

# TP - Recherche textuelle

M. Tellene

**Conseil d'organisation :** il vous est conseillé pour ce TP de créer dans votre dossier personnel, un dossier « recherche textuelle ». C'est ce dossier qui contiendra le travail demandé lors de ce TP. De la même manière, il vous est conseillé de faire un fichier python par exercice.

On s'intéresse au problème de la recherche des occurrences d'une chaîne de caractères (appelée *motif*) dans une autre chaîne de caractères (appelée *texte*). Par exemple, il y a deux occurrences du motif "bra" dans le texte "abracadabra". Plus précisément, on va chercher à quelles positions dans le texte le motif apparaît.

## EXERCICE 1

En numérotant les positions à partir de 0, donner les positions des deux occurrences de "bra" dans "abracadabra".

L'objectif de ce TP est d'écrire une fonction `recherche` qui affiche les positions de toutes les occurrences d'un motif dans un texte. Ainsi l'appel à `recherche("bra", "abracadabra")` devra afficher :

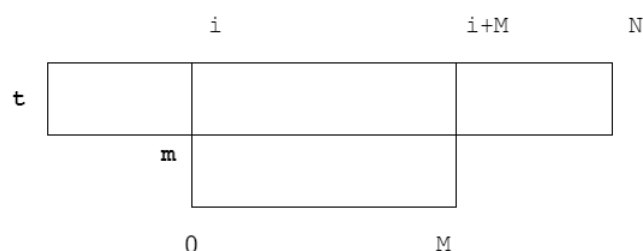
```
occurrence à la position 1
occurrence à la position 8
```

## EXERCICE 2

Écrire les instructions permettant d'avoir :

- la longueur d'une chaîne `s` : .....
- le  $(i+1)$ -ième caractère de `s` : .....
- la sous-chaîne de `s` contenant les caractères `i` inclus à `j` exclus : .....

Durant tout ce TP, on note `m` le motif que l'on recherche et `t` le texte dans lequel on le recherche. On note également `M` la longueur du motif et `N` la longueur du texte. Une première remarque évidente est qu'il ne peut y avoir une occurrence de `m` dans `t` que si  $M \leq N$ . Plus précisément, une occurrence de `m` dans `t` à la position  $i$  est contrainte pour l'inégalité  $0 \leq i \leq N - M$ . Il est utile de représenter une occurrence de `m` dans `t` à la position  $i$  comme ceci :



---

Au dessus, on a représenté les indices des caractères du texte, qui vont de 0 inclus à N exclus. En dessous, on a représenté les indices des caractères du motif, qui vont de 0 inclus à M exclus. S'il y a une occurrence à la position  $i$ , alors les caractères  $t[i], \dots, t[i+M-1]$  du texte coïncident avec les caractères  $m[0], \dots, m[M-1]$  du motif.

### EXERCICE 3

Écrire une fonction `premiere_recherche` qui prend en argument `m` et `t`. Cette fonction devra suivre le fonctionnement décrit précédemment et afficher la position de l'occurrence.

Cette fonction bien que fonctionnel est assez naïve : on va systématiquement construire la sous-chaîne, avec le *slicing*, pour ensuite la comparer à `m`. Or l'extraction de la sous-chaîne a un coût, qui est proportionnel au nombre de caractères.

### EXERCICE 4

- Quel est le coût de la construction de la sous-chaîne? .....
- Combien de fois cette opération est répétée? .....
- Quelle est la complexité de la fonction `premiere_recherche`? .....
- Combien d'opérations fera-t-on si on cherche un motif de 1000 caractères dans un texte de 2000 caractères? .....

On voit bien que cette méthode n'est pas efficace. Pour y remédier, il suffit de comparer directement la chaîne `m` avec la sous-chaîne contenue dans `t` à la position  $i$ , caractère par caractère, sans construire la sous-chaîne. On peut le faire avec une boucle qui parcourt les caractères de `m` et les compare aux caractères de `t`.

### EXERCICE 5

Écrire une fonction `occurrence` qui prend en argument le motif, le texte et la position et qui renvoie `True` si et seulement si une occurrence a été trouvée. La fonction `occurrence` devra suivre le principe énoncé ci-dessus. Cette seconde fonction devra bien évidemment exclure les cas où  $i$  ne désigne pas une position légale dans le texte. Cette fonction devra également comparer les caractères un par un et renvoyer `False` dès qu'il y aura une différence.

Une fois fait, écrire une fonction `deuxieme_recherche` qui prend en argument le motif et le texte. La fonction devra appeler `occurrence` afin d'afficher la position d'une occurrence de `m` dans `t`.

### EXERCICE 6

Expliquer l'intérêt de la deuxième méthode par rapport à la première.

### EXERCICE 7

Dans cet exercice, nous allons comparer le temps d'exécution de nos deux méthodes de recherches. Il faudra, dans un premier temps importer `matplotlib.pyplot`, que vous mettrez sous l'alias `plt`, ainsi que `time` du module `time`. Ensuite, récupérer le code à trous de `evaluation_perfomance.py` à l'adresse suivante : <https://f24.link/cEHxe>.

Compléter ce code et réaliser l'appel ci-dessous, le résultat devrait être un graphique avec deux courbes.

```
1 durees_pr, durees_dr = test()
2 line_plot(durees_pr, durees_dr)
```

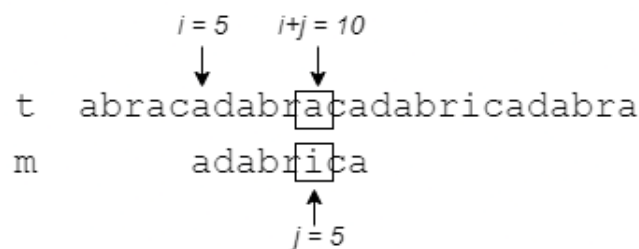
Indiquer à partir de combien de caractères la fonction `deuxieme_recherche` « commence » à être plus performante que `premiere_recherche`. Vous indiquerez également si le test de performance se fait sur le pire ou meilleur cas.

Cette deuxième méthode de recherche effectue une comparaison entre le motif recherché et chacune des sous-chaînes de longueur  $M$  du texte. Cette comparaison est accélérée dans le sens où elle s'interrompt à la première différence trouvée, mais elle doit bien être faite pour tous les indices  $i$  de départ jusqu'à la valeur maximale plausible  $N - M$ .

Une deuxième idée de recherche consiste à ne pas appliquer la comparaison à toutes les sous-chaînes.

Prenons l'exemple suivant, soit le  $m = \text{"bra"}$  et  $t = \text{"bricabrac"}$ . Lorsque l'on compare le troisième caractère de  $m$  avec le caractère correspondant de la sous-chaîne `"bri"` commençant à l'indice  $i = 0$ , on apprend que cette première sous-chaîne n'est pas une occurrence du motif, mais on remarque également que le caractère à l'indice 2 est un `"i"`. Or  $m$  ne contient aucune occurrence de ce caractère. Ainsi, aucune sous-chaîne de  $t$  contenant ce caractère `'i'` ne peut être une occurrence de  $m$ . Il est donc inutile de considérer les sous-chaînes `"ric"` et `"ica"` commençant aux indices  $i = 1$  et  $i = 2$ . On peut donc passer à la sous-chaîne `"cab"` commençant à l'indice  $i = 3$ .

Considérons maintenant la recherche du motif  $m = \text{"adabrica"}$  dans le texte  $t = \text{"abracadabracadabracadabra"}$ , et en particulier la comparaison avec la sous-chaîne `"adabraca"` commençant à l'indice  $i = 5$ . La comparaison du 6<sup>e</sup> caractère (à l'indice  $j = 5$ ) nous apprend que cette sous-chaîne n'est pas une occurrence du motif, mais aussi que le caractère à l'indice  $i + j = 10$  dans le texte est un `"a"`.



Ainsi, on sait qu'à moins de dépasser complètement cette partie du texte, seuls les indices faisant en sorte que ce `"a"` du texte corresponde à l'un des `"a"` du motif sont intéressants. Il y a trois occurrences de `"a"` dans  $m$  : aux indices  $j = 0$ ,  $j = 3$  et  $j = 7$ . Ces trois occurrences sont en correspondance avec le caractère `"a"` à l'indice  $i + j = 10$  dans le texte pour les valeurs respectives 10, 8 et 3 de  $i$ .

A l'inverse, on en déduit qu'il est inutile de considérer les valeurs 4, 5, 6, 7 et 9 de  $i$ , car elles placeraient des caractères différents de `"a"` en correspondance avec le `"a"` situé à l'indice  $i + j = 10$  du texte.

### EXERCICE 8

Parmi les trois valeurs 3, 8 et 10, laquelle choisir comme indice de début de la prochaine sous-

chaîne à prendre en compte ? Vous détaillerez pour chacun des indices pourquoi celui-ci peut être choisi ou non.

En synthétisant les deux exemples précédents, on peut proposer la stratégie suivante :

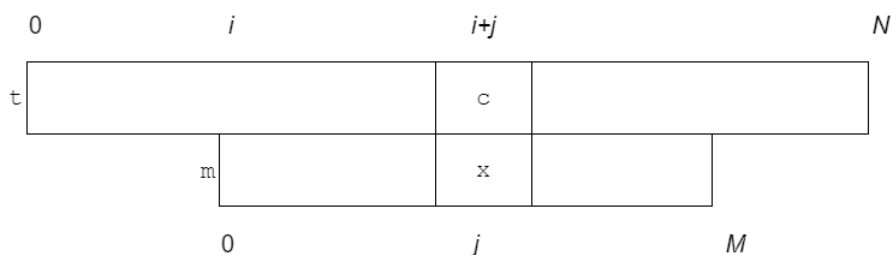
- Si la sous-chaîne commençant à l'indice  $i$  est égale au motif cherché, indiquer  
une ..... et reprendre à l'indice .....
- Sinon, si l'on a observé une différence entre le caractère à l'indice  $j$  du motif et le caractère à l'indice  $i + j = k$  du texte, incrémenter  $i$  jusqu'à ce que :
  - soit  $i$  dépasse .....
  - soit le caractère à l'indice ..... du motif soit égal au caractère ..... du texte

La stratégie que nous venons d'énoncer semble gagner en efficacité par rapport à celle d'origine, puisqu'elle permet de sauter certaines valeurs de  $i$ . Ces sauts ne sont cependant pas gratuits : chacun revient à parcourir une partie du motif, à la recherche de la dernière occurrence passée du caractère observé.

Si l'on recherche le même motif  $m$  dans plusieurs textes différents, ou dans un texte très grand, les mêmes recherches de dernière occurrence d'un certain caractère risquent d'être faites un grand nombre de fois. Une solution possible : faire un pré-traitement du motif afin de pré-calculer les sauts dont on pourra avoir besoin et les enregistrer dans une structure de données adaptée. Au moment où viendra une opportunité de sauter certains indices  $i$ , il suffira de consulter cette structure de données pour savoir directement le nombre d'indices qui peuvent être sautés.

On sacrifie donc un peu de temps avant de commencer la recherche, pour ensuite accélérer la recherche en elle-même. **L'algorithme de Boyer-Moore** réalise cette stratégie avec pré-traitement, en y ajoutant une dernière optimisation. Le principe de cet algorithme est le suivant :

- On va tester l'occurrence de  $m$  dans  $t$  à des positions  $i$  de plus en plus grandes, en partant de  $i = 0$
- Pour une position  $i$  donnée, on va comparer le caractère de  $m$  et de  $t$  de la droite vers la gauche, d'abord  $m[M-1]$  et  $t[i+M-1]$ , puis  $m[M-2]$  et  $t[i+M-2]$ , ...). Ce changement de sens permet d'avancer plus vite dans certains cas
- Si tous les caractères coïncident, on a trouvé une occurrence. Sinon, soit  $j$  l'indice de la première différence, c'est-à-dire le plus grand entier tel que  $0 \leq j \leq M$  et  $m[j] \neq t[i+j]$ . Appelons  $c$  le caractère  $t[i+j]$

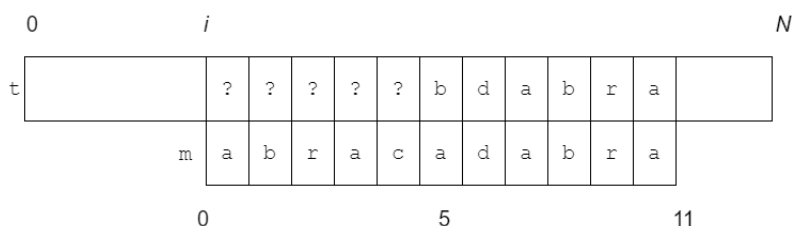


L'idée de l'algorithme de Boyer-Moore consiste à augmenter alors la valeur de  $i$  de

- la grandeur  $j - k$  où  $k$  est le plus grand entier tel que  $0 \leq k \leq j$  et  $m[k] = c$ , si un tel  $k$  existe
- la grandeur de  $j + 1$  sinon

Plutôt que de rechercher un tel  $k$ , on peut pré-calculer une **table de décalages** contenant, à la case indexée par l'entier  $j$  et le caractère  $c$ , le plus grand entier  $k$  tel que  $0 \leq k \leq j$  et  $m[k]=c$  s'il existe, et rien sinon.

Supposons que l'on recherche le motif  $m="abracadabra"$ . On est en train de tester l'occurrence de ce motif à une certaine position  $i$  dans le texte. On procède de droite à gauche, en commençant par la fin du motif. Supposons que les cinq premières comparaisons de caractères ont été positives (les caractères "dabra" coïncident avec le texte). Supposons enfin que le caractère suivant dans le texte ne coïncide pas car il s'agit du caractère "b" alors que le motif a le caractère "a" à cette position.



On consulte alors la table décalages, pour l'indice  $j = 5$  et pour le caractère "b". La table indique la valeur 4, ce qui veut dire qu'il faut décaler le motif de quatre positions vers la droite. Si en revanche le caractère du texte avait été "z", alors la table n'aurait pas contenu d'entrée pour ce caractère, car il n'y a pas d'occurrence de "z" dans les cinq premiers caractères du motif. On aurait alors décalé le motif de  $j + 1 = 6$  positions vers la droite.

La table de décalages est une table à deux clés : l'indice  $j$  du caractère du motif qui diffère et le caractère  $c$  du texte. Pour la première clé, on peut utiliser un tableau de taille  $M$ , pour la seconde, on ne va en revanche pas utiliser un tableau car les caractères possibles sont trop nombreux et la table occuperait trop d'espace mémoire. Seuls les caractères apparaissant dans le motif sont pertinents comme entrées, car il s'agit de décalages amenant un caractère du motif sous celui du texte. Certaines entrées de la table sont vides quand il n'y a pas de décalage possible.

Quelle structure de données serait utile pour faire cette table? .....

Quel serait le type de chacun des éléments? .....

On a représenté ci-contre le table de décalages pour le motif  $m="abracadabra"$ . Il s'agit d'un tableau de taille  $M = 11$ , où chaque élément (chaque ligne) est un dictionnaire dont les clés constituent un sous-ensemble de  $\{a,b,r,c,d\}$ . Dans cette table, on retrouve par exemple l'entrée pour  $j = 5$  et  $c = b$  prise en exemple plus haut, avec la valeur 1. Cette valeur est l'indice de l'occurrence du caractère "b" le plus à droite dans le motif avant l'indice  $j = 5$ . Le décalage s'en déduit comme  $j - 1 = 4$ .

	a	b	r	c	d
0					
1	0				
2	0	1			
3	0	1	2		
4	3	1	2		
5	3	1	2	4	
6	5	1	2	4	
7	5	1	2	4	6
8	7	1	2	4	6
9	7	8	2	4	6
10	7	8	9	4	6

### EXERCICE 9

Construire à la main la table de décalages de Boyer-Moore pour les motifs suivants : "banane", "chercher" et "commissionnee".

---

### EXERCICE 10

Écrire une fonction `table_bm()` qui prend en argument un motif `m`. Cette fonction devra construire et renvoyer la table de décalage du motif. Vérifier les réponses données à l'exercice précédent.

### EXERCICE 11

Écrire une fonction `decalage()` qui prend en argument la table de décalages, l'indice du caractère différent et le caractère lui-même. Cette fonction devra renvoyer le décalage effectué.

### EXERCICE 12

Écrire une fonction `recherche_bm()` qui prend en argument un motif et un texte. Cette fonction doit afficher, s'il y a une occurrence, la position de celle-ci. Cette fonction devra suivre le principe de Boyer-Moore expliqué précédemment.

### EXERCICE 13

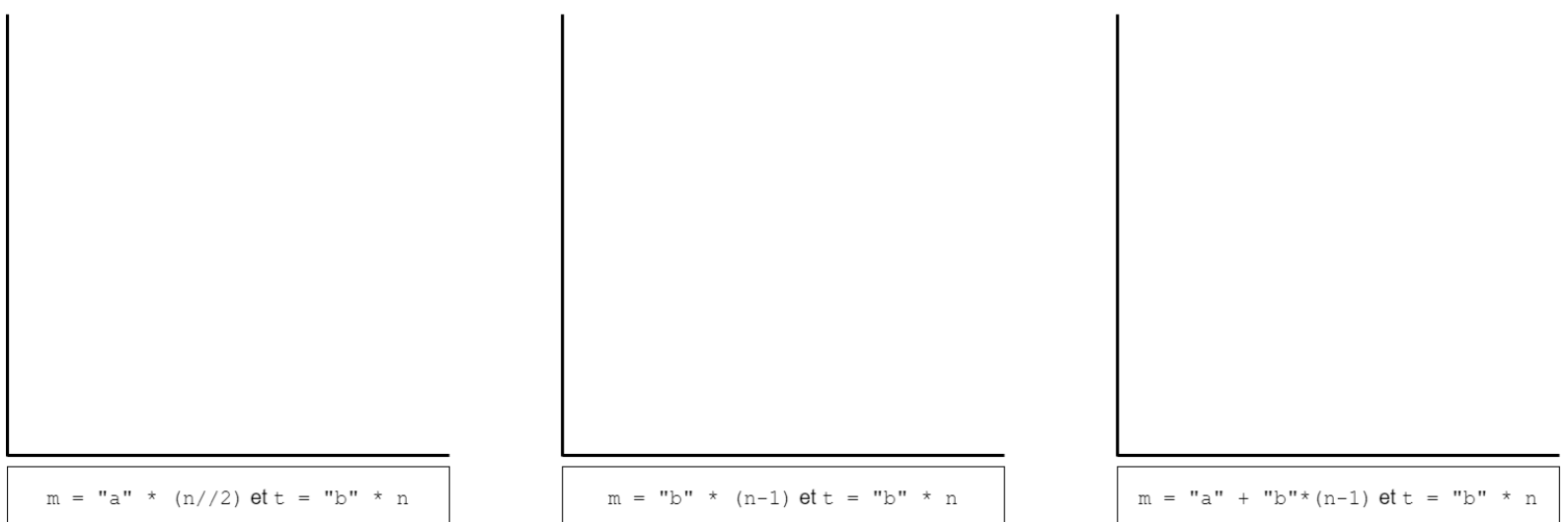
Modifier l'infrastructure de test de performance comme suit :

- faire des sondes de temps pendant la recherche de Boyer-Moore, on ne prendra pas en compte le temps de construction de la table. Ces sondes devront se faire pendant l'exécution de la fonction `recherche_bm` et le temps d'exécution devra être renvoyé par la fonction
- ajouter une courbe pour Boyer-Moore en mettant "Boyer-Moore" comme label de la courbe
- modifier les tableaux de nombre d'éléments (`L` et `x`) pour qu'ils contiennent les éléments suivants : [1000, 2000, 4000, 8000, 16000, 36000, 64000] (si le temps d'exécution est trop long, enlever 64000 et rajouter 500 au début)

Vous réaliserez des tests sur les éléments suivants (`n` est le nombre de caractères, contenu dans les tableaux) :

- `m="a"*(n//2)` et `t="b"*n`
- `m="b"*(n-1)` et `t="b"*n`
- `m="a" + "b"*(n-1)` et `t="b"*n`

Représenter les résultats obtenus ci-dessous :



`m = "a" * (n//2) et t = "b" * n`

`m = "b" * (n-1) et t = "b" * n`

`m = "a" + "b"*(n-1) et t = "b" * n`