# TNSI – Recherche Textuelle (algorithme de Boyer-Moore)

## 1. Introduction

La recherche **d'un motif** dans **un texte** est une application importante en informatique. Elle est utilisée dans tous les moteurs de recherche ou des domaines d'études utilisant l'informatique telle que le séquençage de gènes etc …
On peut procéder par la force brute avec un algorithme naïf ou procéder de manière plus sophistiquée en minimisant le nombre de recherches en effectuant un prè-traitement. C'est l'idée de **l'algorithme de Boyer et Moore**.

## 2. Algorithme naïf utilisation de la force brute.

On parcourt le **texte d**u début à la fin (de gauche à droite) caractère par caractère et on vérifie si le **motif** correspond avec la partie du texte en regard**.** Dans ce cas il existe une **occurrence** du motif dans le texte.

Le texte est une chaine de caractères de longueur n. Le motif une chaine de caractères de longueur p.

Bien entendu, si p > n la recherche de motif dans texte échoue. En pratique on a $1 \leqslant p \leqslant n$, et même p beaucoup plus petit que n. Typiquement, on peut chercher un mot ou une phrase dans tout le texte d'un roman

**Travail à faire ouvrir le texte : Le_rouge_et_le_noir.txt**

- **Rechercher la première occurrence du motif : « julien »**
- **Le nombre d'occurrence du motif « Julien »**
- **Ne pas utiliser la méthode find : ex stendhal.find(« Julien »)**

**Rappels ouverture d'un texte :**

```
fichier = open('Le_rouge_et_le_noir.txt', 'r', encoding="utf-8")
stendhal = fichier.read()
fichier.close()
```

**Compléter le programme de recherche de la première occurrence:**

```
def recherche_naive(texte,motif):
    n = len(texte)
    p = len(motif)
    for i in range(********************):
        j = 0
        recherche = True
        while recherche and j < p:
            if texte[*********] != motif[***]:
                recherche = False
            ************
        if recherche :  return i
    return -1
```

**Modifier le programme pour avoir en retour le nombre d'occurrence « Julien »**

# TNSI – Recherche Textuelle (algorithme de Boyer-Moore)

L'algorithme naïf de recherche textuelle fonctionnel mais lent en fonction de la taille du texte. En effet la fonction doit tester n−p+1 positions dans texte et pour chacune effectuer jusqu'à p comparaisons, soit jusqu'à (n-p)×p.

Toutefois, celui-ci peut être amélioré. Certaines comparaisons pourraient être évitées.Nous allons voir comment rendre cette recherche plus efficace avec l'algorithme de Boyer-Moore simplifié.

## 3. BOYER-MOORE (HORSPOOL):

L'algorithme de Boyer et Moore est un algorithme de recherche textuelle très efficace développé en 1977. **Robert Stephen Boyer** et **J Strother Moore** travaillaient alors à l'université d'Austin au Texas en tant qu'informaticiens.

En 1980, **Nigel Horspool** a conçu une **variante simplifiée de l'algorithme de Boyer-Moore**.
C'est cette version qu'on va étudier

Cet algorithme repose sur deux idées :

1. On compare le mot de droite à gauche à partir de sa dernière lettre.
2. On n'avance pas dans le texte caractère par caractère, mais on utilise un décalage dépendant de la dernière comparaison effectuée.

### Explication de l'algorithme :

Nous considérons ici la recherche du motif `mot = 'dab'` dans le texte `texte = 'abracadabra'`.

On commence la recherche à l'index 2 :

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|
| d | a | b | | | | | | | | |

Il n'y a pas de correspondance à la fin du mot : `'r' != 'b'`, donc on avance, mais de combien de caractères avance-t-on. Pour le décider, on utilise le fait que le caractère `'r'` n'apparaît pas dans le mot cherché, donc on peut avancer de `n = len(mot) = 3` caractères sans crainte de rater le mot.

On recherche donc à l'indice 2 + 3 = 5 :

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | d | a | b | | | | | |

Il n'y a pas de correspondance à la fin du mot : `'a' != 'b'`, donc on avance, cependant, cette fois, comme le caractère `'a'` apparaît pas dans le mot cherché en avant-dernière position, on ne peut avancer que de une case pour faire une comparaison en alignant les `'a'`.

On recherche donc à l'indice 5 + 1 = 6 :

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | d | a | b | | | | |

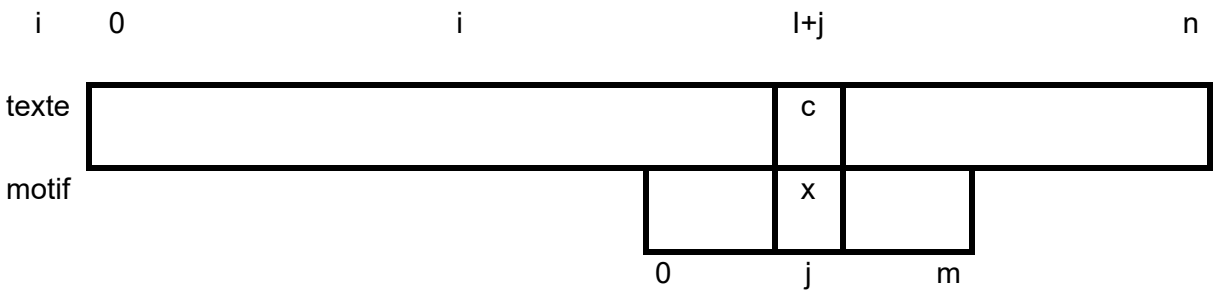
Il n'y a pas de correspondance à la fin du mot : `'d' != 'b'`, donc on avance, cependant, cette fois, comme le caractère `'d'` apparaît dans le mot cherché en avant-avant-dernière position(*première position, mais on doit lire à l'envers !*), on avance de deux cases pour faire une comparaison en alignant les `'d'`.

On recherche donc à l'indice 6 + 2 = 8 :

| a | b | r | a | c | a | d | a | b | r | a |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | d | a | b | | |

Maintenant lorsqu'on effectue les comparaisons à l'envers : les `'b'`, puis les `'a'`, puis les `'d'` correspondent. On a trouvé le mot on renvoie `VRAI`.

De manière générale le décalage est :




On se déplace pour les j descendants de droite à gauche à l'instant qui nous intéresse on est dans le motif à la position j et dans le texte à la position j+i. En face du caractère c du

texte on a positionné le caractère x du motif. C est le premier caractère où texte et motif sont différents.

- Si c est absent du motif on peut décaler i de la longueur du reste du motif soit : j +1
  D = j+1
- Si c apparait dans le motif on parcours le motif de la position j de droite à gauche pour obtenir le premier caractère du motif qui est égal à c . Soit k sa position dans l'intervalle 0 <= i < m-1. Le décalage est alors égal à j-k

  D = j – k

  À défaut, (c'est-à-dire si D $\leqslant$ 0) on se contentera d'un décalage d'une unité, comme dans l'algorithme naïf

**Pré-traitement**

Pour éviter de calculer en permanence les décalages on fait un pré-traitement en remplissant une table de décalage avec m lignes pour j de 0 à m-1 et comme colonne les caractères uniques du motif. On inscrit au croisement la position k du caractère du motif qui serait en face de c. Puis ensuite in faudra calculer le décalage

Sur l'exemple du cours

| j / c | a | b | d |
|---|---|---|---|
| 0 | | | |
| 1 | | | 0 |
| 2 | 1 | | 0 |

D = j -k

k position du caractère dans m

# TNSI – Recherche Textuelle (algorithme de Boyer-Moore)

Sur le même modèle trouver la grille de pré-traitement du motif « chercher »

| j/c | c | h | e | r |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |
| 6 | | | | |
| 7 | | | | |

D = j -k

Avec le motif ‘banane »

| j/c | a | b | e | n |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |
| 5 | | | | |

D = j -k

## 4. Implémentation en Python

Compléter les codes

**Pré-traitement**

- **Table de décalage**

Pour réaliser la table de décalage on utilise une liste avec comme éléments des dictionnaires donnant pour chaque position du motif la position du premier caractère correspondant à cette position si il existe dans le motif sur la gauche caractère de cette position

```
def table_position(motif) :

    assert type(motif ) ==  str

    # creation d'une liste vide
    # la position dans la liste correspond à j
    # pour j on ecrit un dictionnaire avec clé valeur du caractère
et valeur la position du caractère dans le motif avant j
    table = []
    for j in range(len(motif)) :
        dico = {}
        for *** in range( *** ) :

            **********
        table.append(dico)

    return table
```

- **Décalage**

Après avoir déterminer les positions des caractères dans le motif vis-à-vis du caractère en cours qui est différent on calcule le décalage

```
def decalage(table,j,c) :

    """ on utilise la table de décalage défini ci-avant
    lorsque le caractère j est c au lieu du caractère attendu """
    if c in table[j] :
        # c apparait en  table[j][c] et on décale de la
différence  j - k  soit ici j - table[c]
        return ***************

    else :
        # c n'est pas présent dans  m[0 .. m-1] on avance de j+1
        return ***********
```

**Recherche des différentes occurrences**

```
def recherche_boyer_moore(texte, motif) :
    table = table_position(motif)
    i = 0
    while i <= len(texte)-len(motif):
    # d decalage
        d  = 0
        #parcours des caractères de droite à gauche
        for j in range(len(motif)-1,-1,-1):
             # comparaison du texte et du motif position i du texte
            if ************************** :
                 # calcul du décalage pour le caractère en cours
                    d = ************************
                    break  # on sort du for

        if d == 0 : # pas de décalage donc motif en face du texte
            print(" occurence en position : ",i)
            d = 1 # on avance de 1 pour la prochaine occurrence
                  # si on désire la première occurrence on supprime
cette ligne on retourne i et on sort  de la boucle while
         # on avance de d pour la prochaine occurrence
        i += d
```

**exemple**

```
fichier = open('Le_rouge_et_le_noir.txt', 'r', encoding="utf-8")
stendhal = fichier.read()
fichier.close()


# la première occurrence seulement
assert recherche_boyer_moore(stendhal,"Julien") == 26583
```