	4	5	6	7

⇨ Travail à faire :

- Ecrire le code des fonctions dico_lettres() et BM() dans un fichier que vous nommerez rechercheTextuelleBM.py.
- Rajouter un compteur de comparaisons effectuées.
- Ecrire également dans ce fichier, le code de la fonction recherche_naive() avec 2 boucles while et comparaison en commençant par la fin du motif. Y rajouter aussi un compteur de comparaisons.
- Traiter les cas suivants, à partir du texte correspondant au livre « Le rouge et le noir »
  Et remplir le tableau ci-dessous (résultats à mettre en commentaire dans le fichier .py) :

	Script avec 2 boucles while en partant de la fin du motif	Algorithme de Boyer-Moore
Nombre de comparaisons pour le motif : motif = "a"	61802	61802
Nombre de comparaisons pour le motif : motif = "Julien"	79125	29692
Nombre de comparaisons pour le motif : motif = "peut facilement observer le moment de leur rentrée à l'hôtel"	50682	2235
Nombre de comparaisons pour le motif ci-dessous, à copier-coller	14166	875

```
motif = """
Il faut savoir que le curé de Verrières vieillard de quatre-vingts ans,
mais qui devait à l'air vif de ces montagnes une santé et un caractère
de fer, avait le droit de visiter à toute heure la prison, l'hôpital et
même le dépôt de mendicité. C'était précisément à six heures du matin
que M. Appert qui de Paris était recommandé au curé, avait eu la sagesse
d'arriver dans une petite ville curieuse. Aussitôt il était allé au
presbytère."""
```

- Conclure en répondant aux questions suivantes (en commentaire dans le fichier .py) de manière précise, la réponse attendue doit être argumentée et rédigée :
  - Question 1 : Dans quel cas de figure, l'algorithme de Boyer-Moore est-il plus performant
    L'algorithme de Boyer-Moore est d'autant plus performant que le motif de comparaison est long.

  - Question 2 : Quelle est finalement la complexité de cet algorithme ?
    La complexité d'un algorithme de recherche textuelle se mesure essentiellement par le nombre d'opérations de comparaison de caractères qu'il effectue. L'analyse précise de la complexité de l'algorithme que nous avons programmé est difficile, et dépasse le niveau attendu en enseignement NSI.

Simplement, signalons qu'il est considéré comme un algorithme sous-linéaire : dans des cas favorables, les décalages de la fenêtre sont de l'ordre de la taille m du motif. En fait, l'algorithme n'est même pas tenu de lire l'intégralité du texte : c'est la sous-linéarité.
Dans les cas favorables, on peut estimer un coût de l'ordre de $\frac{n}{m}$, ce qui est évidemment très intéressant.