



# C19 Algorithmes des textes

## 1. Problème de la recherche textuelle

### 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. Problème de la recherche textuelle

### 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. Problème de la recherche textuelle

### 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. Problème de la recherche textuelle

### 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. Problème de la recherche textuelle

### 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. Problème de la recherche textuelle

### 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 recherche 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 :



### 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-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.)

### 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 pour chaque indice  $i$  où une correspondance peut exister (de 0 à  $l_t - l_m$ ) les hash du motif  $h(m)$  avec le hash du texte commençant à l'indice  $i$  et de longueur  $l_m$  c'est à dire  $h((t_i, t_{i+1}, \dots, t_{i+l_m-1}))$ .

### Principe

- L'idée de l'algorithme est d'utiliser une fonction de hachage  $h$  sur les chaînes de caractères, et de comparer pour chaque indice  $i$  où une correspondance peut exister (de 0 à  $l_t - l_m$ ) les hash du motif  $h(m)$  avec le hash du texte commençant à l'indice  $i$  et de longueur  $l_m$  c'est à dire  $h((t_i, t_{i+1}, \dots, t_{i+l_m-1}))$ .
- On effectue la comparaison caractère par caractère seulement dans le cas où les deux hash sont égaux.

# C19 Algorithmes des textes

## 4. Algorithme de Rabin-Karp

### Principe

- L'idée de l'algorithme est d'utiliser une fonction de hachage  $h$  sur les chaînes de caractères, et de comparer pour chaque indice  $i$  où une correspondance peut exister (de 0 à  $l_t - l_m$ ) les hash du motif  $h(m)$  avec le hash du texte commençant à l'indice  $i$  et de longueur  $l_m$  c'est à dire  $h((t_i, t_{i+1}, \dots, t_{i+l_m-1}))$ .
- On effectue la comparaison caractère par caractère seulement dans le cas où les deux hash sont égaux.

### Exemple

On suppose qu'on recherche « de » dans le texte « bahcdef » et que la fonction de hachage fait simplement la somme des codes ASCII des caractères. Calculer le hash du motif et détailler les étapes de l'algorithme.

# C19 Algorithmes des textes

## 4. Algorithme de Rabin-Karp

### Principe (amélioré)

### Principe (amélioré)

### Principe (amélioré)

- Si le calcul du hash de la partie du texte  $(t_i, t_{i+1}, \dots, t_{i+l_m-1})$  s'effectue en  $O(m)$  alors la complexité est la même que celle de la recherche naïve.

### Principe (amélioré)

- Si le calcul du hash de la partie du texte  $(t_i, t_{i+1}, \dots, t_{i+l_m-1})$  s'effectue en  $O(m)$  alors la complexité est la même que celle de la recherche naïve.
- On tire parti du fait que deux hash consécutifs à calculer ne diffère que de deux caractères :

$(\textcolor{red}{t}_i, t_{i+1}, \dots, t_{i+l_m-1})$

$(t_{i+1}, \dots, t_{i+l_m-1}, \textcolor{red}{t}_i + \textcolor{red}{l}_m)$

et on utilise une fonction de hachage dite **déroulante** (*rolling hash*) pour calculer les hash de proche en proche en  $O(1)$ .



### Exemple

- 1 Ecrire en C, une fonction de hachage  $h$  qui effectue simplement la somme des codes des caractères.
- 2 Montrer qu'il s'agit d'une fonction de hachage déroulante et donner l'expression de  $h(t_{i+1}, \dots, t_{i+l_m-1}, t_i + l_m)$  en fonction de  $h(t_i, t_{i+1}, \dots, t_{i+l_m-1})$
- 3 Ecrire une l'implémentation de l'algorithme de Rabin-Karp qui utilise cette fonction de hachage.
  - lors que les deux hash sont égaux (celui du motif) et celui de la partie de texte, on pourra utiliser directement la fonction `strncmp` de `<string.h>` qui prend en argument deux chaines de caractères et un entier  $n$  et renvoie 0 lorsque les  $n$  premiers caractères des deux chaines sont égaux.

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Hypothèses

On considère dans toute la suite, qu'on travaille sur un fichier texte au format ASCII et on rappelle que ce format permet de représenter 128 caractères (de code 0 à 127) tous codés sur 8 bits. Par conséquent la taille d'un tel fichier (en bits) est simplement 8 fois son nombre de caractères. On cherche à compresser la taille d'un tel fichier *sans perdre d'informations*.

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Hypothèses

On considère dans toute la suite, qu'on travaille sur un fichier texte au format ASCII et on rappelle que ce format permet de représenter 128 caractères (de code 0 à 127) tous codés sur 8 bits. Par conséquent la taille d'un tel fichier (en bits) est simplement 8 fois son nombre de caractères. On cherche à compresser la taille d'un tel fichier *sans perdre d'informations*.

### Principe

Le principe de base de l'algorithme d'Huffman est de représenter par un code plus court les caractères les plus fréquents :

- on commence par calculer le nombre d'occurrences de chaque caractère
- on construit un arbre dont les feuilles sont les caractères et où la profondeur d'un caractère est fonction directe de sa fréquence (plus un caractère est haut dans l'arbre et plus il apparaît souvent)
- on génère un code **prefixe** unique pour chaque caractère.

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Illustration sur un exemple

On souhaite compresser le texte « les petits tests ».

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Illustration sur un exemple

On souhaite compresser le texte « les petits tests ».

1

Caractère	l	e	i	p	s	t
Occurrences	2	3	1	1	4	4

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Illustration sur un exemple

On souhaite compresser le texte « les petits tests ».

1

Caractère	l	e	i	p	s	t
Occurrences	2	3	1	1	4	4

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Illustration sur un exemple

On souhaite compresser le texte « les petits tests ».

1	Caractère	␣	e	i	l	p	s	t
	Occurrences	2	3	1	1	1	4	4

2 Au départ chaque caractère représente un arbre comportant un noeud unique :



# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Illustration sur un exemple

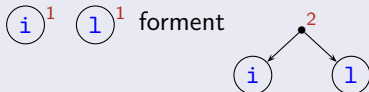
On souhaite compresser le texte « les petits tests ».

1	Caractère	␣	e	i	l	p	s	t
	Occurrences	2	3	1	1	1	4	4

- 2 Au départ chaque caractère représente un arbre comportant un noeud unique :



- 3 On extrait les deux arbres portant les plus petits nombres d'occurrences et on les assemblent en un nouvel arbre ayant la somme de leur nombre d'occurrence :





# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Illustration sur un exemple

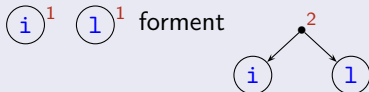
On souhaite compresser le texte « les petits tests ».

1	Caractère	␣	e	i	l	p	s	t
	Occurrences	2	3	1	1	1	4	4

- 2 Au départ chaque caractère représente un arbre comportant un noeud unique :



- 3 On extrait les deux arbres portant les plus petits nombres d'occurrences et on les assemblent en un nouvel arbre ayant la somme de leur nombre d'occurrence :



- 4 Ce nouvel arbre est inséré dans la structure de données et on réitère le processus jusqu'à ce qu'il reste un seul arbre

# C19 Algorithmes des textes

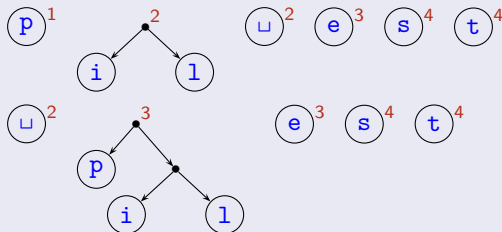
## 5. Compression avec l'algorithme de Huffman

Encore quelques étapes pour illustrer ...

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

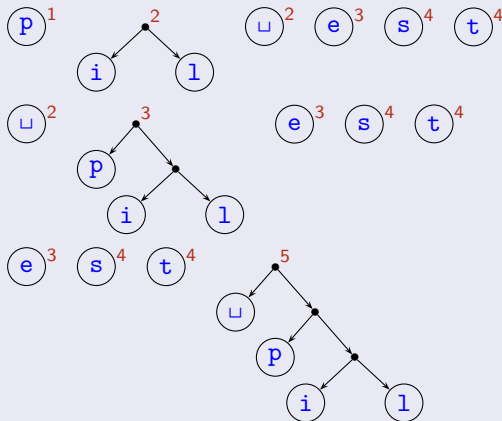
Encore quelques étapes pour illustrer ...



# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

Encore quelques étapes pour illustrer ...



# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

Et pour finir ...

# C19 Algorithmes des textes

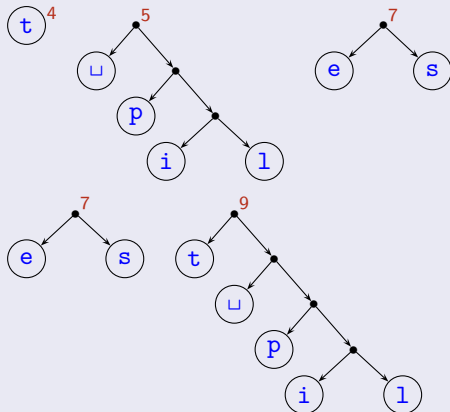
## 5. Compression avec l'algorithme de Huffman

Et pour finir ...

# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

Et pour finir ...

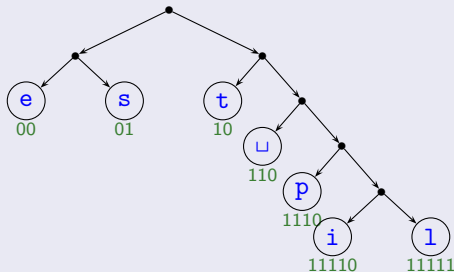


# C19 Algorithmes des textes

## 5. Compression avec l'algorithme de Huffman

### Attribution des codes

Sur l'arbre final on attribut un code à chaque caractère en fonction de sa position dans l'arbre (on ajoute 0 en allant à gauche et 1 en allant à droite)



On obtient un **code préfixe**, c'est à dire que le code d'un caractère n'est jamais le début du code d'un autre (ce qui évite toute ambiguïté lors de la décompression). Le texte initial contenait 4 caractères **s** pour un total de 32 bits, maintenant les **s** sont codés sur 2 bits et donc occupent simplement 8 bits



### Exercices

- 1 Calculer le taux de compression de l'algorithme sur l'exemple précédent.

### Exercices

- 1 Calculer le taux de compression de l'algorithme sur l'exemple précédent.
- 2 Quelle structure de données est la plus adaptée pour ranger les arbres au fur et à mesure de leur construction, pourquoi ?

### Exercices

- 1 Calculer le taux de compression de l'algorithme sur l'exemple précédent.
- 2 Quelle structure de données est la plus adaptée pour ranger les arbres au fur et à mesure de leur construction, pourquoi ?
- 3 Faire fonctionner l'algorithme à la main pour compresser le texte « trop top »

### Introduction

- Le nom de cet algorithme provient de celui de ses trois inventeurs : Lemp, Ziv et Welch.

### Introduction

- Le nom de cet algorithme provient de celui de ses trois inventeurs : Lemp, Ziv et Welch.
- L'algorithme a l'avantage de pouvoir compresser un texte au fur et à mesure de sa lecture et est donc adapté à la compression d'un flux de données

### Introduction

- Le nom de cet algorithme provient de celui de ses trois inventeurs : Lemp, Ziv et Welch.
- L'algorithme a l'avantage de pouvoir compresser un texte au fur et à mesure de sa lecture et est donc adapté à la compression d'un flux de données
- Le principe est d'attribuer un code aux sous chaînes rencontrées dans le texte de façon à disposer d'un code compact si elles se présentent à nouveau.

# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte `saisissais` qui ne contient que les 3 lettres `a`, `i` et `s`. On commence par attribuer à chaque lettre un code : `a`  $\rightarrow$  0, `i`  $\rightarrow$  1 et `s`  $\rightarrow$  2.

# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte **saisissais** qui ne contient que les 3 lettres **a**, **i** et **s**. On commence par attribuer à chaque lettre un code : **a**  $\rightarrow$  0, **i**  $\rightarrow$  1 et **s**  $\rightarrow$  2. Puis on parcourt le texte en émettant le code du **plus long préfixe rencontré**. On rajoute ce préfixe augmenté du caractère suivant dans la liste des codes (de façon à disposer d'un code lorsqu'on le rencontrera à nouveau.)

Position	Emis	Nouveau préfixe
----------	------	-----------------



# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte **saisissais** qui ne contient que les 3 lettres **a**, **i** et **s**. On commence par attribuer à chaque lettre un code : **a**  $\rightarrow$  0, **i**  $\rightarrow$  1 et **s**  $\rightarrow$  2. Puis on parcourt le texte en émettant le code du **plus long préfixe rencontré**. On rajoute ce préfixe augmenté du caractère suivant dans la liste des codes (de façon à disposer d'un code lorsqu'on le rencontrera à nouveau.)

Position	Emis	Nouveau préfixe
<u>s</u> aisissais	2	<b>sa</b> $\rightarrow$ 3

# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte **saisissais** qui ne contient que les 3 lettres **a**, **i** et **s**. On commence par attribuer à chaque lettre un code : **a**  $\rightarrow$  0, **i**  $\rightarrow$  1 et **s**  $\rightarrow$  2. Puis on parcourt le texte en émettant le code du **plus long préfixe rencontré**. On rajoute ce préfixe augmenté du caractère suivant dans la liste des codes (de façon à disposer d'un code lorsqu'on le rencontrera à nouveau.)

Position	Emis	Nouveau préfixe
<u>s</u> aisissais	2	<b>sa</b> $\rightarrow$ 3
s <u>a</u> issais	0	<b>ai</b> $\rightarrow$ 4

# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte `saisissais` qui ne contient que les 3 lettres `a`, `i` et `s`. On commence par attribuer à chaque lettre un code : `a`  $\rightarrow$  0, `i`  $\rightarrow$  1 et `s`  $\rightarrow$  2. Puis on parcourt le texte en émettant le code du **plus long préfixe rencontré**. On rajoute ce préfixe augmenté du caractère suivant dans la liste des codes (de façon à disposer d'un code lorsqu'on le rencontrera à nouveau.)

Position	Emis	Nouveau préfixe
<u>s</u> aisissais	2	<code>sa</code> $\rightarrow$ 3
s <u>a</u> issais	0	<code>ai</code> $\rightarrow$ 4
sai <u>i</u> ssais	1	<code>is</code> $\rightarrow$ 5

# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte **saisissais** qui ne contient que les 3 lettres **a**, **i** et **s**. On commence par attribuer à chaque lettre un code : **a**  $\rightarrow$  0, **i**  $\rightarrow$  1 et **s**  $\rightarrow$  2. Puis on parcourt le texte en émettant le code du **plus long préfixe rencontré**. On rajoute ce préfixe augmenté du caractère suivant dans la liste des codes (de façon à disposer d'un code lorsqu'on le rencontrera à nouveau.)

Position	Emis	Nouveau préfixe
<u>s</u> aissais	2	<b>sa</b> $\rightarrow$ 3
s <u>a</u> issais	0	<b>ai</b> $\rightarrow$ 4
sai <u>i</u> ssais	1	<b>is</b> $\rightarrow$ 5
sais <u>s</u> ais	2	<b>si</b> $\rightarrow$ 6

# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte **saisissais** qui ne contient que les 3 lettres **a**, **i** et **s**. On commence par attribuer à chaque lettre un code : **a**  $\rightarrow$  0, **i**  $\rightarrow$  1 et **s**  $\rightarrow$  2. Puis on parcourt le texte en émettant le code du **plus long préfixe rencontré**. On rajoute ce préfixe augmenté du caractère suivant dans la liste des codes (de façon à disposer d'un code lorsqu'on le rencontrera à nouveau.)

Position	Emis	Nouveau préfixe
<u>s</u> aissais	2	<b>sa</b> $\rightarrow$ 3
s <u>a</u> issais	0	<b>ai</b> $\rightarrow$ 4
sai <u>i</u> ssais	1	<b>is</b> $\rightarrow$ 5
sais <u>s</u> issais	2	<b>si</b> $\rightarrow$ 6
saisi <u>s</u> sais	5	<b>iss</b> $\rightarrow$ 7

# C19 Algorithmes des textes

## 6. Compression avec l'algorithme LZW

### Illustration sur un exemple

On veut compresser le texte **saisissais** qui ne contient que les 3 lettres **a**, **i** et **s**. On commence par attribuer à chaque lettre un code : **a**  $\rightarrow$  0, **i**  $\rightarrow$  1 et **s**  $\rightarrow$  2. Puis on parcourt le texte en émettant le code du **plus long préfixe rencontré**. On rajoute ce préfixe augmenté du caractère suivant dans la liste des codes (de façon à disposer d'un code lorsqu'on le rencontrera à nouveau.)

Position	Emis	Nouveau préfixe
<u>s</u> aissais	2	<b>sa</b> $\rightarrow$ 3
s <u>a</u> issais	0	<b>ai</b> $\rightarrow$ 4
sai <u>i</u> ssais	1	<b>is</b> $\rightarrow$ 5
sai <u>s</u> issais	2	<b>si</b> $\rightarrow$ 6
sais <u>i</u> ssais	5	<b>iss</b> $\rightarrow$ 7
saisis <u>s</u> aiss	3	<b>sai</b> $\rightarrow$ 3
saisissai <u>s</u>	5	