Structures de texte

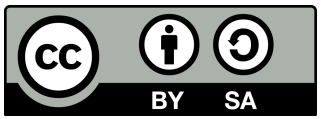
Recherche de motif

Jean-Marc.Vincent@univ-grenoble-alpes.fr[1]

[1] Laboratoire LIG
Équipe-Projet INRIA POLARIS
Université Grenoble-Alpes

DIU EIL- Université de Polynésie Française
2020

I. LE PROBLÈME : Recherche de motifs
II. ALGORITHME : force brute
III. RAFFINEMENT : utiliser l'information
IV. ALGORITHME DE KNUTH-MORRIS-PRATT
V. ALGORITHME DE BOYER-MOORE-HORSPOL
VI. SYNTHÈSE



UGA Université Grenoble Alpes

LE PROBLÈME

On recherche un motif dans un texte, mais en général le motif et le texte peuvent être :

- des suites de caractères (ASCII ou pas)
- des suites d'octets dans un fichier quelconque
- des séquences ADN (suites de bases ATGC)
- ...

LE PROBLÈME

On recherche un motif dans un texte, mais en général le motif et le texte peuvent être :

- des suites de caractères (ASCII ou pas)
- des suites d'octets dans un fichier quelconque
- des séquences ADN (suites de bases ATGC)
- ...

Formalisation

Plus généralement, on considère :

- un alphabet A fini
- deux suites ordonnées d'éléments de A (le "texte" et le "motif")

et on cherche à répondre à différentes questions plus ou moins équivalentes :

- le motif apparaît-il dans le texte ?
- si oui, déterminer la première position où il apparaît ;
- déterminer toutes les positions où ce motif apparaît.

UGA Université Grenoble Alpes

STRUCTURES DE DONNÉES ET NOTATIONS

On considère que le texte et le motif sont stockés dans des tableaux, ce qui permet de se déplacer arbitrairement dans l'un ou dans l'autre.
(Concrètement ce n'est pas forcément le cas : flot/stream...)

On notera :

- T le tableau contenant le texte et n sa longueur.
- M le tableau contenant le motif et k sa longueur.

(indexés à partir de 0)

STRUCTURES DE DONNÉES ET NOTATIONS

On considère que le texte et le motif sont stockés dans des tableaux, ce qui permet de se déplacer arbitrairement dans l'un ou dans l'autre.
(Concrètement ce n'est pas forcément le cas : flot/stream...)

On notera :

- T le tableau contenant le texte et n sa longueur.
- M le tableau contenant le motif et k sa longueur.

(indexés à partir de 0)

Par exemple, soit l'alphabet $A = \{a, b, c\}$.
Si on recherche le motif *abaa* dans *aacabacabaabaaa* :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*

M présent aux positions 7 et 10 de T. Ces deux occurrences se chevauchent : la lettre $a = T[10]$ est commune.

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
a	*b*	*a*	*a*											

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
	a	*b*	*a*	*a*										

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
		a	*b*	*a*	*a*									

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
			a	*b*	*a*	*a*								

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
							a	*b*	*a*	*a*				

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
							a	*b*	*a*	*a*				

```
i := 0 ; j := 0
while j<k && i ⩽ n-k
    j := 0
    while j<k && T[i+j] = M[j]
        j := j + 1
    i := i + 1
return i -1     // "motif absent" si j < k
```

UGA Université Grenoble Alpes

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
							a	*b*	*a*	*a*				

```
i := 0 ; j := 0
while j<k && i ⩽ n-k
    j := 0
    while j<k && T[i+j] = M[j]
        j := j + 1
    i := i + 1
return i -1      // "motif absent" si j < k
```

Complexité

- Au pire k boucles internes et $n - k$ boucles externes
  D'où $\mathcal{O}(k \times (n - k))$, soit $\mathcal{O}(k \times n)$ si $k \ll n$

UN ALGORITHME NAÏF

a	*a*	*c*	*a*	*b*	*a*	*c*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*a*
							a	*b*	*a*	*a*				

```
i := 0 ; j := 0
while j<k && i ⩽ n-k
    j := 0
    while j<k && T[i+j] = M[j]
        j := j + 1
    i := i + 1
return i -1      // "motif absent" si j < k
```

Complexité

- Au pire k boucles internes et $n - k$ boucles externes
  D'où $\mathcal{O}(k \times (n - k))$, soit $\mathcal{O}(k \times n)$ si $k \ll n$
- Pire cas atteint pour $M = bbba$ et $T = bbbbbbbbbbbb$

DES ÉCONOMIES POSSIBLES

Tests redondants

Lorsqu'un préfixe de M est présent à une position i de T :

- on examine **déjà** les caractères $T[i+1], T[i+2], \ldots$ avant d'échouer
- mais on ne conserve pas cette information
- donc on les examine à nouveau pour tester la présence de M à la position $i+1$

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

DES ÉCONOMIES POSSIBLES

Tests redondants

Lorsqu'un préfixe de M est présent à une position i de T :

- on examine **déjà** les caractères $T[i+1], T[i+2], \ldots$ avant d'échouer
- mais on ne conserve pas cette information
- donc on les examine à nouveau pour tester la présence de M à la position $i+1$

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

Qu'a-t-on appris ici ?

DES ÉCONOMIES POSSIBLES

Tests redondants

Lorsqu'un préfixe de M est présent à une position i de T :

- on examine **déjà** les caractères $T[i+1], T[i+2], \ldots$ avant d'échouer
- mais on ne conserve pas cette information
- donc on les examine à nouveau pour tester la présence de M à la position $i+1$

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

Qu'a-t-on appris ici ?

- $T[4] = b$ donc inutile de chercher $abaa$ pour $i = 4$;

DES ÉCONOMIES POSSIBLES

Tests redondants

Lorsqu'un préfixe de M est présent à une position i de T :

- on examine **déjà** les caractères $T[i+1], T[i+2], \ldots$ avant d'échouer
- mais on ne conserve pas cette information
- donc on les examine à nouveau pour tester la présence de M à la position $i+1$

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

Qu'a-t-on appris ici ?

- $T[4] = b$ donc inutile de chercher *abaa* pour $i = 4$;
- $T[5] = a$ donc on pourrait chercher *baa* à partir de $i = 6$;

DES ÉCONOMIES POSSIBLES

Tests redondants

Lorsqu'un préfixe de M est présent à une position i de T :

- on examine **déjà** les caractères $T[i+1], T[i+2], \ldots$ avant d'échouer
- mais on ne conserve pas cette information
- donc on les examine à nouveau pour tester la présence de M à la position $i+1$

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

Qu'a-t-on appris ici ?

- $T[4] = b$ donc inutile de chercher *abaa* pour $i = 4$;
- $T[5] = a$ donc on pourrait chercher *baa* à partir de $i = 6$;
- mais $T[6] \neq b$ et donc inutile de continuer la recherche à cet endroit.

L'IDÉE

Ce qu'il faut faire n'est pas toujours si clair

Si on recherche **maman** dans un texte commençant par **mamam...**, quelles sont les possibilités encore viables après l'échec ?

L'IDÉE

Ce qu'il faut faire n'est pas toujours si clair

Si on recherche **maman** dans un texte commençant par **mamam...**, quelles sont les possibilités encore viables après l'échec ?

On prend un parti pris :

- un caractère de T ne sera jamais lu plusieurs fois ;
- en cas d'échec on revient en arrière dans M autant que possible.

L'IDÉE

Ce qu'il faut faire n'est pas toujours si clair

Si on recherche **maman** dans un texte commençant par **mamam...**, quelles sont les possibilités encore viables après l'échec ?

On prend un parti pris :

- un caractère de T ne sera jamais lu plusieurs fois ;
- en cas d'échec on revient en arrière dans M autant que possible.

Pour cela, il ne suffit pas de connaître M sous forme d'un tableau :

- il faut analyser finement la structure de ce motif

L'IDÉE

Ce qu'il faut faire n'est pas toujours si clair

Si on recherche **maman** dans un texte commençant par **mamam...**, quelles sont les possibilités encore viables après l'échec ?

On prend un parti pris :

- un caractère de T ne sera jamais lu plusieurs fois ;
- en cas d'échec on revient en arrière dans M autant que possible.

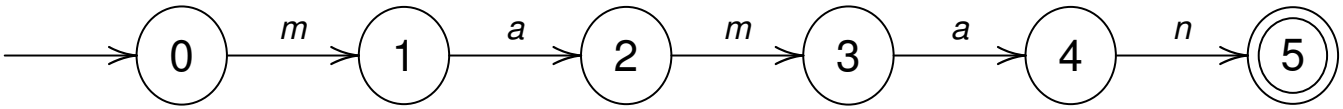
Pour cela, il ne suffit pas de connaître M sous forme d'un tableau :

- il faut analyser finement la structure de ce motif
- cela ne dépend pas de T
- on peut donc le faire **une fois pour toutes** en amont de l'algorithme : phase de pré-traitement
- si possible peu coûteux, et en tous cas qui accélère la suite

UGA Université Grenoble Alpes

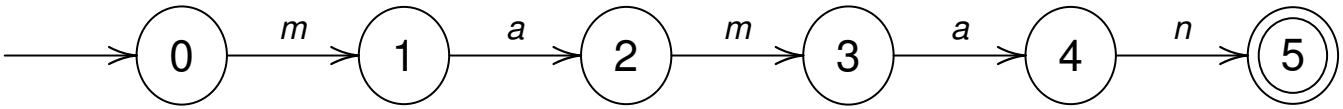
REPRÉSENTATION DE M COMME UN AUTOMATE

On cherche à mémoriser **toutes** les positions de M correspondant à la dernière lettre lue dans T.
Pour cela on construit l'automate qui reconnaît M :

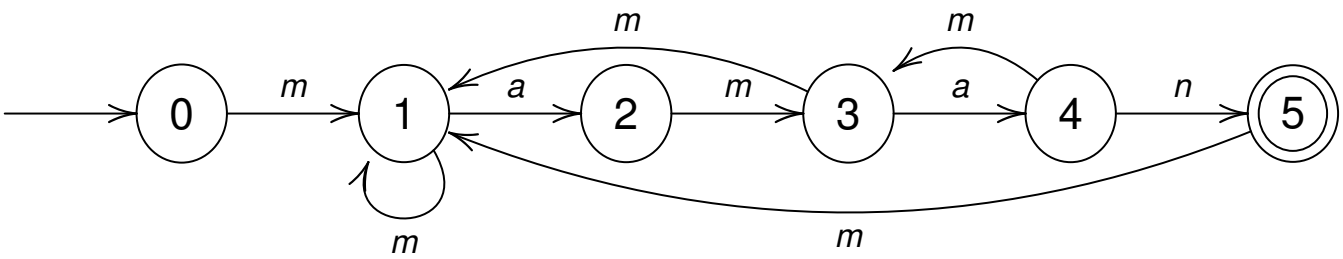


REPRÉSENTATION DE M COMME UN AUTOMATE

On cherche à mémoriser **toutes** les positions de M correspondant à la dernière lettre lue dans T.
Pour cela on construit l'automate qui reconnaît M :



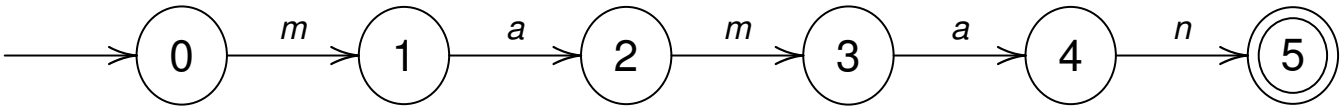
Puis on l'enrichit avec des retours vers les positions antérieures :



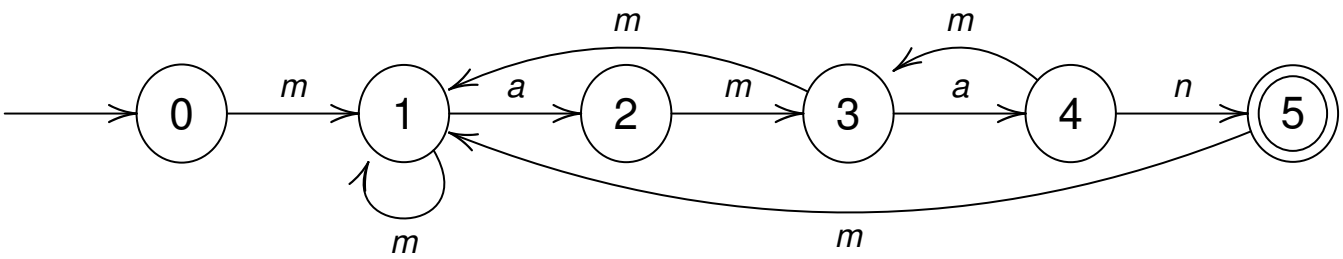
$j \xrightarrow{a}$ le plus long préfixe de M égal à un suffixe de $M[1..j]$

REPRÉSENTATION DE M COMME UN AUTOMATE

On cherche à mémoriser **toutes** les positions de M correspondant à la dernière lettre lue dans T.
Pour cela on construit l'automate qui reconnaît M :



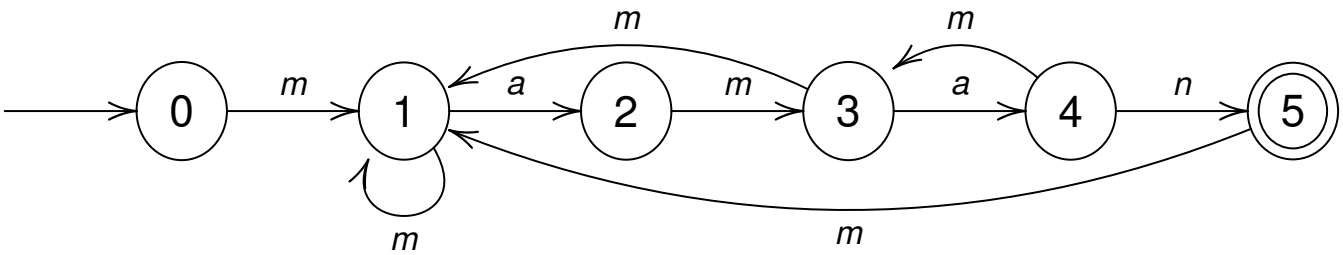
Puis on l'enrichit avec des retours vers les positions antérieures :



$j \xrightarrow{a}$ le plus long préfixe de M égal à un suffixe de $M[1..j]$

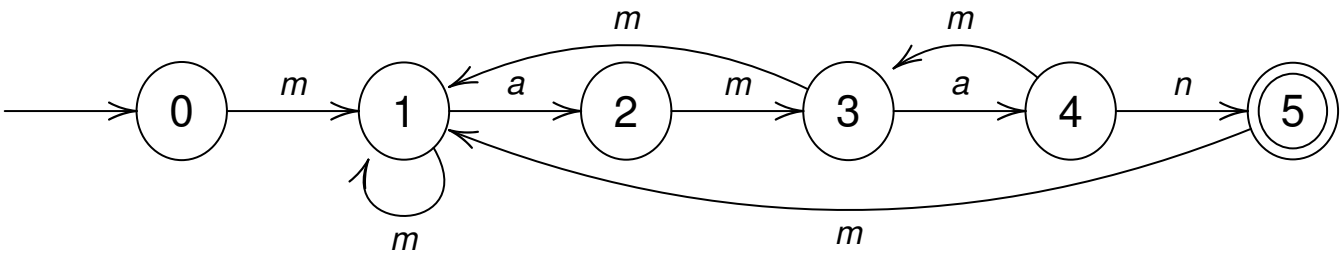
Implicitement, toutes les transitions manquantes reviennent à 0.

REPRÉSENTATION DE M COMME UN AUTOMATE : UTILISATION



Il suffit désormais de suivre les transitions de l'automate pour le texte T, l'état final marque la présence du motif.

REPRÉSENTATION DE M COMME UN AUTOMATE : UTILISATION

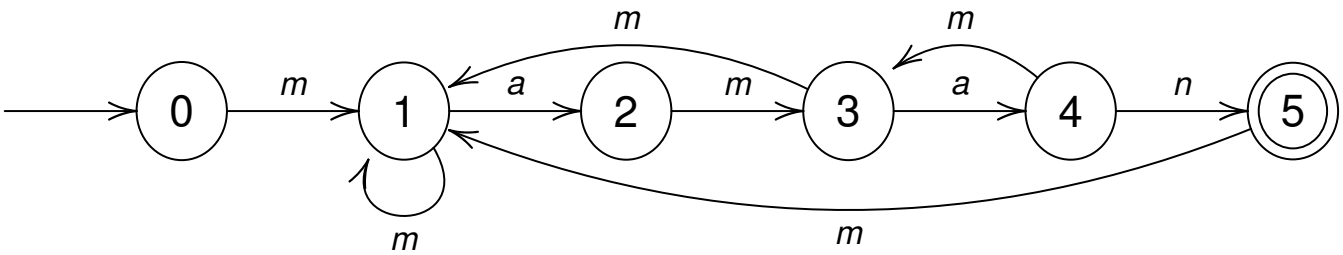


Il suffit désormais de suivre les transitions de l'automate pour le texte T, l'état final marque la présence du motif.

Complexité

Chaque caractère de T est lu une seule fois : recherche du motif en $O(n)$

REPRÉSENTATION DE M COMME UN AUTOMATE : UTILISATION



Il suffit désormais de suivre les transitions de l'automate pour le texte T, l'état final marque la présence du motif.

Complexité

Chaque caractère de T est lu une seule fois : recherche du motif en $O(n)$

Mais calcul de l'automate coûteux :

- Pour chaque position j dans M, pour chaque symbole de A,
  il faut tester l'égalité entre j préfixes de longueurs $1, 2 \ldots j$.
  Au total : $O(k^3 \times |A|)$
- En fait on peut se ramener à $O(k \times |A|)$
- Cependant il n'est exécuté qu'une seule fois en pré-traitement

LA RECHERCHE DU MOTIF

En cas d'échec on recule dans M indépendemment du prochain symbole :

LA RECHERCHE DU MOTIF

En cas d'échec on recule dans M indépendemment du prochain symbole :

- $D[j]$ donne la position **la plus à droite** dans M d'où reprendre

LA RECHERCHE DU MOTIF

En cas d'échec on recule dans M indépendemment du prochain symbole :

- $D[j]$ donne la position **la plus à droite** dans M d'où reprendre
- $D[j] = 0$ si rien ne convient et qu'il faut avancer dans T.

LA RECHERCHE DU MOTIF

En cas d'échec on recule dans M indépendemment du prochain symbole :

- $D[j]$ donne la position **la plus à droite** dans M d'où reprendre
- $D[j] = 0$ si rien ne convient et qu'il faut avancer dans T.

```
i := 0 ; j := 0
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    if j = k
        renvoyer i − k
renvoyer "motif absent"
```

UGA Université Grenoble Alpes

LA RECHERCHE DU MOTIF

En cas d'échec on recule dans M indépendemment du prochain symbole :

- $D[j]$ donne la position **la plus à droite** dans M d'où reprendre
- $D[j] = 0$ si rien ne convient et qu'il faut avancer dans T.

```
i := 0 ; j := 0
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    if j = k
        renvoyer i − k
renvoyer "motif absent"
```

LA RECHERCHE DU MOTIF

En cas d'échec on recule dans M indépendemment du prochain symbole :

- $D[j]$ donne la position **la plus à droite** dans M d'où reprendre
- $D[j] = 0$ si rien ne convient et qu'il faut avancer dans T.

```
i := 0 ; j := 0
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    if j = k
        renvoyer i − k
renvoyer "motif absent"
```

LA RECHERCHE DU MOTIF

En cas d'échec on recule dans M indépendemment du prochain symbole :

- $D[j]$ donne la position **la plus à droite** dans M d'où reprendre
- $D[j] = 0$ si rien ne convient et qu'il faut avancer dans T.

```
i := 0 ; j := 0
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    if j = k
        renvoyer i − k
renvoyer "motif absent"
```

UGA Université Grenoble Alpes

PRÉ-TRAITEMENT : CALCUL DE D

Objectif

Pour toute position j dans M, on cherche la longueur d du plus long préfixe de M de la forme $M[j-d \ldots j-1]$.
C'est cette longueur que l'on appellera $D[j]$.

PRÉ-TRAITEMENT : CALCUL DE D

Objectif

Pour toute position j dans M, on cherche la longueur d du plus long préfixe de M de la forme $M[j-d \ldots j-1]$.
C'est cette longueur que l'on appellera $D[j]$.

C'est plus simple que ça n'en a l'air :

> Si $M[j-d \ldots j-1]$ est un préfixe de M, alors $M[j-d \ldots j-2]$ aussi.

PRÉ-TRAITEMENT : CALCUL DE D

Objectif

Pour toute position j dans M, on cherche la longueur d du plus long préfixe de M de la forme $M[j-d \ldots j-1]$.
C'est cette longueur que l'on appellera $D[j]$.

C'est plus simple que ça n'en a l'air :

Si $M[j-d \ldots j-1]$ est un préfixe de M, alors $M[j-d \ldots j-2]$ aussi.

Par conséquent :

- Si $M[j-1] = M[d]$ alors alors on prolonge le préfixe (éventuellement vide) déjà reconnu
- Sinon on essaye de se rabattre sur un préfixe plus court,
  le meilleur est donné par $D[d]$
- S'il n'y a plus de préfixe plus court utilisable ($D[d] = 0$) on repart du début de M

PRÉ-TRAITEMENT : CALCUL DE D (SUITE)

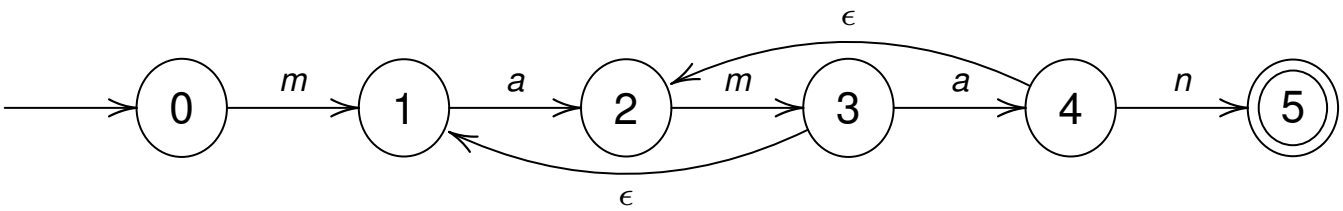
calcul_D(M)

Données: Un motif de longueur k
Résultat: La table D utilisée dans l'algorithme KMP
$D[0] := 0$
$D[1] := 0$
$d := 0$ est la longueur du plus long préfixe déjà identifié

for $j := 2$ **to** k
  while $d > 0$ **et** $M[j-1] \neq M[d]$
    $d := D[d]$
  if $M[j-1] = M[d]$
    $d := d + 1$
  $D[j] := d$
renvoyer D

TRADUCTION EN TERMES D'AUTOMATE

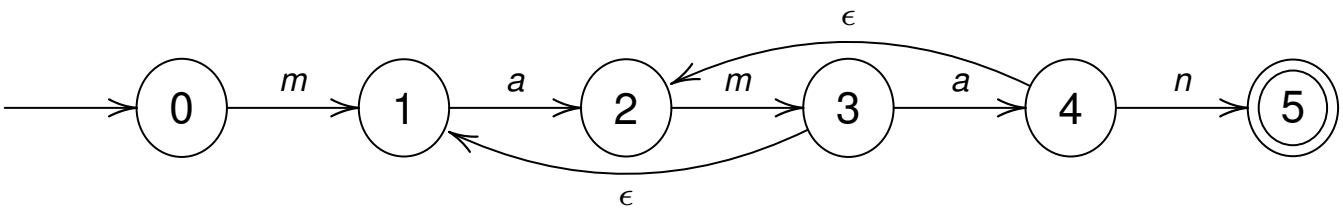
Cela revient à construire



où ϵ représente une **transition vide**

TRADUCTION EN TERMES D'AUTOMATE

Cela revient à construire



où ϵ représente une **transition vide**

On lève le non-déterminisme en imposant de ne suivre les transitions vides **que** si les transitions "vraies" ont échoué.

Comme il y a au maximum une transition vide à partir de chaque état, il y en a au maximum k au lieu des $k \times |A|$ précédentes.

COMPLEXITÉ : RECHERCHE DU MOTIF

On considère la quantité $2i - j$. À chaque itération du Tantque :

```
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    j = k
        return i − k
```

if

COMPLEXITÉ : RECHERCHE DU MOTIF

On considère la quantité $2i - j$. À chaque itération du Tantque :

```
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    j = k
        return i − k
```

$2i - j$ augmente de 1

if

COMPLEXITÉ : RECHERCHE DU MOTIF

On considère la quantité $2i - j$. À chaque itération du Tantque :

```
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    j = k
        return i − k
```

$2i - j$ augmente de 1

$2i - j$ augmente car j diminue

if

COMPLEXITÉ : RECHERCHE DU MOTIF

On considère la quantité $2i - j$. À chaque itération du Tantque :

```
while i < n
    if T[i] = M[j]
        i := i + 1
        j := j + 1
    else if D[j] > 0
        j := D[j]
    else
        i := i + 1
        j := 0
    j = k
        return i − k
```

- `if T[i] = M[j]` : $2i - j$ augmente de 1
- `else if D[j] > 0` : $2i - j$ augmente car j diminue
- `else` : $2i - j$ augmente d'au moins 2

if

COMPLEXITÉ : RECHERCHE DU MOTIF

On considère la quantité $2i - j$. À chaque itération du Tantque :

while $i < n$
- **if** $T[i] = M[j]$
  - $i := i + 1$
  - $j := j + 1$

  } $2i - j$ augmente de 1
- **else if** $D[j] > 0$
  - $j := D[j]$

  } $2i - j$ augmente car j diminue
- **else**
  - $i := i + 1$
  - $j := 0$

  } $2i - j$ augmente d'au moins 2

  if
- $j = k$
  - **return** $i - k$

Comme $2i - j$ est positif et majoré par $2n$, il y a au pire $2n$ itérations.

COMPLEXITÉ : PRÉTRAITEMENT

On considère cette fois $2j - d$:

for $j := 2$ **to** k
  while $d > 0$ **et** $M[j-1] \neq M[d]$
    $d := D[d]$
  if $M[j-1] = M[d]$
    $d := d + 1$
  $D[j] := d$

COMPLEXITÉ : PRÉTRAITEMENT

On considère cette fois $2j - d$:

```
for j := 2 to k
    while d > 0 et M[j − 1] ≠ M[d]
        d := D[d]
    if M[j − 1] = M[d]
        d := d + 1
    D[j] := d
```

} $2j - d$ augmente car d diminue

COMPLEXITÉ : PRÉTRAITEMENT

On considère cette fois $2j - d$:

```
for j := 2 to k
    while d > 0 et M[j − 1] ≠ M[d]
        d := D[d]                          } 2j − d augmente car d diminue
    if M[j − 1] = M[d]                     } 2j − d diminue de 1
        d := d + 1
    D[j] := d
```

COMPLEXITÉ : PRÉTRAITEMENT

On considère cette fois $2j - d$:

```
for j := 2 to k                                          } mais 2j − d augmente de 2
    while d > 0 et M[j − 1] ≠ M[d]                       }
        d := D[d]                                        } 2j − d augmente car d diminue
    if M[j − 1] = M[d]                                   }
        d := d + 1                                       } 2j − d diminue de 1
    D[j] := d
```

COMPLEXITÉ : PRÉTRAITEMENT

On considère cette fois $2j - d$:

```
for j := 2 to k                                          } mais 2j − d augmente de 2
    while d > 0 et M[j − 1] ≠ M[d]
        d := D[d]                                        } 2j − d augmente car d diminue
    if M[j − 1] = M[d]                                   } 2j − d diminue de 1
        d := d + 1
    D[j] := d
```

Cette quantité étant majorée par $2k$, le pré-traitement est en $\mathcal{O}(k)$.

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...

```
CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...

```
  CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...

```
  CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...

```
  CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...
- ... mais on teste l'alignement du motif sur le texte en partant de la droite du motif
- Décalage optimisé :
  - pas de `A` dans le motif
  - on saute directement à la position courante + 6 (longueur du motif)

```
CGGCTC

ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...
- ... mais on teste l'alignement du motif sur le texte en partant de la droite du motif
- Décalage optimisé :
  - pas de `A` dans le motif
  - on saute directement à la position courante + 6 (longueur du motif)

```
   CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...
- ... mais on teste l'alignement du motif sur le texte en partant de la droite du motif
- Décalage optimisé :
  - il existe un autre C dans le motif
  - on saute directement à ce C

```
      CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...
- ... mais on teste l'alignement du motif sur le texte en partant de la droite du motif
- Décalage optimisé :
  - il existe un autre G dans le motif
  - on saute directement à ce G

```
      CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...
- ... mais on teste l'alignement du motif sur le texte en partant de la droite du motif

```
          CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

Algorithme de Boyer-Moore

- Plus rapide des algorithmes connus
- Retour sur le glissé du motif sur le texte, utilisé dans l'algorithme naïf
- On glisse le motif de gauche à droite...
- ... mais on teste l'alignement du motif sur le texte en partant de la droite du motif

```
          CGGCTC
ATAACAGGAGTAAATAACGGCTCGAGTAAATA
```

seulement 11 comparaisons,

là où l'algorithme naïf en aurait eu besoin de 24 et l'automate 23...

Version simple : algorithme de Boyer-Moore-Horspool

- On compare le motif M avec un facteur $T[p..p+m-1]$ du texte
- Si une différence entre M et ce facteur est constatée (ou si on a trouvé une occurrence du motif mais qu'on les veut toutes), on décale le motif vers la droite de manière à faire coïncider la dernière lettre du facteur, c'est-à-dire la lettre $T[p+m-1]$, avec son occurrence la plus à droite dans M

chaîne	a	b	a	c	a	c	a	**b**	a	**c**	a	b	a	**b**	a	**a**	**b**	b	**a**	b
(1)	a	b	a	c	a	b	a	*c*												
(2)			a	*b*	a	c	a	b	a	c										
(3)							a	b	a	c	a	b	a	*c*						
(4)									a	b	a	c	a	b	a	*c*				
(5)										a	b	a	c	a	b	a	*c*			
(6)												a	b	a	c	a	b	a	*c*	

Complexité ?

- Première remarque : premier algorithme qui ne lit pas nécessairement tous les caractères du texte !
- Complexité dans le pire des cas $O(nm)$
- En pratique (plus précisément, en moyenne sur des motifs aléatoires), l'algorithme de Horspool effectue un nombre de comparaisons de l'ordre de $O\left(\frac{n}{m} + \frac{n}{2|\Sigma|}\right)$

Merci à Benjamin Mommege DIU EIL Marseille

SYNTHÈSE

- La recherche des occurrences d'un motif de longueur k dans un texte de longueur n est de l'ordre de $n \times k$
- Un prétraitement (calcul de l'automate des préfixes) permet de rendre le coût de l'ordre de n (mais avec un surcoût au début)
- En parcourant le motif par suffixes croissant on peut encore diminuer le nombre de lectures (mais on change de sens dans le parcours du texte) Algorithme de Boyer Moore
- On peut également restreindre le test de motif par une fonction adéquate Algorithme de Karp-Rabin