Activité adidactique

Pour chaque étape de cette activité vous ne pouvez pas revenir en arrière, alors jouez le jeu et assurez vous que vous avez ce qu'il vous faut pour la suite.

Étape 1 : Par équipes de **3** ou **4** élèves, vous allez devoir trouver l'algorithme d'une fonction qui recherche dans un motif dans un texte. On considère acquis que la taille du motif est plus petite que la taille du texte (15 minutes)

Étape 2 : Déroulez votre algorithme sur cet exemple :
TEXTE = "SAUCISSES
MOTIF = "SES"

Étape 2 : Écrire votre fonction au tableau (5 minutes)

Étape 3 : Critiques des fonctions et votes
Pour chaque fonction exceptée la vôtre, vous voterez pour la meilleure. Une justification pourra vous être demandée si votre choix n'est pas évident. (5 minutes)

Étape 4 : Implémentation sur PC et test des fonctions par l'enseignant (20 minutes)

Étape 5 : Discussion sur les pistes d'amélioration (5 minutes ?)

Étape 6 : Bilan (5 minutes)

Recherche textuelle :

La recherche textuelle est au centre du fonctionnement des moteurs de recherche qui parcourent les codes des pages et cherchent des motifs pour les indexer. Elle est aussi utilisée en génétique pour le séquençage de génome. Dans toutes ces situations les textes à parcourir comme les motifs à trouver peuvent être très longs.

Nous avons mis au point ensemble un algorithme de recherche textuelle fonctionnel mais lent. La fonction doit tester N−nN−n positions dans `texte` et pour chacune effectuer jusqu'à N−n comparaisons, soit jusqu'à (N−n)×n

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

La complexité de cet algorithme est dans le pire des cas $O((N-n)\times n)$, c'est une complexité quadratique $O(N^2)$ car souvent $N>n$.

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.

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

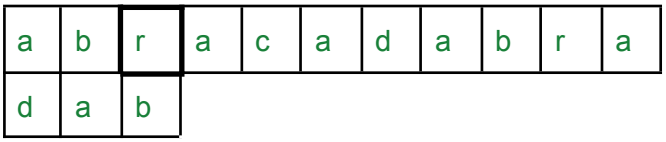
Cet algorithme repose sur deux idées :

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:

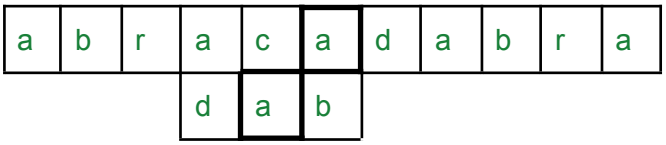
Nous considérons ici la recherche du motif `mot = 'dab'` dans le texte `texte = 'abracadabra'`.

On commence la recherche à l'index 2 :



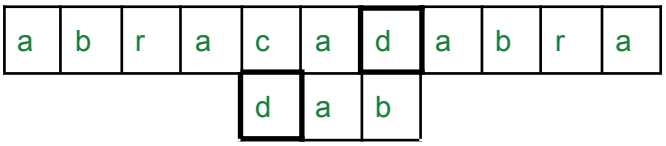
Il n'y a pas de correspondance à la fin du mot : `'r' != 'b'`, donc on avance, mais de combien de caractères avance-t-on. Pour le décider, on utilise le fait que le caractère `'r'` n'apparaît pas dans le mot cherché, donc on peut avancer de `n = len(mot) = 3` caractères sans crainte de rater le mot.

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



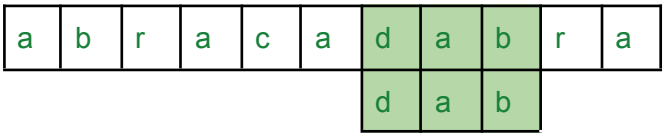
Il n'y a pas de correspondance à la fin du mot : `'a' != 'b'`, donc on avance, cependant, cette fois, comme le caractère `'a'` apparaît pas dans le mot cherché en avant-dernière position, on ne peut avancer que de une case pour faire une comparaison en alignant les `'a'`.

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



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 :



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`.

À vous maintenant!

Implémentation

Pour implémenter efficacement cet algorithme, on va passer par un prétraitement du nom pour facilement accéder au décalage à effectuer. On utilise un dictionnaire pour cela.

```
def pre_traitement(motif):
    # motif : chaîne de caractères

    décalages ← dictionnaire vide
    n ← taille du motif
    pour i allant de 0 à n :
        décalages[motif[i]]   ← n - i -1
    renvoyer décalages

# tests
print(pre_traitement("dab"))
print(pre_traitement("maman"))
```

Maintenant la fonction de recherche :

```
def recherche_motif_boyer(texte, mot):
    # création de notre dictionnaire de décalages
    décalages ← pre_traitement(motif)

    n ← taille du texte
    p ← taille du motif
    i ← p -1
    tant que i < n - p + 1:
        saut ← p
        j ← 0
        recherche ← Vrai
        tant que recherche est vrai et j < p:
            if texte[i - j] != motif[p - 1 - j]:
                recherche ← Faux
                si texte[i] est dans décalages:
                    saut ← décalages[texte[i]]
            j ← j + 1
        si recherche est vrai :
            renvoyer i - p
        i ← i + saut
    renvoyer -1

# Quelques tests
print(recherche_motif_boyer('abracadabra', 'dab')) #True
print(recherche_motif_boyer('abracadabra', 'abra')) #True
print(recherche_motif_boyer('abracadabra', 'obra')) #False
print(recherche_motif_boyer('abracadabra', 'bara')) #False
print(recherche_motif_boyer('maman est là', 'maman')) #True
print(recherche_motif_boyer('bonjour maman', 'maman')) #True
print(recherche_motif_boyer('bonjour maman', 'papa')) #False
```