Recherche textuelle
Christophe Viroulaud
Terminale - NSI
Algo 27

Recherche textuelle

Christophe Viroulaud

Terminale - NSI

Algo 27

La recherche textuelle est une fonctionnalité intégrée dans tous les logiciels de traitements de texte.

également dans EDI

La recherche textuelle est une fonctionnalité intégrée dans tous les logiciels de traitements de texte.

FIGURE 1 – Des applications multiples

- 4 bases nucléiques : Adénine, Cytosine, Guanine, Thymine,
- ADN humain : 3 milliards de bases répartis sur 23 paires de chromosomes.

Remarque

À titre de comparaison un roman compte environ 500000 caractères.

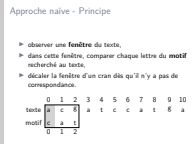
Comment effectuer une recherche textuelle efficace ?

Comment effectuer une recherche textuelle efficace ?

Sommaire

1. Approche naïve
1.1 Principe
1.2 Implémentation
2. Approche plus efficace : Boyer-Moore

Sommaire



Approche naïve - Principe

- observer une **fenêtre** du texte,
- dans cette fenêtre, comparer chaque lettre du **motif** recherché au texte,
- décaler la fenêtre d'un cran dès qu'il n'y a pas de correspondance.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif	c	a	t								
	0	1	2								

FIGURE 2 – Première comparaison : pas de correspondance

Décalage de la fenêtre

FIGURE 3 – Première comparaison : correspondance

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif		c	a	t							
		0	1	2							

FIGURE 3 – Première comparaison : correspondance

FIGURE 4 – Deuxième comparaison : pas de correspondance

Décalage de la fenêtre

0 1 2 3 4 5 6 7 8 9 10
texte a c g a t c c a t g a
motif c a t
0 1 2

FIGURE 4 – Deuxième comparaison : pas de correspondance

Décalage de la fenêtre

Sommaire

1. Approche naïve
1.1 Principe
1.2 Implémentation
2. Approche plus efficace : Boyer-Moore

Sommaire

Implémentation

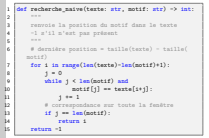
Activité 1 :
1. Écrire la fonction `recherche_naive(texte: str, motif: str)` → `int` qui renvoie la position du *motif* dans le *texte* ou -1 s'il n'est pas présent.
2. Estimer la complexité temporelle de cet algorithme dans le pire des cas : le motif n'est pas présent dans le texte.

Implémentation

Activité 1 :

1. Écrire la fonction **`recherche_naive(texte: str, motif: str)`** → **`int`** qui renvoie la position du *motif* dans le *texte* ou -1 s'il n'est pas présent.
2. Estimer la complexité temporelle de cet algorithme dans le pire des cas : le motif n'est pas présent dans le texte.

Correction



Correction

```python
def recherche_naive(texte: str, motif: str) -> int:
    """
    renvoie la position du motif dans le texte
    -1 s'il n'est pas présent
    """
    # dernière position = taille(texte) - taille(
    motif)
    for i in range(len(texte)-len(motif)+1):
        j = 0
        while j < len(motif) and
                motif[j] == texte[i+j]:
            j += 1
        # correspondance sur toute la fenêtre
        if j == len(motif):
            return i
    return -1
```

$O(P.S)$

Correction

Imaginons le cas :

	0	1	2	3	4	5	6	7	8	9	10
texte	a	a	a	a	a	a	a	a	a	a	t
motif	a	a	t								
	0	1	2								

- On vérifie toute la fenêtre à chaque fois.
- À chaque **non correspondance** la fenêtre avance de 1.
- La complexité dépend de la taille du texte et de celle du motif.

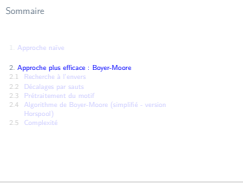
Correction

Imaginons le cas :

	0	1	2	3	4	5	6	7	8	9	10
texte	a	a	a	a	a	a	a	a	a	a	t
motif	a	a	t								
	0	1	2								

- On vérifie toute la fenêtre à chaque fois.
- À chaque **non correspondance** la fenêtre avance de 1.
- La complexité dépend de la taille du texte et de celle du motif.

Sommaire



Sommaire

1. évoquera la complexité de Boyer-Moore en fin de cours
2. version Horspool est simplifiée mais pas forcément aussi efficace que BM

Boyer-Moore
- 1970 : algorithme de Knuth-Morris-Pratt. $O(T + M)$

Boyer-Moore

- 1970 : algorithme de Knuth-Morris-Pratt. $O(T + M)$

1. évoquera la complexité de Boyer-Moore en fin de cours
2. version Horspool est simplifiée mais pas forcément aussi efficace que BM

Boyer-Moore
- 1970 : algorithme de Knuth-Morris-Pratt. $O(T+M)$
- 1977 : algorithme de Boyer-Moore.
  - meilleur des cas : $O(T/M)$.
  - pire des cas : $O(T+M)$

Boyer-Moore

- 1970 : algorithme de Knuth-Morris-Pratt. $O(T+M)$
- 1977 : algorithme de Boyer-Moore.
  - meilleur des cas : $O(T/M)$
  - pire des cas : $O(T+M)$

1. évoquera la complexité de Boyer-Moore en fin de cours
2. version Horspool est simplifiée mais pas forcément aussi efficace que BM

Boyer-Moore

- 1970 : algorithme de Knuth-Morris-Pratt. $O(T + M)$
- 1977 : algorithme de Boyer-Moore.
  - meilleur des cas : $O(T/M)$
  - pire des cas : $O(T + M)$
- 1980 : Horspool propose une version simplifiée de l'algorithme de Boyer-Moore. $O(T)$

Boyer-Moore

- 1970 : algorithme de Knuth-Morris-Pratt. $O(T + M)$
- 1977 : algorithme de Boyer-Moore.
  - meilleur des cas : $O(T/M)$
  - pire des cas : $O(T + M)$
- 1980 : Horspool propose une version simplifiée de l'algorithme de Boyer-Moore. $O(T)$

Recherche à l'envers

La première idée de cet algorithme est de commencer la recherche **en partant de la fin du motif**.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif	c	a	t								
	0	1	2								

FIGURE 5 – Première comparaison : pas de correspondance

Recherche à l'envers

La première idée de cet algorithme est de commencer la recherche **en partant de la fin du motif**.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif	c	a	t								
	0	1	2								

FIGURE 5 – Première comparaison : pas de correspondance

Remarque
Pour l'instant cette approche ne semble par apporter d'amélioration par rapport à l'algorithme précédent.

Remarque

Pour l'instant cette approche ne semble par apporter d'amélioration par rapport à l'algorithme précédent.

Sommaire

Sommaire

Décalage par sauts

Le motif ne contient pas la lettre **g** (la dernière lettre de la fenêtre).

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif	c	a	t								
	0	1	2								

Figure 6 – Comparaisons inutiles

Décalage par sauts

Le motif ne contient pas la lettre **g** (la dernière lettre de la fenêtre).

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif	c	a	t								
	0	1	2								

Figure 6 – Comparaisons inutiles

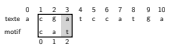


FIGURE 7 – Comparaison inutile

		0	1	2	3	4	5	6	7	8	9	10
texte	a		c	g	a	t	c	c	a	t	g	a
motif			c	a	t							
		0	1	2								

FIGURE 7 – Comparaison inutile

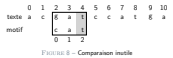


FIGURE 8 – Comparaison inutile

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif			c	a	t						
			0	1	2						

FIGURE 8 – Comparaison inutile

On peut donc directement décaler le motif à l'indice 3 du texte.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif				c	a	t					
				0	1	2					

FIGURE 9 – Décalage par saut

On peut donc directement décaler le motif à l'indice 3 du texte.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif				c	a	t					
				0	1	2					

FIGURE 9 – Décalage par saut

On n'observe pas de correspondance par contre la lettre c existe dans le motif. On va donc le décaler pour les faire coïncider.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif				c	a	t					
				0	1	2					

Figure 10 – Nouvelle situation

On n'observe pas de correspondance par contre la lettre **c** existe dans le motif. On va donc le décaler pour les faire coïncider.

	0	1	2	3	4	5	6	7	8	9	10
texte	a	c	g	a	t	c	c	a	t	g	a
motif				c	a	t					
				0	1	2					

Figure 10 – Nouvelle situation

FIGURE 11 – Décalage par saut

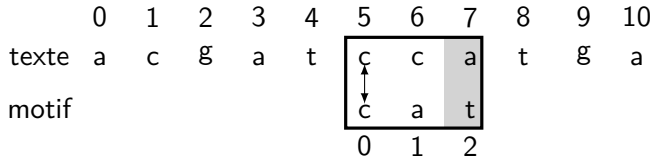


FIGURE 11 – Décalage par saut

À retenir
On décale la position de recherche dans le texte en fonction de la dernière lettre de la fenêtre.

1. et de sa présence éventuelle dans le motif
2. il existe variante *bad char* : on décale en fonction du caractère qui ne correspond pas

À retenir

On décale la position de recherche dans le texte en fonction de la dernière lettre de la fenêtre.

Sommaire

Sommaire

Prétraitement du motif
À retenir
Pour pouvoir décaler par saut, il faut connaître la dernière position de chaque lettre dans le motif. Le prétraitement consiste à calculer le décalage à appliquer pour amener chaque caractère du motif à la place du dernier caractère.
texte a c g a t c c a t g a
motif c a t
2 1
Figure 12 – Calculs des décalages

Prétraitement du motif

À retenir

Pour pouvoir décaler par saut, il faut connaître la dernière position de chaque lettre dans le motif. Le prétraitement consiste à calculer le décalage à appliquer pour amener chaque caractère du motif à la place du dernier caractère.

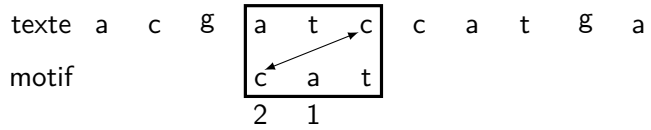
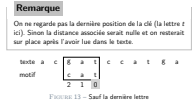


Figure 12 – Calculs des décalages

attention t pourrait être présent ailleurs dans motif → on prend en compte alors

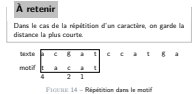


Remarque

On ne regarde pas la dernière position de la clé (la lettre *t* ici). Sinon la distance associée serait nulle et on resterait sur place après l'avoir lue dans le texte.

texte	a	c	g	a	t	c	c	a	t	g	a
motif			c	a	t						
			2	1	0						

FIGURE 13 – Sauf la dernière lettre



on fait coïncider le premier t du motif avec la dernière lettre de la fenêtre

À retenir

Dans le cas de la répétition d'un caractère, on garde la distance la plus courte.

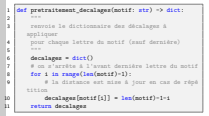
texte	a	c	g	a	t	c	c	a	t	g	a
motif	t	a	c	a	t						
	4		2	1							

FIGURE 14 – Répétition dans le motif

Activité 2 : Écrire la fonction `pretraitement_decalages(motif: str) → dict` qui associe chaque lettre du motif (sauf la dernière) à son décalage.

Activité 2 : Écrire la fonction `pretraitement_decalages(motif: str)` $\rightarrow$ `dict` qui associe chaque lettre du motif (sauf la dernière) à son décalage.

Correction



Correction

```python
def pretraitement_decalages(motif: str) -> dict:
    """
    renvoie le dictionnaire des décalages à appliquer
    pour chaque lettre du motif (sauf dernière)
    """
    decalages = dict()
    # on s'arrête à l'avant dernière lettre du motif
    for i in range(len(motif)-1):
        # la distance est mise à jour en cas de répétition
        decalages[motif[i]] = len(motif)-1-i
    return decalages
```

Sommaire
1. Approche naïve
2. Approche plus efficace : Boyer-Moore
2.1. Recherche à l'envers
2.2. Décalages par sauts
2.3. Prétraitement du motif
2.4 Algorithme de Boyer-Moore (simplifié - version Horspool)
2.5. Complexité

Sommaire

Horspool : en 1980, version simplifiée de Boyer-Moore ; il existe plusieurs versions.

Algorithme de Boyer-Moore

L'algorithme de Boyer-Moore s'écrit alors :

```
Créer le tableau des décalages
Tant qu'on n'est pas à la fin du texte
    Comparer le motif à la position du texte
    Si le motif est présent
        Renvoyer la position
    Sinon
        Décaler la fenêtre
Renvoyer -1 si le motif n'est pas présent
```

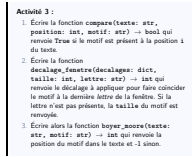
Code 1 – Algorithme de Boyer-Moore (version Horspool)

Algorithme de Boyer-Moore

L'algorithme de Boyer-Moore s'écrit alors :

```
Créer le tableau des décalages
Tant qu'on n'est pas à la fin du texte
    Comparer le motif à la position du texte
    Si le motif est présent
        Renvoyer la position
    Sinon
        Décaler la fenêtre
Renvoyer -1 si le motif n'est pas présent
```

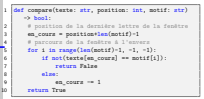
Code 1 – Algorithme de Boyer-Moore (version Horspool)


Activité 3 :
1. Écrire la fonction `compare(texte: str, position: int, motif: str) → bool` qui renvoie `True` si le motif est présent à la position `i` du texte.
2. Écrire la fonction `decalage_fenetre(decalages: dict, taille: int, lettre: str) → int` qui renvoie le décalage à appliquer pour faire coïncider le motif à la dernière *lettre* de la fenêtre. Si la lettre n'est pas présente, la `taille` du motif est renvoyée.
3. Écrire alors la fonction `boyer_moore(texte: str, motif: str) → int` qui renvoie la position du motif dans le texte et -1 sinon.

Activité 3 :

1. Écrire la fonction `compare(texte: str, position: int, motif: str) → bool` qui renvoie `True` si le motif est présent à la position `i` du texte.
2. Écrire la fonction `decalage_fenetre(decalages: dict, taille: int, lettre: str) → int` qui renvoie le décalage à appliquer pour faire coïncider le motif à la dernière *lettre* de la fenêtre. Si la lettre n'est pas présente, la `taille` du motif est renvoyée.
3. Écrire alors la fonction `boyer_moore(texte: str, motif: str) → int` qui renvoie la position du motif dans le texte et -1 sinon.

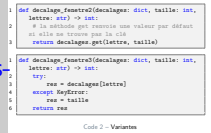
Correction



Correction

```python
def compare(texte: str, position: int, motif: str)
   -> bool:
    # position de la dernière lettre de la fenêtre
    en_cours = position+len(motif)-1
    # parcours de la fenêtre à l'envers
    for i in range(len(motif)-1, -1, -1):
        if not(texte[en_cours] == motif[i]):
            return False
        else:
            en_cours -= 1
    return True
```

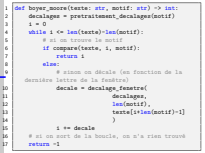
```python
def decalage_fenetre(decalages: dict, taille: int,
    lettre: str) -> int:
    for cle, val in decalages.items():
        if cle == lettre:
            return val
    # si la lettre n'est pas dans le dico (= le
    motif)
    return taille
```



```python
def decalage_fenetre2(decalages: dict, taille: int,
    lettre: str) -> int:
    # la méthode get renvoie une valeur par défaut
    si elle ne trouve pas la clé
    return decalages.get(lettre, taille)
```

```python
def decalage_fenetre3(decalages: dict, taille: int,
    lettre: str) -> int:
    try:
        res = decalages[lettre]
    except KeyError:
        res = taille
    return res
```

Code 2 – Variantes



```python
def boyer_moore(texte: str, motif: str) -> int:
    decalages = pretraitement_decalages(motif)
    i = 0
    while i <= len(texte)-len(motif):
        # si on trouve le motif
        if compare(texte, i, motif):
            return i
        else:
            # sinon on décale (en fonction de la dernière lettre de la fenêtre)
            decale = decalage_fenetre(
                            decalages,
                            len(motif),
                            texte[i+len(motif)-1]
                            )
            i += decale
    # si on sort de la boucle, on n'a rien trouvé
    return -1
```

Sommaire

1. Approche naïve
2. Approche plus efficace : Boyer-Moore
2.1. Recherche à l'envers
2.2. Décalages par sauts
2.3. Prétraitement du motif
2.4. Algorithme de Boyer-Moore (simplifié - version Horspool)
2.5. Complexité

Sommaire

Complexité

Intuitivement l'algorithme semble plus rapide que la version naïve car il ne teste pas toutes les lettres du texte.

a a a a b a a a a b a a a a b
c c c c c

Figure 15 – Un cas représentatif

Complexité

Intuitivement l'algorithme semble plus rapide que la version naïve car il ne teste pas toutes les lettres du texte.

a a a a b a a a a b a a a a b
c c c c c

Figure 15 – Un cas représentatif

Figure 16 – Algorithme naïf

Observation

L'algorithme naïf effectue 10 décalages.

a a a a b a a a a b a a a a b
c c c c c
a a a a b a a a a b a a a a b
c c c c c

Figure 16 – Algorithme naïf

Observation

L'algorithme naïf effectue 10 décalages.

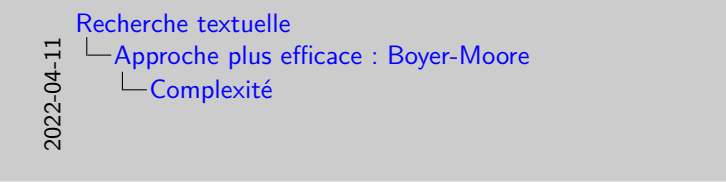


FIGURE 17 – Algorithme de Boyer-Moore

Observation

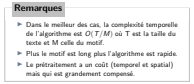
L'algorithme de Boyer-Moore-Horspool effectue 3 décalages.

a a a a b a a a a b a a a a b
c c c c c
a a a a b a a a a b a a a a b
c c c c c

FIGURE 17 – Algorithme de Boyer-Moore

Observation

L'algorithme de Boyer-Moore-Horspool effectue 3 décalages.



1. complexité sous-linéaire !
2. Naïf $O(T.M)$
3. Complexité moyenne : $O(3.M)$ démontrée par Richard Cole en 1991.
4. coût spatial si alphabet grand.

Remarques

- Dans le meilleur des cas, la complexité temporelle de l'algorithme est $O(T/M)$ où T est la taille du texte et M celle du motif.
- Plus le motif est long plus l'algorithme est rapide.
- Le prétraitement a un coût (temporel et spatial) mais qui est grandement compensé.

.

Cas critique

FIGURE 18 – Cas critique

Cas critique

texte t a a a a a a a a a a

motif a a a a a a a a

texte t a a a a a a a a a a

motif a a a a a a a a

FIGURE 18 – Cas critique

Remarque
En plus d'un traitement du mauvais caractère, la version complète de l'algorithme de Boyer-Moore recherche des bons suffixes dans le motif.

Remarque

En plus d'un traitement du mauvais caractère, la version complète de l'algorithme de Boyer-Moore recherche des bons suffixes dans le motif.