

# C19 Algorithmes des textes

## 1. Position du problème

### Définitions et notations

On s'intéresse au problème de la recherche d'une chaîne de caractères appelée **motif** dans une autre chaîne de caractères appelée **texte**. Plus précisément, on veut lister toutes les occurrences (par leur position) du motif dans le texte.

On notera :

# C19 Algorithmes des textes

## 1. Position du problème

### Définitions et notations

On s'intéresse au problème de la recherche d'une chaîne de caractères appelée **motif** dans une autre chaîne de caractères appelée **texte**. Plus précisément, on veut lister toutes les occurrences (par leur position) du motif dans le texte.

On notera :

- $m$  le motif et  $l_m$  sa longueur

# C19 Algorithmes des textes

## 1. Position du problème

### Définitions et notations

On s'intéresse au problème de la recherche d'une chaîne de caractères appelée **motif** dans une autre chaîne de caractères appelée **texte**. Plus précisément, on veut lister toutes les occurrences (par leur position) du motif dans le texte.

On notera :

- $m$  le motif et  $l_m$  sa longueur
- $t$  le texte et  $l_t$  sa longueur

# C19 Algorithmes des textes

## 1. Position du problème

### Définitions et notations

On s'intéresse au problème de la recherche d'une chaîne de caractères appelée **motif** dans une autre chaîne de caractères appelée **texte**. Plus précisément, on veut lister toutes les occurrences (par leur position) du motif dans le texte.

On notera :

- $m$  le motif et  $l_m$  sa longueur
- $t$  le texte et  $l_t$  sa longueur

D'autre part, parfois le problème se réduira à déterminer si  $m$  est présent ou non dans  $t$ , ou encore on cherchera uniquement la première occurrence.

# C19 Algorithmes des textes

## 1. Position du problème

### Définitions et notations

On s'intéresse au problème de la recherche d'une chaîne de caractères appelée **motif** dans une autre chaîne de caractères appelée **texte**. Plus précisément, on veut lister toutes les occurrences (par leur position) du motif dans le texte.

On notera :

- $m$  le motif et  $l_m$  sa longueur
- $t$  le texte et  $l_t$  sa longueur

D'autre part, parfois le problème se réduira à déterminer si  $m$  est présent ou non dans  $t$ , ou encore on cherchera uniquement la première occurrence.

### Exemple

La recherche du motif  $m=\text{exe}$  ( $l_m = 3$ ) dans le texte  $t=\text{un excellent exemple et un exercice extraordinaire}$  ( $l_t = 50$ ) doit produire la liste d'occurrences :  $[13; 27]$ .

# C19 Algorithmes des textes

## 1. Position du problème

### Définitions et notations

On s'intéresse au problème de la recherche d'une chaîne de caractères appelée **motif** dans une autre chaîne de caractères appelée **texte**. Plus précisément, on veut lister toutes les occurrences (par leur position) du motif dans le texte.

On notera :

- $m$  le motif et  $l_m$  sa longueur
- $t$  le texte et  $l_t$  sa longueur

D'autre part, parfois le problème se réduira à déterminer si  $m$  est présent ou non dans  $t$ , ou encore on cherchera uniquement la première occurrence.

### Exemple

La recherche du motif  $m=\text{exe}$  ( $l_m = 3$ ) dans le texte  $t=\text{un excellent exemple et un exercice extraordinaire}$  ( $l_t = 50$ ) doit produire la liste d'occurrences : [13 ; 27].

un\_excellent\_exemple\_et\_un\_exercice\_extraordinaire  
0 13 27 49

### Recherche naïve

Pour rechercher si un motif  $m$  se trouve dans un texte  $t$ , on peut :

- 1 parcourir chaque caractère de  $t$  jusqu'à celui d'indice  $?$  inclus (indice de la dernière occurrence possible) :

indice dans le texte	0	...	?	$l_t - 1$
indice dans le motif			0	$l_m - 1$

### Recherche naïve

Pour rechercher si un motif  $m$  se trouve dans un texte  $t$ , on peut :

- 1 parcourir chaque caractère de  $t$  jusqu'à celui d'indice  $l_t - l_m$  inclus (indice de la dernière occurrence possible) :

indice dans le texte	0	...	$l_t - l_m$	$l_t - 1$
indice dans le motif			0	$l_m - 1$



### Recherche naïve

Pour rechercher si un motif  $m$  se trouve dans un texte  $t$ , on peut :

- 1 parcourir chaque caractère de  $t$  jusqu'à celui d'indice  $l_t - l_m$  inclus (indice de la dernière occurrence possible) :

indice dans le texte	0	...	$l_t - l_m$	$l_t - 1$
indice dans le motif			0	$l_m - 1$

- 2 si le caractère correspond au premier caractère du motif  $m$ , alors on avance dans le motif tant que les caractères coïncident.

### Recherche naïve

Pour rechercher si un motif  $m$  se trouve dans un texte  $t$ , on peut :

- 1 parcourir chaque caractère de  $t$  jusqu'à celui d'indice  $l_t - l_m$  inclus (indice de la dernière occurrence possible) :

indice dans le texte	0	...	$l_t - l_m$	$l_t - 1$
indice dans le motif			0	$l_m - 1$

- 2 si le caractère correspond au premier caractère du motif  $m$ , alors on avance dans le motif tant que les caractères coïncident.
- 3 si on atteint la fin du motif, alors  $m$  se trouve dans  $t$ . Sinon on passe au caractère suivant de  $t$ .

### Recherche naïve

Pour rechercher si un motif  $m$  se trouve dans un texte  $t$ , on peut :

- 1 parcourir chaque caractère de  $t$  jusqu'à celui d'indice  $l_t - l_m$  inclus (indice de la dernière occurrence possible) :

indice dans le texte	0	...	$l_t - l_m$	$l_t - 1$
indice dans le motif			0	$l_m - 1$

- 2 si le caractère correspond au premier caractère du motif  $m$ , alors on avance dans le motif tant que les caractères coïncident.
- 3 si on atteint la fin du motif, alors  $m$  se trouve dans  $t$ . Sinon on passe au caractère suivant de  $t$ .

### Exemple

Visualisation en ligne du fonctionnement de l'algorithme

### Implémentation en OCaml

On renvoie la *liste* de toutes les occurrences : `naive : string -> string -> int list`

### Implémentation en OCaml

On renvoie la *liste* de toutes les occurrences : `naive : string -> string -> int list`

```
1 let naive motif texte =
2   let lm = String.length motif in
3   let lt = String.length texte in
4   let occ = ref [] in
5   for it=0 to lt-lm do
6     let im = ref 0 in
7     while (!im<lm && motif.[!im]=texte.[it+ !im]) do
8       im := !im +1;
9     done;
10    if (!im=lm) then occ := it::!occ
11    done;
12  !occ;;
```

### Implémentation en OCaml

Dans le cas où on teste simplement la présence, on peut provoquer une sortie anticipée de la boucle `for` à l'aide d'une exception.

### Implémentation en OCaml

Dans le cas où on teste simplement la présence, on peut provoquer une sortie anticipée de la boucle `for` à l'aide d'une exception.

```
1  let present motif texte =
2    let lm = String.length motif in
3    let lt = String.length texte in
4    try
5      for it=0 to lt-lm do
6        let im = ref 0 in
7        while (!im<lm && motif.[!im]=texte.[it+ !im]) do
8          im := !im +1;
9        done;
10       if (!im=lm) then raise Exit (* Exit est prédéfinie *)
11     done;
12     false
13   with Exit -> true;;
```

### Coût de la recherche simple

En notant  $l_m$  la longueur du motif et  $l_t$  celle de la chaîne :



### Coût de la recherche simple

En notant  $l_m$  la longueur du motif et  $l_t$  celle de la chaîne :

- La boucle **for** est parcourue au plus  $l_t - l_m + 1$  fois

### Coût de la recherche simple

En notant  $l_m$  la longueur du motif et  $l_t$  celle de la chaîne :

- La boucle **for** est parcourue au plus  $l_t - l_m + 1$  fois
- Pour chacun de ces parcours, la boucle **while** interne est parcourue au plus  $l_m$  fois

### Coût de la recherche simple

En notant  $l_m$  la longueur du motif et  $l_t$  celle de la chaîne :

- La boucle **for** est parcourue au plus  $l_t - l_m + 1$  fois
- Pour chacun de ces parcours, la boucle **while** interne est parcourue au plus  $l_m$  fois

Au plus, l'algorithme effectue donc  $l_m(l_t - l_m + 1)$  comparaisons.

### Coût de la recherche simple

En notant  $l_m$  la longueur du motif et  $l_t$  celle de la chaîne :

- La boucle **for** est parcourue au plus  $l_t - l_m + 1$  fois
- Pour chacun de ces parcours, la boucle **while** interne est parcourue au plus  $l_m$  fois

Au plus, l'algorithme effectue donc  $l_m(l_t - l_m + 1)$  comparaisons.

### Exemple

Combien de comparaisons seront nécessaires si on recherche le motif **bbbbbbbbbba** (neuf fois le caractère **b** suivi d'un **a**) dans une chaîne contenant un million de **b** ?

### Accélération de la recherche

Supposons qu'on recherche le motif extra dans la chaîne un excellent exemple et un exercice extraordinaire. La comparaison naïve ci-dessus commence par :

### Accélération de la recherche

Supposons qu'on recherche le motif extra dans la chaîne un excellent exemple et un exercice extraordinaire. La comparaison naïve ci-dessus commence par :

u	n		e	x	c	e	l	l	e	n	t
$\updownarrow$											
e	x	t	r	a							

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Accélération de la recherche

Supposons qu'on recherche le motif extra dans la chaîne un excellent exemple et un exercice extraordinaire. La comparaison naïve ci-dessus commence par :

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



e	x	t	r	a							
---	---	---	---	---	--	--	--	--	--	--	--

Deux idées vont permettre d'accélérer la recherche :

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Accélération de la recherche

Supposons qu'on recherche le motif extra dans la chaîne un excellent exemple et un exercice extraordinaire. La comparaison naïve ci-dessus commence par :

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



e	x	t	r	a							
---	---	---	---	---	--	--	--	--	--	--	--

Deux idées vont permettre d'accélérer la recherche :

- Commencer par la fin du motif.



### Accélération de la recherche

Supposons qu'on recherche le motif extra dans la chaîne un excellent exemple et un exercice extraordinaire. La comparaison naïve ci-dessus commence par :

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



e	x	t	r	a							
---	---	---	---	---	--	--	--	--	--	--	--

Deux idées vont permettre d'accélérer la recherche :

- Commencer par la fin du motif.
- Prétraiter le motif de façon à éviter des comparaisons inutiles.

### Accélération de la recherche

Dans l'exemple ci-dessus cela donne :

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Accélération de la recherche

Dans l'exemple ci-dessus cela donne :

u	n		e	x	c	e	l	l	e	n	t	
				↕								
e	x	t	r	a								

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Accélération de la recherche

Dans l'exemple ci-dessus cela donne :

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



e	x	t	r	a							
---	---	---	---	---	--	--	--	--	--	--	--

On peut directement décaler le motif de 3 emplacements car le dernier x du motif se trouve à 3 emplacements de la fin du motif.

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Accélération de la recherche

Dans l'exemple ci-dessus cela donne :

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



e	x	t	r	a							
---	---	---	---	---	--	--	--	--	--	--	--

On peut directement décaler le motif de 3 emplacements car le dernier x du motif se trouve à 3 emplacements de la fin du motif.

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



			e	x	t	r	a				
--	--	--	---	---	---	---	---	--	--	--	--

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Accélération de la recherche

Dans l'exemple ci-dessus cela donne :

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



e	x	t	r	a							
---	---	---	---	---	--	--	--	--	--	--	--

On peut directement décaler le motif de 3 emplacements car le dernier x du motif se trouve à 3 emplacements de la fin du motif.

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



			e	x	t	r	a				
--	--	--	---	---	---	---	---	--	--	--	--

Cette fois, le l ne se trouve pas dans le motif, on peut donc décaler de la longueur du motif. Et la recherche s'arrête en ayant effectué seulement deux comparaisons.

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Accélération de la recherche

Dans l'exemple ci-dessus cela donne :

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



e	x	t	r	a							
---	---	---	---	---	--	--	--	--	--	--	--

On peut directement décaler le motif de 3 emplacements car le dernier x du motif se trouve à 3 emplacements de la fin du motif.

u	n		e	x	c	e	l	l	e	n	t
---	---	--	---	---	---	---	---	---	---	---	---



			e	x	t	r	a				
--	--	--	---	---	---	---	---	--	--	--	--

Cette fois, le l ne se trouve pas dans le motif, on peut donc décaler de la longueur du motif. Et la recherche s'arrête en ayant effectué seulement deux comparaisons.

### Visualisation en ligne

Visualisation en ligne du fonctionnement de l'algorithme accéléré

### Algorithme de Boyer-Moore-Horspool

- La première phase consiste en un prétraitement du motif, afin de construire une **fonction de décalage** qui indique pour chaque caractère  $c$  :



### Algorithme de Boyer-Moore-Hoorspool

- La première phase consiste en un prétraitement du motif, afin de construire une **fonction de décalage** qui indique pour chaque caractère **c** :
  - Si **c** est dans le motif, Le nombre de caractères entre la *dernière occurrence* de **c** et la fin du motif (l'avant dernière si **c** est le dernier caractère.)

### Algorithme de Boyer-Moore-Horspool

- La première phase consiste en un prétraitement du motif, afin de construire une **fonction de décalage** qui indique pour chaque caractère **c** :
  - Si **c** est dans le motif, Le nombre de caractères entre la *dernière occurrence* de **c** et la fin du motif (l'avant dernière si **c** est le dernier caractère.)
  - Sinon la longueur du motif

### Algorithme de Boyer-Moore-Horspool

- La première phase consiste en un prétraitement du motif, afin de construire une **fonction de décalage** qui indique pour chaque caractère **c** :
  - Si **c** est dans le motif, Le nombre de caractères entre la *dernière occurrence* de **c** et la fin du motif (l'avant dernière si **c** est le dernier caractère.)
  - Sinon la longueur du motif
- Ensuite on effectue une recherche en partant de la fin du motif en cas de non correspondance, on décale de la valeur fournie par la fonction de décalage.

### Algorithme de Boyer-Moore-Horspool

- La première phase consiste en un prétraitement du motif, afin de construire une **fonction de décalage** qui indique pour chaque caractère **c** :
  - Si **c** est dans le motif, Le nombre de caractères entre la *dernière occurrence* de **c** et la fin du motif (l'avant dernière si **c** est le dernier caractère.)
  - Sinon la longueur du motif
- Ensuite on effectue une recherche en partant de la fin du motif en cas de non correspondance, on décale de la valeur fournie par la fonction de décalage.

### Exemple

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Algorithme de Boyer-Moore-Hoorspool

- La première phase consiste en un prétraitement du motif, afin de construire une **fonction de décalage** qui indique pour chaque caractère **c** :
  - Si **c** est dans le motif, Le nombre de caractères entre la *dernière occurrence* de **c** et la fin du motif (l'avant dernière si **c** est le dernier caractère.)
  - Sinon la longueur du motif
- Ensuite on effectue une recherche en partant de la fin du motif en cas de non correspondance, on décale de la valeur fournie par la fonction de décalage.

### Exemple

- 1 Construire la table de décalage du motif "toto"

### Algorithme de Boyer-Moore-Horspool

- La première phase consiste en un prétraitement du motif, afin de construire une **fonction de décalage** qui indique pour chaque caractère **c** :
  - Si **c** est dans le motif, Le nombre de caractères entre la *dernière occurrence* de **c** et la fin du motif (l'avant dernière si **c** est le dernier caractère.)
  - Sinon la longueur du motif
- Ensuite on effectue une recherche en partant de la fin du motif en cas de non correspondance, on décale de la valeur fournie par la fonction de décalage.

### Exemple

- 1 Construire la table de décalage du motif "toto"
- 2 Simuler le fonction de l'algorithme de Boyer-Moore-Horspool pour recherche ce motif dans le chaîne "zéro plus zéro = la tête à toto"

### Exercices

- 1 Ecrire en C, une fonction de signature `int *cree_decalage(char *motif)` qui renvoie la table de décalage d'un motif. On représentera un caractère par son code (donc un entier) et on suppose qu'on utilise la table ASCII standard qui contient 127 caractères.
- 2 Simuler la recherche de `abb` dans le texte `b...b` ( $t$  fois la lettre `b`) avec l'algorithme de Boyer-Mooore. Donner le nombre de comparaisons effectué et comparer avec la recherche naïve.

# C19 Algorithmes des textes

## 3. Algorithme de Boyer-Moore

### Implémentation en C

Fonction qui renvoie true si motif est présent dans texte et false sinon.

```
1  bool appartient_bmh(char *motif, char *texte){
2      int *dec = cree_decalage(motif);
3      int lt = strlen(texte);
4      int lm = strlen(motif);
5      int idx = 0;
6      int im;
7      while (idx < lt - lm + 1){
8          im = lm - 1;
9          while (im >= 0 && texte[idx + im] == motif[im]){
10              im = im - 1;}
11          if (im < 0){
12              return true;}
13          idx += dec[(int)texte[idx + lm - 1]];
14      return false;}
```



### Principe

L'idée de l'algorithme est d'utiliser une fonction de hachage  $h$  sur les chaînes de caractères, et de comparer les hash du motif  $h(m)$  avec le hash du texte commençant à l'indice  $i$  et de longueur  $l_m$  pour  $i$  de 0 à  $l_t - l_m$ . On effectue la comparaison caractère par caractère seulement dans le cas où les deux hash sont égaux.