Recherche de Motifs

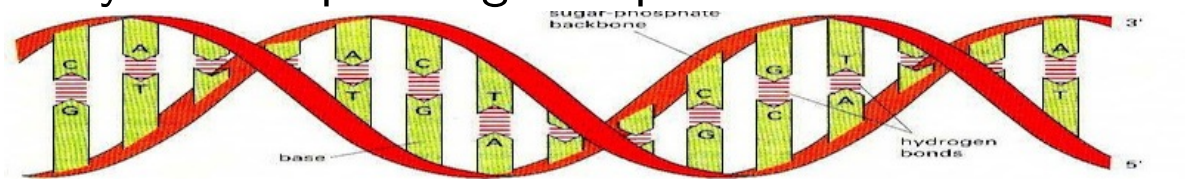
Samuel Blanquart, d'après les supports de Hélène Touzet

AeA – M1 informatique

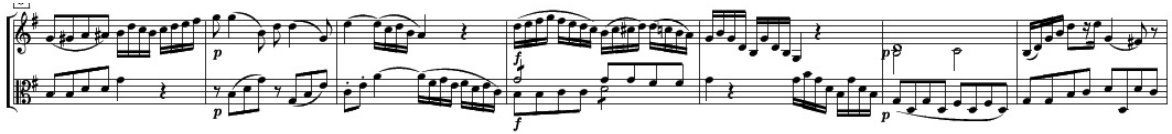
Longtemps, je me suis couché de bonne heure. Parfois, à peine ma bougie éteinte, mes yeux se fermaient si vite que je n'avais pas le temps de me dire : Je m'endors. Et, une demi-heure après, la pensée qu'il était temps de chercher le sommeil m'éveillait ; je voulais poser le volume que je croyais avoir dans les mains et souffler ma lumière ; je n'avais pas cessé en dormant de faire des réflexions sur ce que je venais de lire, mais ces réflexions avaient pris un tour particulier ; il me semblait que j'étais moi-même ce dont parlait l'ouvrage : une église, un quatuor, la rivalité de François Ier et de Charles-Quint. Cette croyance survivait pendant quelques secondes à mon réveil ; elle ne choquait pas ma raison, mais pesait comme des écailles sur mes yeux et les empêchait de se rendre compte que le bougeoir n'était plus allumé.

Recherche de motifs

- un des plus vieux problèmes de l'informatique
- nombreuses applications
  - éditeurs de texte
    `grep` en Unix – `CTRL s` sous Emacs – `CTRL f` sous Word
  - moteurs de recherche
  - analyse de séquence génétiques



  - recherche de motifs musicaux



Voir projet

Quelques repères historiques

- Extrait de wikipedia, article Chronologie de l'informatique
  - L'algorithme de Dijkstra par Edsger Dijkstra 1959
  - L'algorithme de Floyd par Robert Floyd 1959
  - L'algorithme Quicksort par Tony Hoare 1961
  - Invention de l'algorithme de Knuth-Morris-Pratt 1975 ⋆
  - Invention de l'algorithme de Boyer-Moore 1977 ⋆
- Prix Turing
  - Donald Knuth 1974
  - Michael Rabin, 1976
  - Richard Karp, 1985

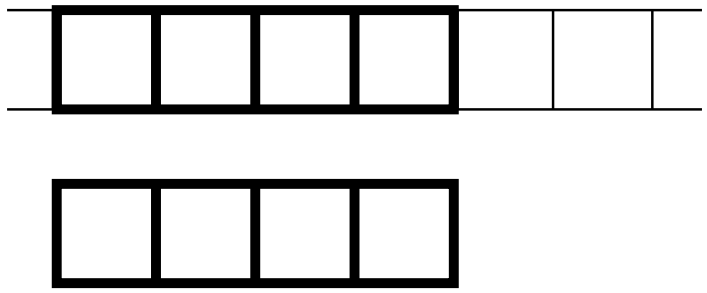
Définitions

- Alphabet : Σ, ensemble fini de lettres (caractères, symboles)
- Mot : suite finie d'éléments de Σ
- Concaténation : la concaténation de deux mots u et v est le mot composé des lettres de u, suivi des lettres de v. Elle est notée $u \circ v$.
- Facteur : un mot u est un facteur du mot v, si, et seulement s'il existe deux mots w et z tels que $v = w \circ u \circ z$
- Occurrence : Si u est un facteur de v, on dit que u apparait dans v, ou que v contient une occurrence de u
- Préfixe : un mot u est un préfixe du mot v, si, et seulement s'il existe un mot t tel que $v = u \circ t$
- Suffixe : un mot u est un suffixe du mot v, si, et seulement s'il existe un mot t tel que $v = t \circ u$
- Bord : un mot u est un bord du mot v, si et seulement si u est à la fois un préfixe propre et un suffixe propre de v.

Recherche de motifs

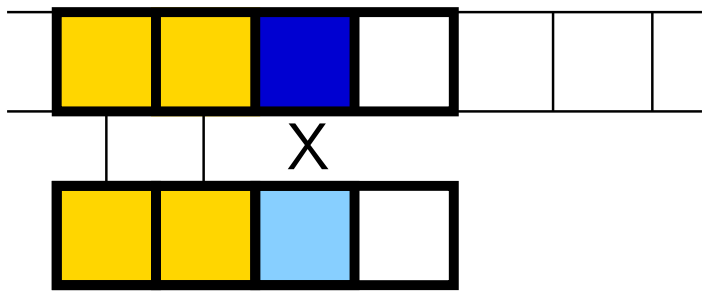
- Un texte T, mot de longueur n : $T(0..n-1)$
- Un motif M, mot de longueur m $(m < n)$: $M(0..m-1)$
- Trouver toutes les occurrences de M dans T

Algorithme naif (par force brute)



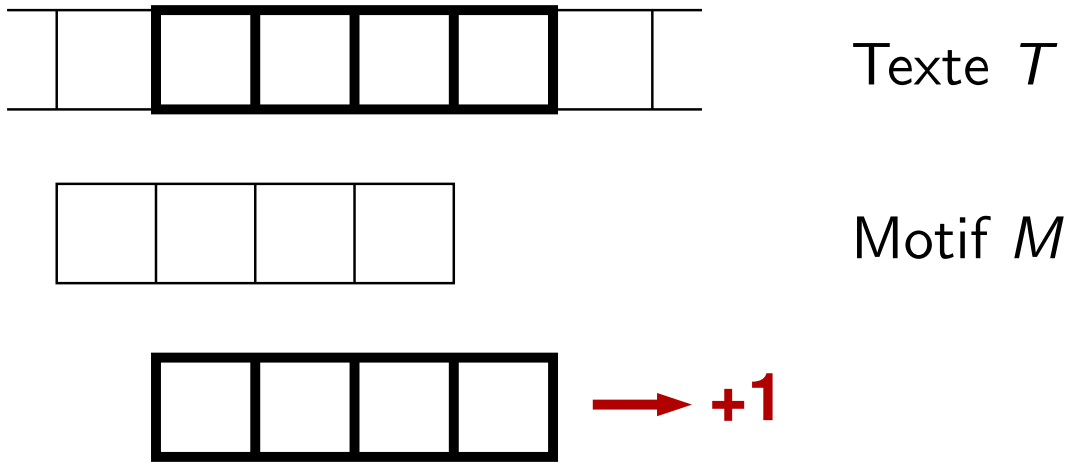
- sens de lecture du texte : de gauche à droite

Algorithme naif (par force brute)



- sens de lecture du texte : de gauche à droite
- tentative : comparaison du motif contre le texte, caractère par caractère, de gauche à droite

Algorithme naif (par force brute)



- sens de lecture du texte : de gauche à droite
- tentative : comparaison du motif contre le texte, caractère par caractère, de gauche à droite
- décalage du motif : $+1$ position, vers la droite

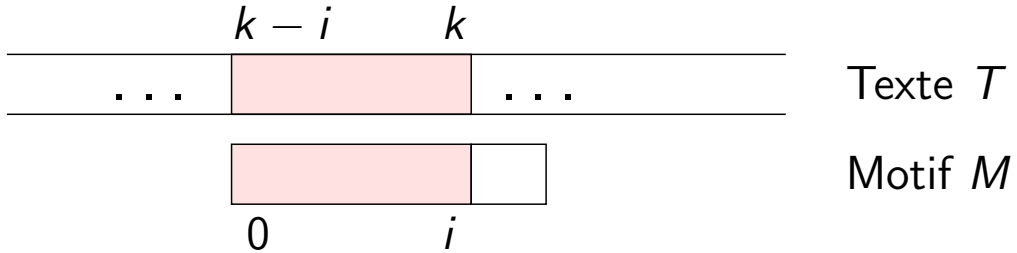
Amélioration de l'approche par force brute

- sens de lecture du texte : de gauche à droite
- tentative
  - sens de lecture du motif : droite, gauche, mélangé
  - comparaison caractère par caractère /comparaison globale
- décalage
  - de plus de 1 position
  - sans manquer d'occurrences

Algorithme Shift-Or (Baeza-Yates, Gonnet, 1992)

- décalage : $+1$ position, à droite
- tentative : comparaison grâce à un tableau de bits
- $\mathcal{B}$: matrice $m \times n$ sur 0 et 1

$$\mathcal{B}(i,k) = 1 \Leftrightarrow M(0..i) = T(k-i..k)$$



- les occurrences du motif M se trouvent aux positions p telles que $\mathcal{B}(m-1, p+m-1) = 1$

Shift-Or : Calcul de la matrice $\mathcal{B}$

- exemple : le motif *tactaga*
- définition de quatre vecteurs (un par lettre de l'alphabet)

$U(a)$	$U(c)$	$U(g)$	$U(t)$
0	0	0	1
1	0	0	0
0	1	0	0
0	0	0	1
1	0	0	0
0	0	1	0
1	0	0	0

- $U(x)(i) = 1 \leftrightarrow M(i) = x$

Première colonne de $\mathcal{B}$

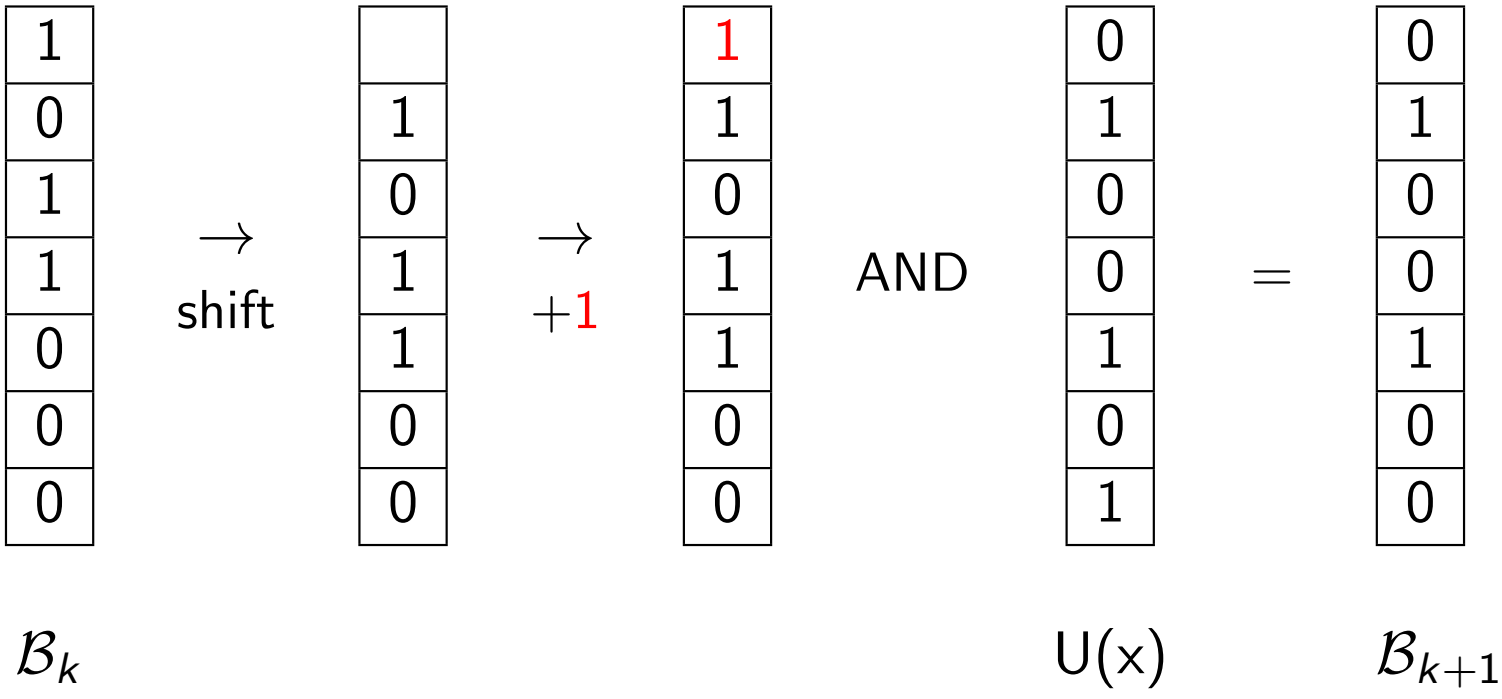
0
0
0
0
0
0
0

si $T(0) \neq M(0)$

1
0
0
0
0
0
0

si $T(0) = M(0)$

Colonnes suivantes de $\mathcal{B}$: la colonne $k+1$ à partir de la colonne k



où $x = T(k+1)$.
AND est l'opérateur qui effectue le **Et logique** bit par bit.

Shift-Or : Exemple

- Mot *ctactatatatc*
- Motif *tata*

$U(a)$	$U(c)$	$U(g)$	$U(t)$
0	0	0	1
1	0	0	0
0	0	0	1
1	0	0	0

- La matrice $\mathcal{B}$

	c	t	a	c	t	a	t	a	t	a	t	c
t	0	1	0	0	1	0	1	0	1	0	1	0
a	0	0	1	0	0	1	0	1	0	1	0	0
t	0	0	0	0	0	0	1	0	1	0	1	0
a	0	0	0	0	0	0	0	1	0	1	0	0

- Une occurence est trouvée si on lit 1 sur la dernière ligne.

Shift-or : conclusion

- Points forts
  - pré-traitement quasi inexistant
  - facile à implémenter
  - fonctionne bien si le motif est de taille inférieure à un mot machine : les opérations se font alors en temps constants. On obtient alors une complexité linéaire.
  - s'adapte facilement au cas de motifs approchés
- Points faibles
  - inadapté quand le motif est long

Algorithme de Karp-Rabin (1987)

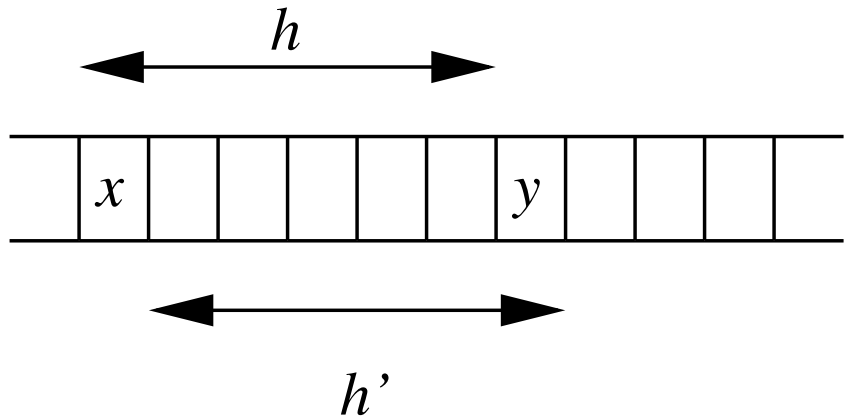
- décalage : $+1$ position, à droite
- tentative : comparaison grâce à une fonction de hachage h
- arithmétique modulo et décalage
- un mot est codé par un entier en base d

$$h(u) = (u_0 d^{m-1} + u_1 d^{m-2} \cdots + u_{m-1}) \text{ modulo } q$$

- d : taille de l'alphabet
- q : plus grand entier. Le calcul du modulo se fait implicitement.

Karp-Rabin : déroulement de l'algorithme

1. calcul de $h(M)$
2. calcul de $h(T(0..m-1)$
3. déplacement d'une fenêtre de longueur m le long du texte, de 1 en 1, avec réactualisation de la valeur de h



$$h' = (d(h - xd^{m-1}) + y) \text{ modulo } q \text{ (temps constant)}$$

4. quand on trouve une fenêtre f telle que $h(f) = h(u)$, on vérifie en comparant f et u caractère par caractère

Karp-Rabin : Exemple

- Alphabet : $\Sigma = \{a, c, g, t\}$,
- Motif $tata \leftrightarrow 4, 1, 4, 1$
- Hachage : $h(tata) = 4 \times 4^3 + 1 \times 4^2 + 4 \times 4^1 + 1 = 289$,
- Le texte et son hachage :

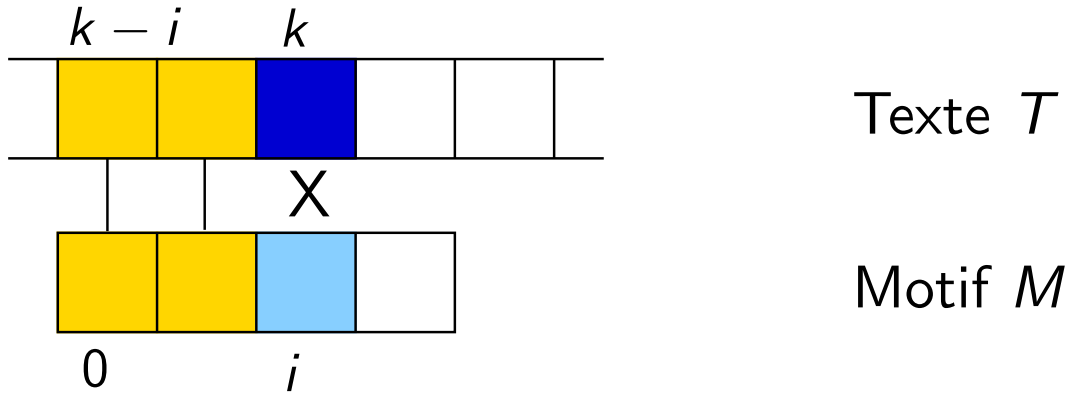
c	t	a	c	t	a	t	a	t	a	t	c
198	284	113	200	289	136	289	136	290			

Karp-Rabin : conclusion

- Points forts
  - Facile à implémenter
  - Plus efficace en pratique que l'algorithme naif
- Points faibles
  - Complexité en $O(mn)$ dans le pire des cas
  - Pas le plus efficace en pratique

Algorithme de Morris et Pratt (1970)

- décalage intelligent : $+1$ position ou plus, vers la droite
- tentative : comparaison caractère par caractère, de gauche à droite

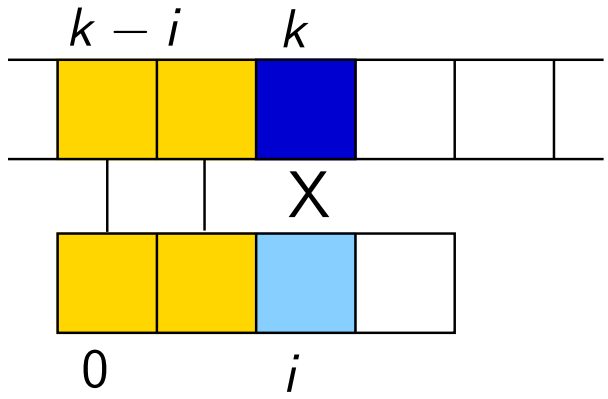


- information exploitée

$$T(k-i..k-1) \quad = \quad M(0..i-1)$$

- quand une erreur intervient en position i, on décale le motif le long du texte directement a la prochaine position ou peut démarrer M

Algorithme de Knuth, Morris et Pratt (1975)

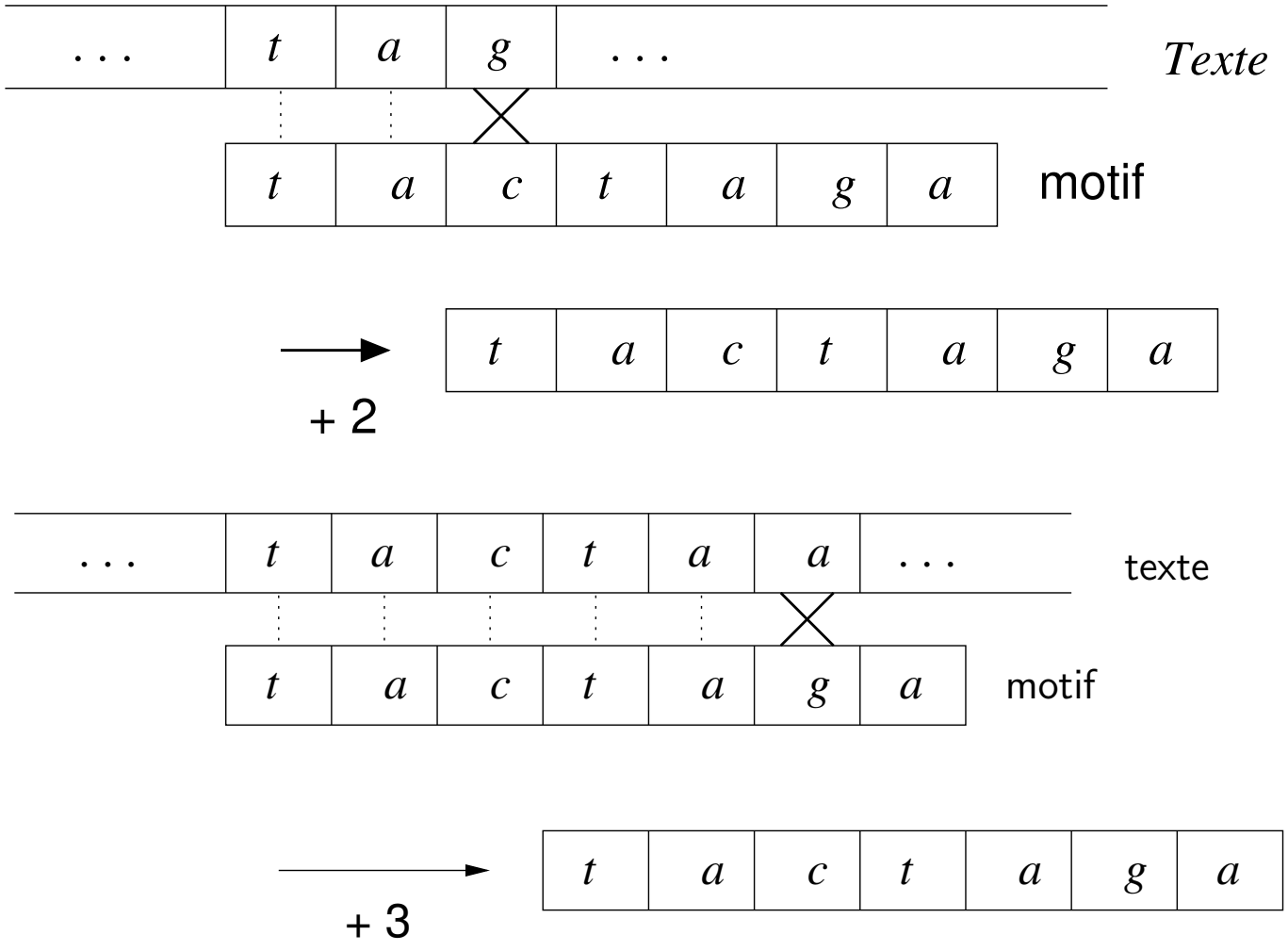


Texte T

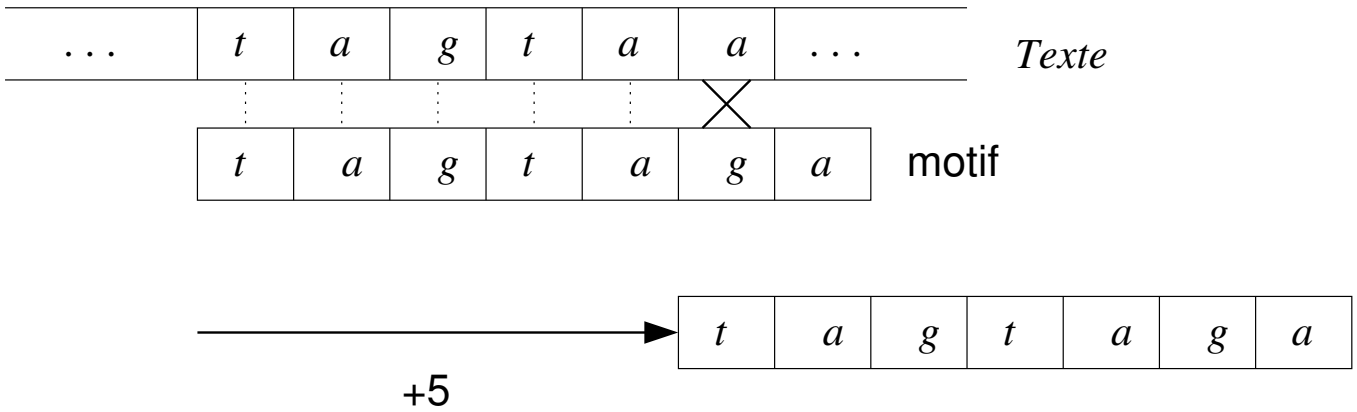
Motif M

- Deux informations sont exploitées
  1. $T(k-i..k-1) = M(0..i-1)$ (Morris-Pratt)
  2. $T(k) \neq M(i)$

- Exemple 1 : le motif *tactaga*



- Exemple 2 : le motif *tagtaga*



KMP : Mise en œuvre

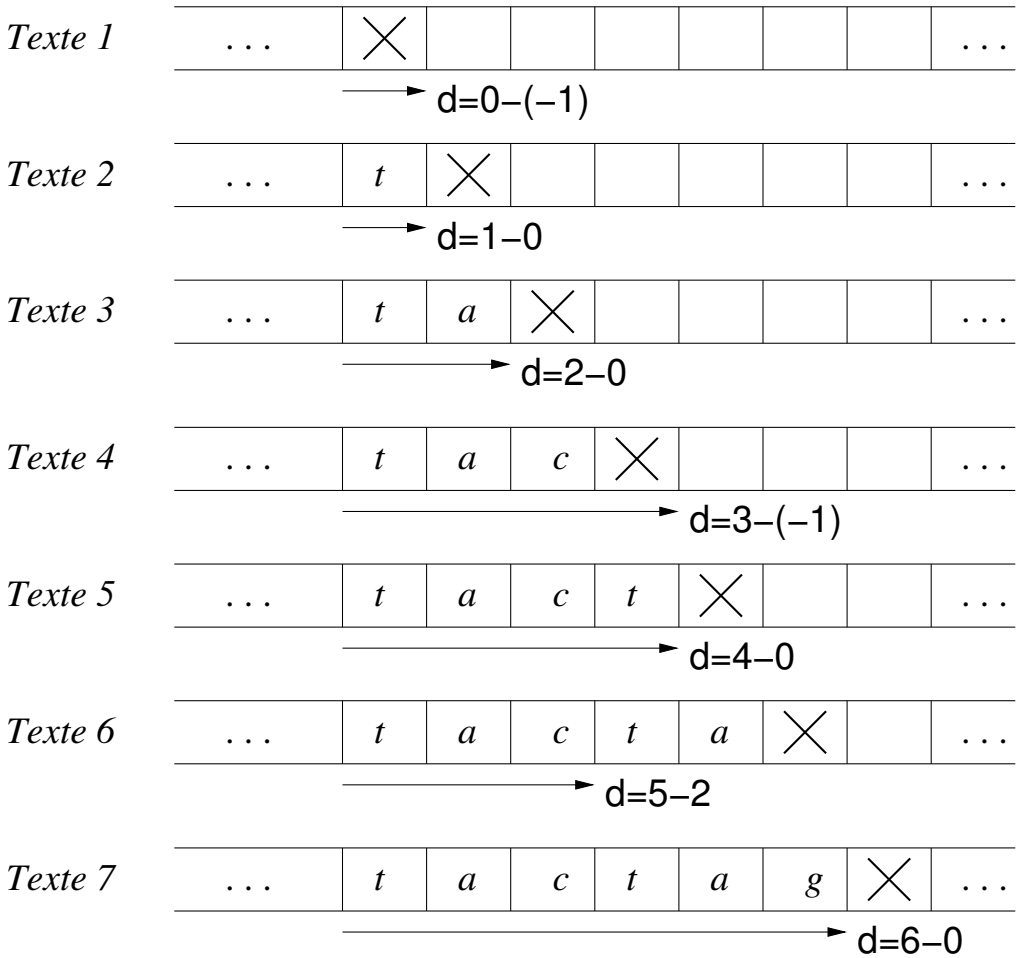
- Phase 1 **Prétraitement du motif**, le tableau Next
  - **i** : position dans le motif
  - **Next(i)** : longueur du plus long mot u possible tel que u est un bord de $M(0..i-1)$ et $uM(i)$ n'est pas un préfixe de M. **Next(i)=-1** si un tel u n'existe pas.

	t	a	c	t	a	g	a	
i	0	1	2	3	4	5	6	7
Next(i)	-1	0	0	-1	0	2	0	0

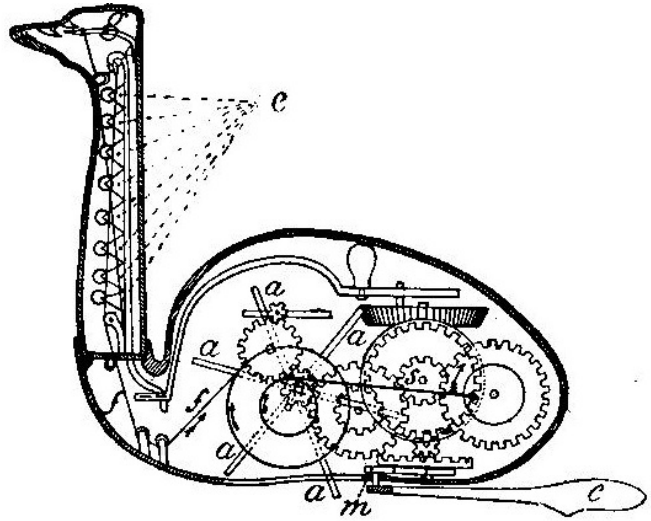
	t	a	g	t	a	g	
i	0	1	2	3	4	5	6
Next(i)	-1	0	0	-1	0	0	3

- Phase 2 : **Recherche du motif dans le texte.** Si erreur en position i du motif, décaler le motif sur le texte de $i - \text{Next}(i)$ vers la droite, reprendre la comparaison à la position courante dans le texte.

motif	t	a	c	t	a	g	a
i	0	1	2	3	4	5	6
Next(i)	-1	0	0	-1	0	2	0

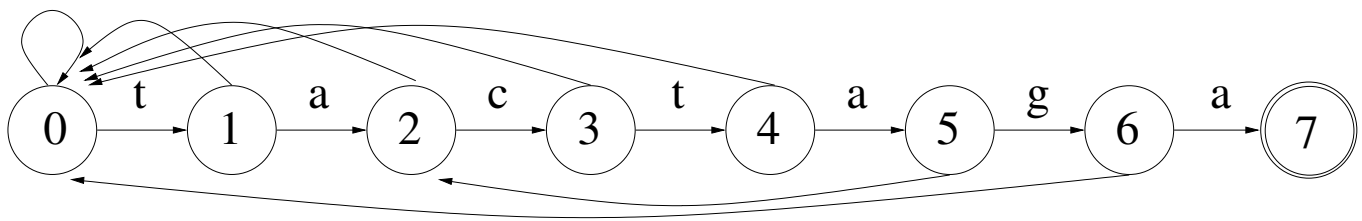


Il y a un automate caché derrière KMP



Chaque état possède deux transitions :

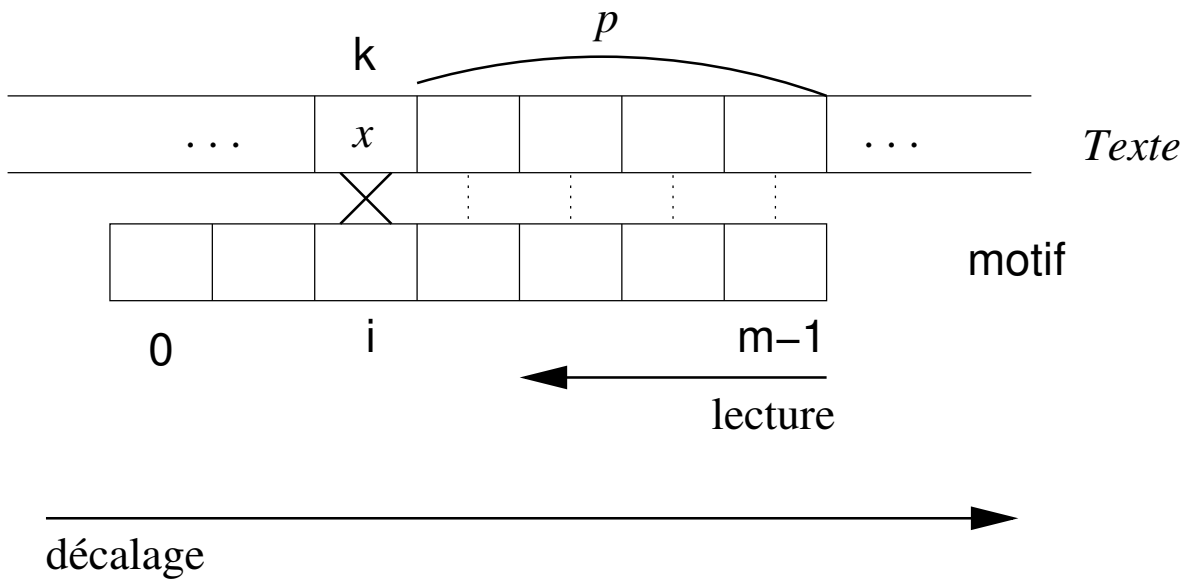
- Une transition de succès qui permet d'avancer d'un pas vers la droite du texte et du motif,
- Une transition d'échec qui renvoie à l'endroit où l'analyse doit être poursuivie dans le motif, suite à son décalage adéquate dans le texte.



KMP : conclusion

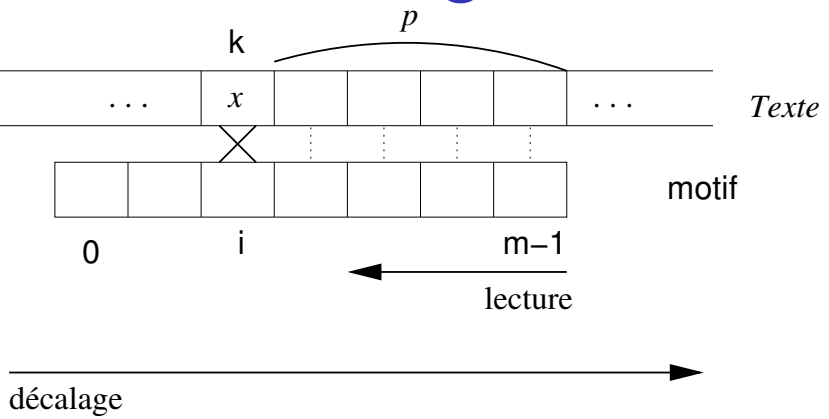
- Points forts
  - Rôle fondateur
  - Complexité linéaire
  - Pré-traitement du motif facile à implémenter (non détaillé)
- Points faibles
  - Moins efficace en moyenne que l'algorithme de Boyer-Moore

Algorithme de Boyer-Moore (1977)



- Tentative : le motif est lu de la droite vers la gauche, caractère par caractère
- Décalage : le plus de positions possibles, en prétraitant le motif

Boyer-Moore : Règle du bon suffixe



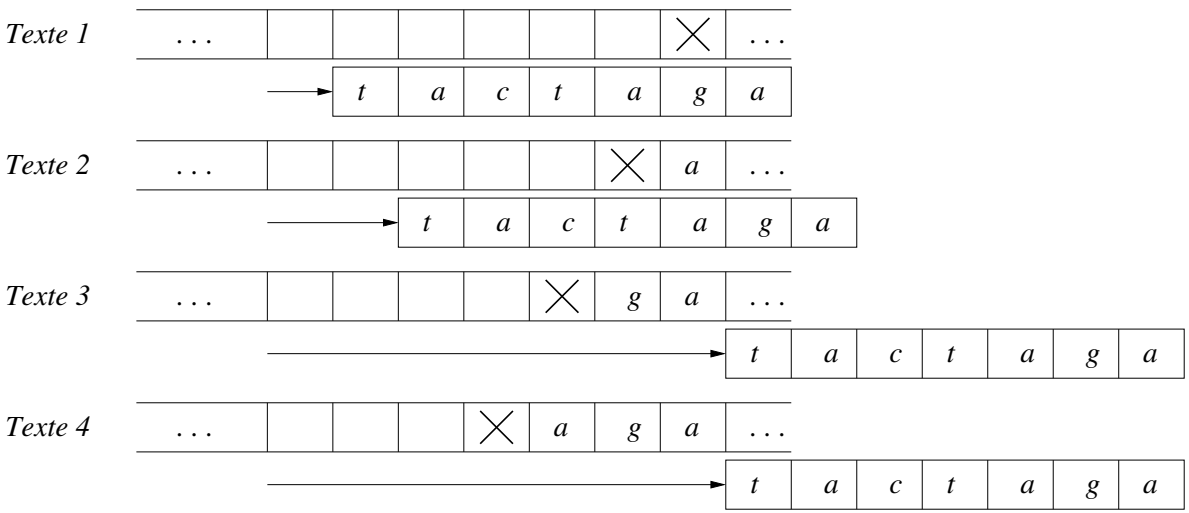
- Information exploitée : à la position $k+1$, le texte contient le suffixe $p = M(i+1..m-1)$ du motif, et $T(k) \neq M(i)$
- Décalage : la prochaine occurrence (vers la gauche) de p dans M, telle que le caractère précédent est différent de $M(i)$, est alignée avec la position $k+1$ de T :
  - $GS(i)$ (good suffix) : plus petit entier non nul ℓ tel que le décalage de ℓ positions satisfasse
  - $M(i+1-\ell..m-1-\ell) = M(i+1..m-1)$
  - $M(i-\ell) \neq M(i)$
- Ou, si ℓ n'existe pas : $\ell = m-k$, où k est le plus long bord du motif.

- Application de la règle du bon suffixe : $GS(i) = \ell$ tel que :
  - $M(i+1-\ell..m-1-\ell) = M(i+1..m-1)$
  - $M(i-\ell) \neq M(i)$
  - Exemple : $i == 7$, $\ell == 6$

Texte	S	T	A	B	S	T	U	B	A	B	V	Q	X	R
i	0	1	2	3	4	5	6	7	8	9				
Motif	Q	C	A	B	D	A	B	D	A	B				
$+=6$							Q	C	A	B	D	A	B	D

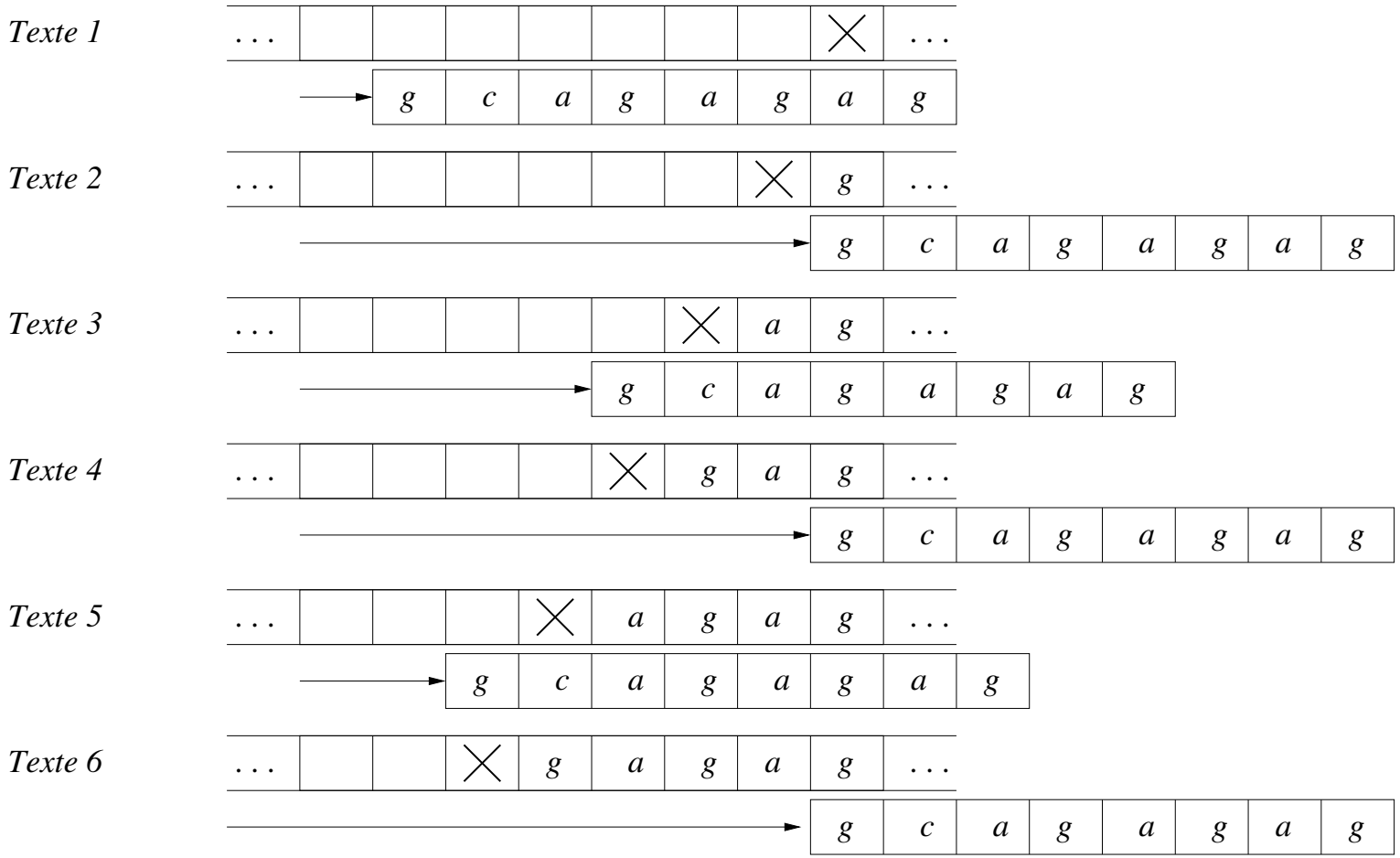
- Application de la règle du plus long bord : $\ell = m - k$ tel que $M(m-1-k..m-1) = M(0..k)$

motif	t	a	c	t	a	g	a
i	0	1	2	3	4	5	6
GS(i)	7	7	7	7	7	2	1

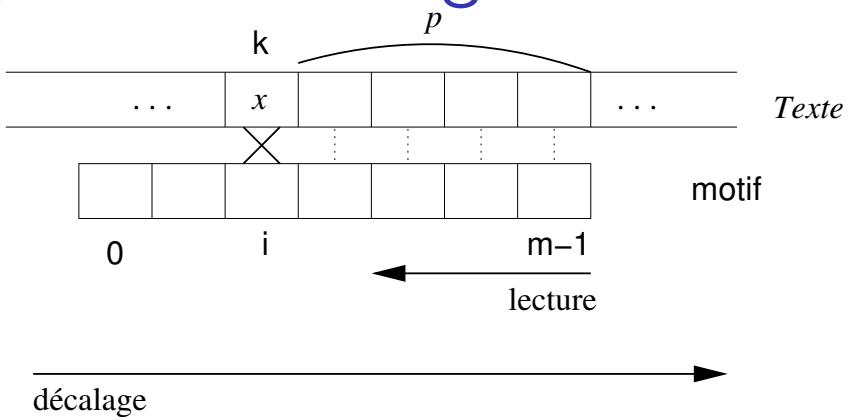


Boyer-Moore : dernier exemple

motif	*g*	*c*	*a*	*g*	*a*	*g*	*a*	*g*
i	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
GS(i)	*7*	*7*	*7*	*2*	*7*	*4*	*7*	*1*



Boyer-Moore : règle du mauvais caractère



- information exploitée : le texte contient le caractère $x = T(k)$ en position k
- décalage : la position k du texte est alignée avec un caractère x du motif
- **BC**(x) (bad character) : longueur du plus long suffixe de M qui ne contient pas x, sauf éventuellement en dernière position
- Exemple : le motif *gcagagag*

a	c	g	t
1	6	2	8

- Bon suffixe + mauvais caractère : Quand un mismatch intervient entre la position i du motif et la position k du texte, le décalage se fait de $\max\{\mathbf{GS(i)}, \mathbf{BC(T(k))} - (\mathbf{m} - \mathbf{i})\}$ positions.

Boyer-Moore : conclusion

- Points forts
  - Complexité : $O(mn)$ dans le pire des cas, mais sous-linéaire en moyenne
    Sous-linéaire : un caractère est lu moins d'une fois.
  - Terriblement efficace en pratique si l'alphabet est grand (26 caractères) , ou si le motif est long

- Points faibles
  - Pré-traitement en temps linéaire, mais délicat à implémenter (non détaillé)
  - Moins convaincant pour de courts motifs sur un petit alphabet (l'ADN par exemple)