Recherche textuelle
Christophe Viroulaud
Terminale - NSI

Recherche textuelle

Christophe Viroulaud

Terminale - NSI

Sommaire

1. Problématique
2. Approche naïve
3. Approche plus efficace : Boyer-Moore

Sommaire

également dans EDI

Problématique

Une fonctionnalité intégrée dans tous les logiciels de traitements de texte.

Problématique

Une fonctionnalité intégrée dans tous les logiciels de traitements de texte.

roman = 500 000 caractères

Des applications multiples

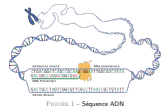


Figure 1 – Séquence ADN

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

Des applications multiples

Figure 1 – Séquence ADN

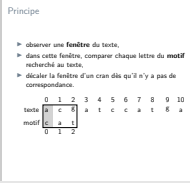
- 4 bases nucléiques : Adénine, Cytosine, Guanine, Thymine,
- ADN humain : 3 milliards de bases répartis sur 23 paires de chromosomes.

Comment effectuer une recherche textuelle efficace ?

Comment effectuer une recherche textuelle efficace ?

Sommaire

Sommaire



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								

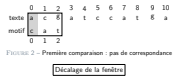


FIGURE 2 – Première comparaison : pas de correspondance

Décalage de la fenêtre

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

	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

Sommaire

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

Code 1 – Approche naïve

Correction

```
1  def recherche_naive(texte: str, motif: str) -> int:
2      """
3      renvoie la position du motif dans le texte
4      -1 s'il n'est pas présent
5      """
6      # dernière position = taille(texte) - taille(motif)
7      for i in range(len(texte)-len(motif)+1):
8          j = 0
9          while (j < len(motif)) and (motif[j] == texte[i+j]):
10             j += 1
11         if j == len(motif): # correspondance sur toute la fenê
       tre
12             return i
13     return -1
```

Code 1 – Approche naïve

$O(P.S)$

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 de la fenêtre 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 de la fenêtre 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.

Boyer-Moore

- 1970 : algorithme de Knuth-Morris-Pratt. $O(T + M)$
- 1977 : algorithme de Boyer-Moore.

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

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.
- 1980 : Horspool propose une version simplifiée de l'algorithme de Boyer-Moore. $O(T)$

Recherche à l'envers

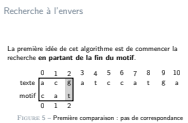


FIGURE 5 – Première comparaison : pas de correspondance

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

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

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).

FIGURE 6 – Comparaisons inutiles

Décalage par sauts

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

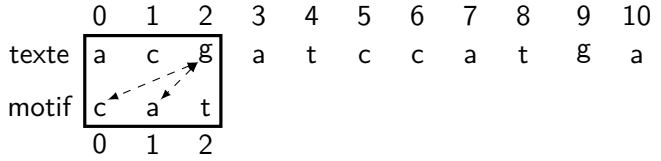


FIGURE 6 – Comparaisons inutiles

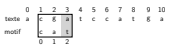


FIGURE 7 – Comparaison inutile

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

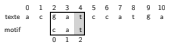


FIGURE 8 – Comparaison inutile

	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 (figure 9).

	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 (figure 9).

	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.

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.

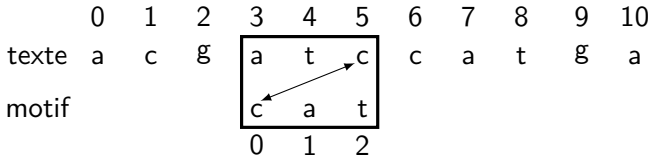


Figure 10 – Nouvelle situation

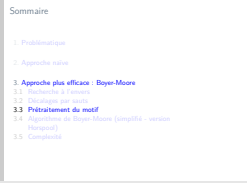
FIGURE 11 – Décalage par saut

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

Prétraitement du motif

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

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.

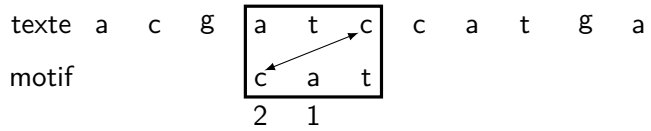
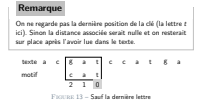


FIGURE 12 – Calculs des décalages

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

Autre exemple

Figure 14 – Répétition dans le motif

Autre exemple

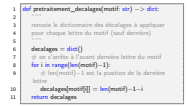
on fait coïncider le premier t du motif avec la dernière lettre de la fenêtre

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)` $\rightarrow$ `dict` qui associe chaque lettre du motif (sauf la dernière) à son décalage.

Correction



S'il y a répétition (slide précédent) le dictionnaire est mis à jour (ligne 10).

Correction

```
1  def pretraitement_decalages(motif: str) -> dict:
2      """
3      renvoie le dictionnaire des décalages à appliquer
4      pour chaque lettre du motif (sauf dernière)
5      """
6      decalages = dict()
7      # on s'arrête à l'avant dernière lettre du motif
8      for i in range(len(motif)-1):
9          # len(motif)-1 est la position de la dernière
       lettre
10         decalages[motif[i]] = len(motif)-1-i
11     return decalages
```

Sommaire
1. Problématique
2. Approche naïve
3. Approche plus efficace : Boyer-Moore
3.1 Recherche à l'envers
3.2 Décalages par sauts
3.3 Prétraitement du motif
3.4 Algorithme de Boyer-Moore (simplifié - version Horspool)
3.5 Complexité

Sommaire

Algorithme de Boyer-Moore

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

```
1 Créer le tableau des décalages
2 Tant qu'on n'est pas à la fin du texte
3     Comparer le motif à la position du texte
4     Si le motif est présent
5         Renvoyer la position
6     Sinon
7         Décaler la fenêtre
8 Renvoyer −1 si le motif n'est pas présent
```

Code 2 – Algorithme de Boyer-Moore (version Horspool)

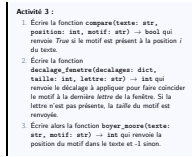
Horspool : en 1980, version simplifiée de Boyer-Moore ; il existe plusieurs versions.

Algorithme de Boyer-Moore

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

```
1 Créer le tableau des décalages
2 Tant qu'on n'est pas à la fin du texte
3     Comparer le motif à la position du texte
4     Si le motif est présent
5         Renvoyer la position
6     Sinon
7         Décaler la fenêtre
8 Renvoyer −1 si le motif n'est pas présent
```

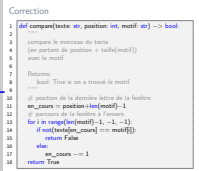
Code 2 – Algorithme de Boyer-Moore (version Horspool)


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)`** $\rightarrow$ **`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)`** $\rightarrow$ **`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)`** $\rightarrow$ **`int`** qui renvoie la position du motif dans le texte et -1 sinon.


Correction

```
def compare(texte: str, position: int, motif: str) -> bool:
    """
    compare le morceau du texte
    (en partant de position + taille(motif))
    avec le motif

    Returns:
        bool: True si on a trouvé le motif
    """
    # 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
```

Correction

```
def compare(texte: str, position: int, motif: str) -> bool:
    """
    compare le morceau du texte
    (en partant de position + taille(motif))
    avec le motif

    Returns:
        bool: True si on a trouvé le motif
    """
    # 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
```

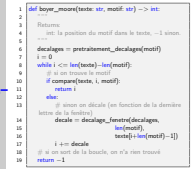
```python
def decalage_fenetre(decalages: dict, taille: int, lettre: str) -> int:
    """
    renvoie la valeur du décalage à appliquer.
    si la lettre n'est pas dans le tableau
    c'est la taille du motif qui est appliqué

    Args:
        decalages (dict): dico des décalages
        taille (int): taille du motif (= décalage max)
        lettre (str): dernière lettre de la fenêtre

    Returns:
        int: décalage à appliquer
    """
    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)


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



```python
def boyer_moore(texte: str, motif: str) -> int:
    """
    Returns:
        int: la position du motif dans le texte, -1 sinon.
    """
    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

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

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

FIGURE 17 – Algorithme de Boyer-Moore

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

Activité 4 : Compter le nombre d'itérations de la recherche avec l'algorithme naïf puis celui de Boyer-Moore.

Activité 4 : Compter le nombre d'itérations de la recherche avec l'algorithme naïf puis celui de Boyer-Moore.

Correction
- Algorithme naïf : 10 décalages,
- Algorithme de Boyer-Moore : 3 décalages.

Correction

- Algorithme naïf : 10 décalages,
- Algorithme de Boyer-Moore : 3 décalages.

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

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.

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.

1. coût spatial si alphabet grand.

Le prétraitement a un coût (temporel et spatial) mais qui est grandement compensé.

Le prétraitement a un coût (temporel et spatial) mais qui est grandement compensé.

La version plus complexe de l'algorithme recherche des sous-motifs dans le motif (good suffix).

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

Code complet
Les programmes du cours sont téléchargeables ici.

Code complet

Les programmes du cours sont téléchargeables ici.