Algorithmique avancée : Algorithmique du texte

Serge Haddad

LMF, ENS Paris-Saclay & CNRS

L3 et M2 FESUP Informatique

Motivation

Le problème central : la recherche de motif dans un texte.

- présent dans tous les traitements de texte
- étendu aux expressions rationnelles
- et à la recherche approximative.

mais aussi ...

- gestion de dictionnaire ;
- compression de texte ;
- recherche de régularités.

Des applications diversifiées (e.g. en bio-informatique).

Plan

Une première solution

Le problème.

- **Entrée.** T un texte de n caractères et P un motif de $m \leq n$ caractères
- **Résultat.** $\{i \mid i \leq n - m + 1 \wedge T[i, i + m - 1] = P\}$

Un algorithme naïf.

$L \leftarrow \emptyset$;
For i **from** 0 **do** $n - m$ **do**
  $j \leftarrow 1$;
  While $1 \leq j \leq m$ **do**
    If $T[i + j] = P[j]$ **then** $j \leftarrow j + 1$ **else** $j \leftarrow 0$;
  If $j = m + 1$ **then** $L \leftarrow L \cup \{i + 1\}$;

Complexité en $\Theta(nm)$ (e.g. $T = a^n$ et $P = a^m$)

L'approche automate

Soit un préfixe $T[1, i]$ pour $m \leq i \leq n$,
$T[i - m + 1, i] = P$ si et seulement si $T[1, i] \in \Sigma^* P$

Principe de l'approche déterministe

- Construction de $\mathcal{A}$, automate déterministe complet tel que $\mathcal{L}(\mathcal{A}) = \Sigma^* P$;
- Maintien de l'état courant initialisé à l'état initial de $\mathcal{A}$;
- Pour i de 1 à n, mise à jour de l'état courant à la lecture de $T[i]$;
- Appariement du motif lorsque l'état courant est un état final.

Principe de l'approche non déterministe

- Construction de $\mathcal{A}$, automate non déterministe tel que $\mathcal{L}(\mathcal{A}) = \Sigma^* P$;
- Maintien d'un ensemble d'états initialisé à l'ensemble des états initiaux ;
- Pour i de 1 à n, mise à jour de l'ensemble courant à la lecture de $T[i]$;
- Appariement du motif lorsque l'un des états courants est un état final.

Préfixes et suffixes

Notations. $x \sqsubset y$ si x est un préfixe de y et $x \sqsupset y$ si x est un suffixe de y.

Observation. $y \sqsupset x$, $z \sqsupset x$ et $|z| \leq |y|$ implique $z \sqsupset y$.

$$\boxed{\sigma(x) = \max(k \mid 0 \leq k \leq m \wedge P[1,k] \sqsupset x)}$$

Proposition. Soit $x \in \Sigma^*$, $a \in \Sigma$ et $k = \sigma(x)$. Alors $\sigma(xa) = \sigma(P[1,k]a)$.

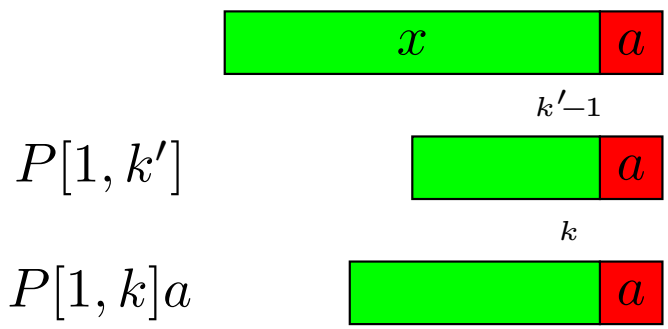
Preuve. Puisque $P[1,k]a \sqsupset xa$, pour tout y, $y \sqsupset P[1,k]a$ implique $y \sqsupset xa$.
Par conséquent $\sigma(xa) \geq \sigma(P[1,k]a)$.

Soit $k' = \sigma(xa)$.

- Si $k' = 0$ alors $0 = \sigma(xa) \leq \sigma(P[1,k]a)$.
- Si $k' > 0$ alors $P[k'] = a$ et $P[1,k'-1] \sqsupset x$.

  D'où $k' - 1 \leq k$ par définition de k. Donc $P[1,k'-1] \sqsupset P[1,k]$.

  Puis $P[1,k'] \sqsupset P[1,k]a$, ce qui implique $k' \leq \sigma(P[1,k]a)$.

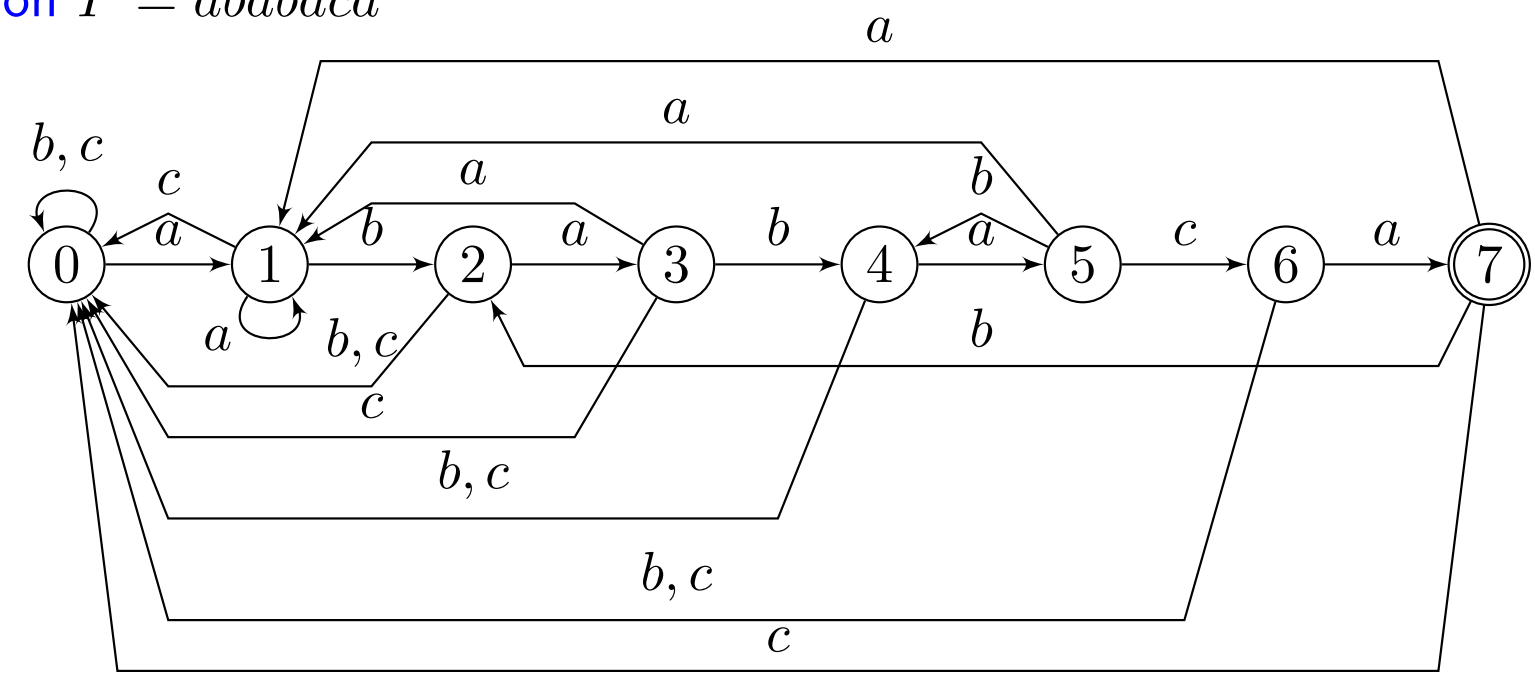


L'automate du motif

Un automate déterministe complet

- Les états $\{0, \ldots, m\}$ avec 0 l'état initial et m l'état final
- Les transitions $\{k \xrightarrow{a} k' \mid \sigma(P[1,k]a) = k'\}$

Illustration $P = abababaca$



Lors de la lecture de T, on maintient l'état courant
et on signale un appariement lorsqu'on visite l'état m.

Minimalité de l'automate

Tout automate non déterministe qui reconnaît $\Sigma^* P$ a au moins $m+1$ états.

Preuve par l'absurde.

Soit $\mathcal{A}$, un automate qui reconnaît $\Sigma^* P$ avec au plus m états.

Soit $q_0 \xrightarrow{P[1]} q_1 \xrightarrow{P[2]} q_2 \cdots q_{m-1} \xrightarrow{P[m]} q_m$ tel que q_m soit un état final.

Il existe $i < j$ tel que $q_i = q_j$. D'où :

- si $j < m$ $q_0 \xrightarrow{P[1]} q_1 \cdots q_i \xrightarrow{P[j+1]} q_{j+1} \cdots q_{m-1} \xrightarrow{P[m]} q_m$ impliquant $P[1,i]P[j+1,m] \in \Sigma^* P$, absurde.
- si $j = m$ $q_0 \xrightarrow{P[1]} q_1 \cdots q_{i-1} \xrightarrow{P[i]} q_m$ impliquant $P[1,i] \in \Sigma^* P$, absurde.

Complexité de l'approche automate

Construction de l'automate en $\Theta(m^3|\Sigma|)$

For i **from** 0 **do** m **do**
  For $a \in \Sigma$ **do**
    $j \leftarrow \min(i+1, m)$
    While not $P[1,j] \sqsupset P[1,i]a$ **do** $j \leftarrow j-1$
    $\delta(i,a) \leftarrow j$

Reconnaissance du motif en $\Theta(n)$

La fonction π

- π est la fonction de $\{1, \ldots, m\}$ dans $\{0, \ldots, m-1\}$

telle que $P[1, \pi(k)]$ soit le plus grand suffixe propre de $P[1, k]$:

$$\pi(k) = \max(k' < k \mid P[1, k'] \sqsupset P[1, k])$$

- Illustration $P = ababaca$

k	1	2	3	4	5	6	7
$\pi(k)$	0	0	1	2	3	0	1

Relation entre π et δ

Soient $a \in \Sigma$ et $k \in \{0, \ldots m\}$, alors :

1. Si $k < m \wedge P[k+1] = a$ alors $\delta(k, a) = k + 1$
2. Si $(0 < k < m \wedge P[k+1] \neq a) \vee k = m$ alors $\delta(k, a) = \delta(\pi(k), a)$
3. Si $k = 0 \wedge P[k+1] \neq a$ alors $\delta(k, a) = k$

Preuve Soit $k' = \delta(k, a) = \sigma(P[1, k]a)$, $k' \leq k + 1$.

1. Immédiat puisque $P[1, k]a = P[1, k+1]$.
2. Si $0 < k < m$, et $a \neq P[k+1]$, $P[1, k']$ est un suffixe propre de $P[1, k]a$.
   Si $k = m$ alors $k' \leq k$ et $P[1, k']$ est un suffixe propre de $P[1, k]a$.

   $P[1, \pi(k)]a \sqsupset P[1, k]a$ et $\sigma(P[1, \pi(k)]a) \leq \sigma(P[1, k]a)$.

   - Si $k' = 0$ alors $0 \leq \sigma(P[1, \pi(k)]a) \leq \sigma(P[1, k]a) = 0$
   - Sinon $P[1, k'-1]$ est un suffixe propre de $P[1, k]$ et un suffixe de $P[1, \pi(k)]$.
     Par conséquent, $P[1, k']$ est un suffixe de $P[1, \pi(k)]a$.
     D'où $k' \leq \sigma(P[1, \pi(k)]a)$ puis $k' = \sigma(P[1, \pi(k)]a)$.
3. $k' \in \{0, 1\}$ et $a \neq P[1]$ implique $k' = 0$.

Construction efficace de l'automate

Construction de l'automate en $\Theta(m|\Sigma|)$

For $a \in \Sigma$ **do**

  If $a = P[1]$ **then** $\delta(0, a) \leftarrow 1$ **else** $\delta(0, a) \leftarrow 0$

For i **from** 1 **do** m **do**

  For $a \in \Sigma$ **do**

    If $i < m$ **and** $a = P[i+1]$ **then** $\delta(i, a) \leftarrow i+1$ **else** $\delta(i, a) \leftarrow \delta(\pi(i), a)$

Comment calculer π ?

Les ensembles $\pi^*(i)$

L'ensemble d'indices $\pi^*(i)$ défini inductivement par :

$$\pi^*(0) = \{0\} \text{ et pour } i > 0,\ \pi^*(i) = \{i\} \cup \pi^*(\pi(i))$$

$$\boxed{\forall i\ \pi^*(i) = \{j \mid P[1,j] \sqsupset P[1,i]\}}$$

Preuve par récurrence. Immédiat pour $i = 0$.

- Soit $i > 0$, $P[1,i] \sqsupset P[1,i]$.
  Soit j tel que $j \in \pi^*(\pi(i))$.
  Par hypothèse de récurrence, $P[1,j] \sqsupset P[1,\pi(i)]$ et donc $P[1,j] \sqsupset P[1,i]$.
  Donc $\pi^*(i) \subseteq \{j \mid P[1,j] \sqsupset P[1,i]\}$.
- Soit j tel que $P[1,j] \sqsupset P[1,i]$.
  Si $j = i$ alors $j \in \pi^*(i)$.
  Sinon $P[1,j]$ est un préfixe de P et suffixe propre de $P[1,i]$.
  Puisque $P[1,\pi(i)]$ est le plus grand préfixe de P et suffixe propre de $P[1,i]$, $P[1,j] \sqsupset P[1,\pi(i)]$.
  Donc par hypothèse de récurrence, $j \in \pi^*(\pi(i)) \subseteq \pi^*(i)$.

Une propriété de π

Soit $1 < i \leq m$ et $E_i = \{j \mid j \in \pi^*(\pi(i-1)) \wedge P[j+1] = P[i]\}$.

Pour tout i, si $E_i = \emptyset$ alors $\pi(i) = 0$ sinon $\pi(i) = 1 + \max(E_i)$

Preuve.

Prouvons que $E_i = \{j < i-1 \mid P[1, j+1] \sqsupset P[1, i]\}$.

- Si $j \in E_i$ alors $P[1, j] \sqsupset P[1, \pi(i-1)] \sqsupset P[1, i-1]$ et $P[j+1] = P[i]$.
  Donc $P[1, j+1]$ est un suffixe propre de $P[1, i]$.
  D'où $E_i \subseteq \{j < i-1 \mid P[1, j+1] \sqsupset P[1, i]\}$.
- Soit $j \in \{j < i-1 \mid P[1, j+1] \sqsupset P[1, i]\}$.
  Donc $P[1, j]$ est un suffixe propre de $P[1, i-1]$ et $P[j+1] = P[i]$.
  D'où $P[1, j] \in \pi^*(\pi(i-1))$.
  Par conséquent $\{j < i-1 \mid P[1, j+1] \sqsupset P[1, i]\} \subseteq E_i$.
- Si $\{j < i-1 \mid P[1, j+1] \sqsupset P[1, i]\} = E_i = \emptyset$ alors $\pi(i) = 0$.
  Sinon $\pi(i) = 1 + \max(j < i-1 \mid P[1, j+1] \sqsupset P[1, i]) = 1 + \max(E_i)$.

Calcul de la fonction π

$\pi(1) \leftarrow 0$; $j \leftarrow 0$

For i **from** 2 **to** m **do**

$(j = \pi(i-1))$

While $j > 0$ **and** $P[j+1] \neq P[i]$ **do** $j \leftarrow \pi(j)$

If $P[j+1] = P[i]$ **then** $j \leftarrow j+1$

$\pi(i) \leftarrow j$

Complexité en $\Theta(m)$.

Soit $2j$ la fonction de potentiel.

Alors les instructions de la boucle **For**, y compris le **While** ont une complexité amortie constante.

Plan

Syntaxe d'un automate étendu

$\mathcal{D} = \{\uparrow, \rightarrow\}$ est l'ensemble des déplacements.

$\mathcal{F}(\Sigma)$ est l'ensemble des formules défini inductivement par :

- $\mathbf{true} \in \mathcal{F}(\Sigma)$ et pour tout $a \in \Sigma$, $a \in \mathcal{F}(\Sigma)$;
- Si $\varphi, \psi \in \mathcal{F}(\Sigma)$ alors $\neg\varphi, \varphi \wedge \psi \in \mathcal{F}(\Sigma)$.

$\models$ la relation de satisfaction est définie inductivement par :

- Pour tout $a, b \in \Sigma$, $a \models \mathbf{true}$ et $a \models b$ ssi $a = b$;
- Pour tout $a \in \Sigma$, $\varphi, \psi \in \mathcal{F}(\Sigma)$,
  $a \models \neg\varphi$ ssi $a \not\models \varphi$, $a \models \varphi \wedge \psi$ ssi $a \models \varphi$ et $a \models \psi$.

On note $[\varphi] = \{a \mid a \models \varphi\}$.

Un automate étendu $\mathcal{A}$ est défini par :

- Q un ensemble finis d'états,
  $q_0 \in Q$ l'état initial et $F \subseteq Q$ le sous-ensemble des états finals.
- $\delta \subseteq Q \times \mathcal{F}(\Sigma) \times \mathcal{D} \times Q$, la relation de transition.
  Une transition (q, φ, d, q') est notée $q \xrightarrow{\varphi, d} q'$
- Pour tout $q_{\alpha(0)} \xrightarrow{\varphi_1, \uparrow} q_{\alpha(1)}, \ldots, q_{\alpha(n-1)} \xrightarrow{\varphi_n, \uparrow} q_{\alpha(n)} = q_{\alpha(0)}$, $\bigcap_{i=1}^{n} [\varphi_i] = \emptyset$.

Sémantique d'un automate étendu

Un automate étendu est dit *déterministe complet* si pour tout état q et $\{\varphi_i\}_{i\leq k}$ les formules étiquetant les transitions sortantes de q :

$$\Sigma = \biguplus_{i\leq k} [\varphi_i]$$

Soit un automate étendu déterministe complet, $\overline{\delta}(q, w)$ l'état atteint par lecture de w à partir de q est défini inductivement ainsi :

- Si $w = \varepsilon$ alors $\overline{\delta}(q, w) = q$;
- Si $w = aw'$ et si $q \xrightarrow{\varphi, d} q'$ avec $a \models \varphi$ alors $\overline{\delta}(q, w) = \overline{\delta}(q', w'')$ où $w'' = w'$ si $d = \rightarrow$ et $w' = w$ si $d = \uparrow$.

Cette définition est bien fondée en raison de la contrainte sur les circuits de transitions.

L'automate de Morris-Pratt

À partir des propriétés suivantes :

Soient $a \in \Sigma$ et $k \in \{0, \dots m\}$, alors :

1. Si $k < m \wedge P[k+1] = a$ alors $\delta(k, a) = k + 1$
2. Si $(0 < k < m \wedge P[k+1] \neq a) \vee k = m$ alors $\delta(k, a) = \delta(\pi(k), a)$
3. Si $k = 0 \wedge P[k+1] \neq a$ alors $\delta(k, a) = k$

On construit un automate étendu qui reconnaît le même langage .
L'ensemble des états est $\{0, 1, \dots, m\}$ avec $q_0 = 0$ et $F = \{m\}$.

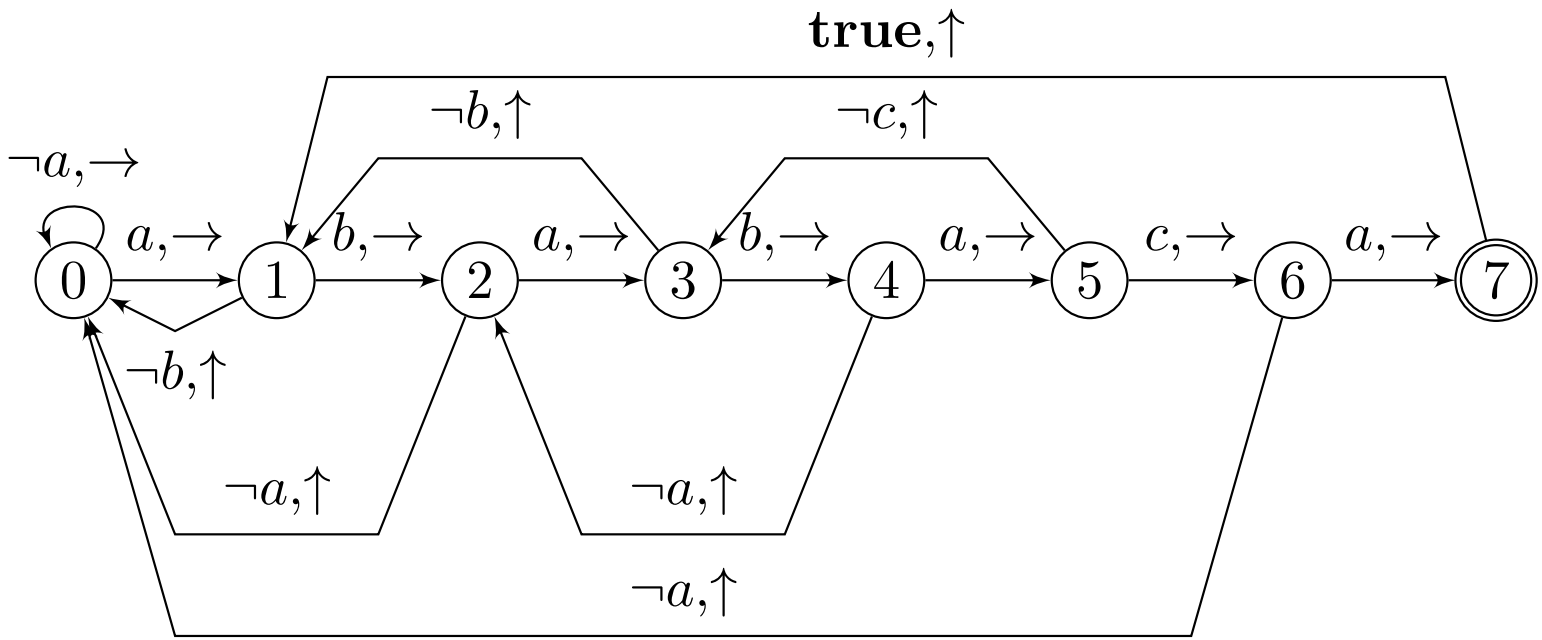
1. Pour tout $k < m$, $k \xrightarrow{P[k+1], \rightarrow} k + 1$.
2. Pour tout $0 < k < m$, $k \xrightarrow{\neg P[k+1], \uparrow} \pi(k)$.
3. $m \xrightarrow{\mathbf{true}, \uparrow} \pi(m)$.
4. $0 \xrightarrow{\neg P[1], \rightarrow} 0$.

Cette construction s'effectue en $\Theta(m)$.

Illustration

$$P = ababaca$$

k	1	2	3	4	5	6	7
$\pi(k)$	0	0	1	2	3	0	1



Complexité de Morris-Pratt

Lors de la lecture d'un texte T de longueur n par l'automate de Morris-Pratt, on parcourt au plus $2n - 1$ transitions.

Preuve.

- Soit k, l'état atteint après lecture de T par l'automate.
- Soit n^+, le nombre de transitions parcourues qui ont incrémenté l'état *(d'une unité)*.
- Soit n^-, le nombre de transitions parcourues qui ont décrémenté l'état.
- Soit $n^=$, le nombre de transitions parcourues qui ont laissé l'état inchangé *(la boucle autour de l'état 0)*.

On a $n = n^+ + n^=$ et $n^+ - n^- \geq k$.

La dernière transition parcourue « consomme » le dernier caractère.

- S'il s'agit de la boucle en 0, alors $n^= > 0$ et $n^+ - n^- \geq 0$.
  D'où $n^= + n^+ + n^- \leq n^= + 2n^+ < 2n^= + 2n^+ = 2n$.
- S'il ne s'agit pas de la boucle en 0, alors $k > 0$ et $n^+ - n^- > 0$.
  D'où $n^= + n^+ + n^- < n^= + 2n^+ \leq 2n^= + 2n^+ = 2n$.

L'automate de Knuth-Morris-Pratt

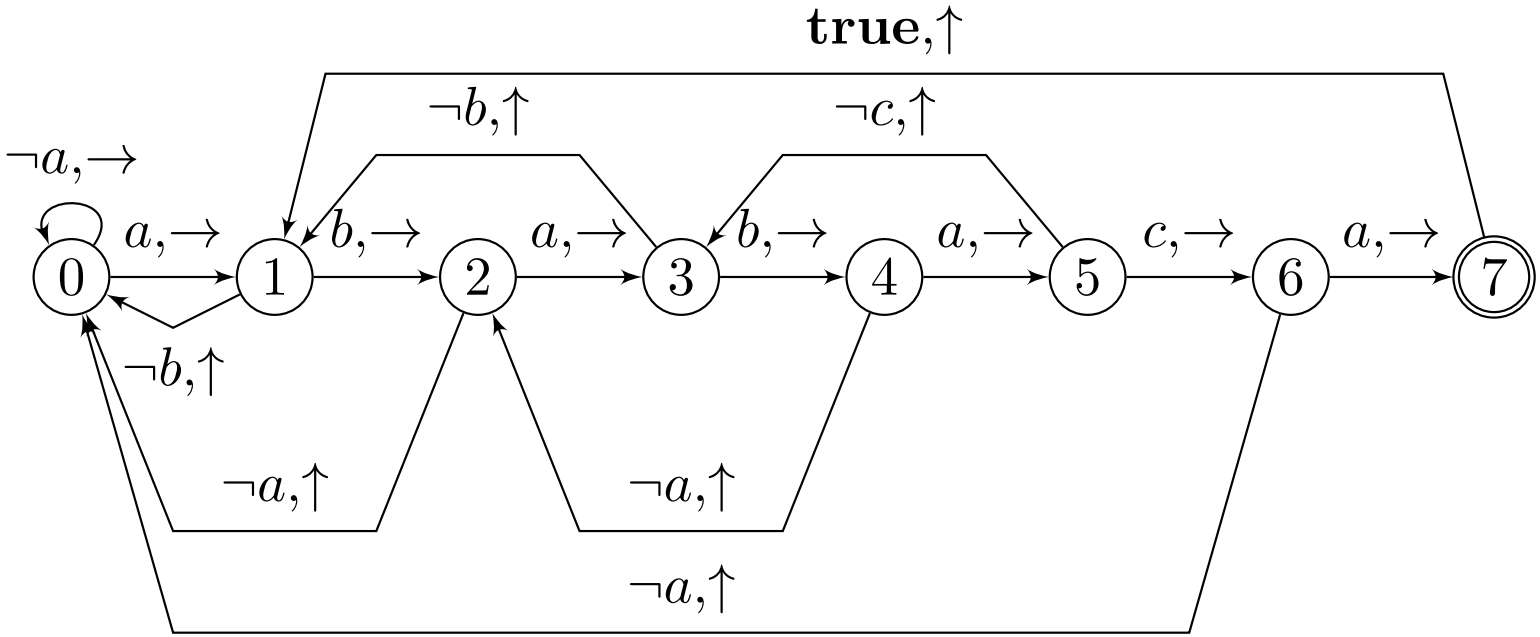
Observations à propos de l'automate de Morris-Pratt.

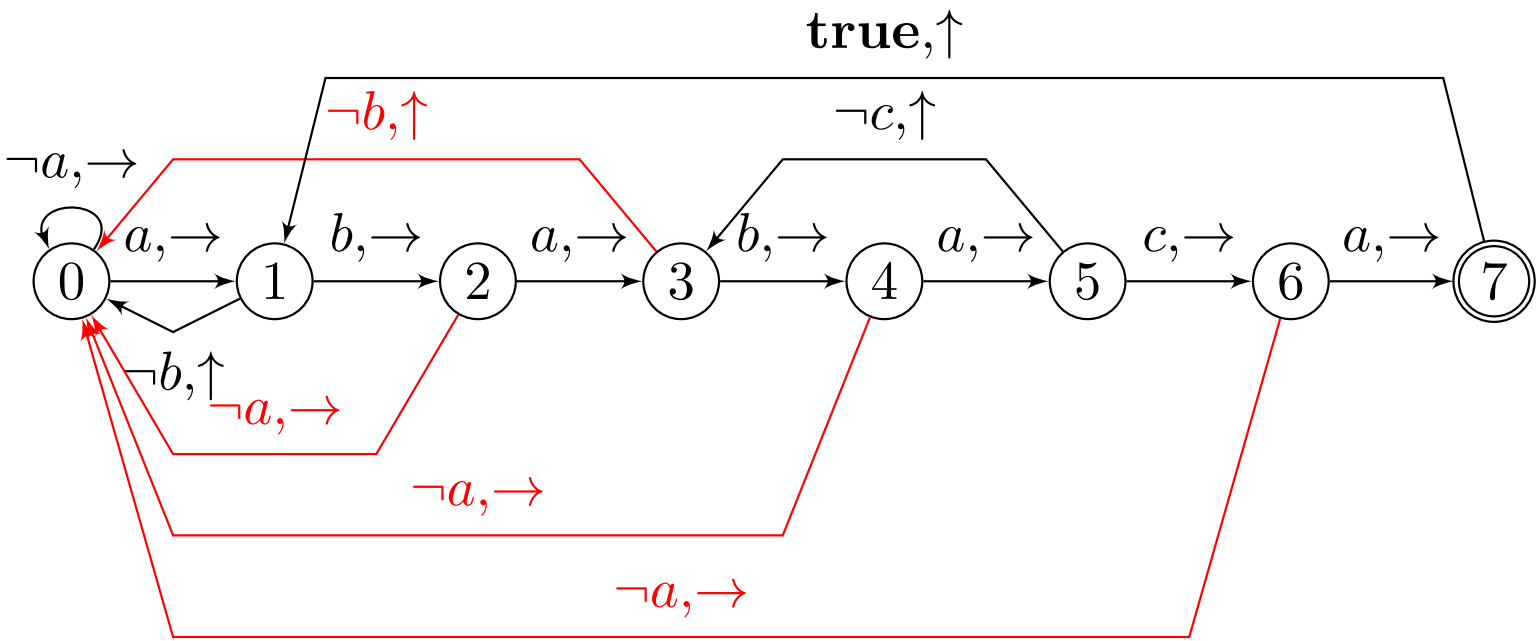
- à un chemin maximal de transitions
  $k \xrightarrow{\neg a,\uparrow} \pi(k) \xrightarrow{\neg a,\uparrow} \dots \pi^{(n_k)}(k) = 0 \xrightarrow{\neg a,\rightarrow} 0$,
  on peut substituer une unique transition $k \xrightarrow{\neg a,\rightarrow} 0$.
- à un chemin maximal de transitions
  $k \xrightarrow{\neg a,\uparrow} \pi(k) \xrightarrow{\neg a,\uparrow} \dots \pi^{(n_k)}(k)$,
  on peut substituer une unique transition $k \xrightarrow{\neg a,\uparrow} \pi^{(n_k)}(k)$.

L'automate de Knuth-Morris-Pratt est obtenu par ses substitutions.

Le temps de lecture de T par l'automate de Knuth-Morris-Pratt
est toujours plus petit ou égal à celui de T par l'automate de Morris-Pratt.

Illustration





Construction de l'automate KMP

Idée. Effectuer la « fermeture transitive » à la volée.

For i **from** 0 **do** $m-1$ **do Add** $i \xrightarrow{P[i+1],\rightarrow} i+1$

Add $0 \xrightarrow{\neg P[1],\rightarrow} 0$

Add $m \xrightarrow{\mathbf{true},\uparrow} \pi(m)$

For i **from** 1 **do** $m-1$ **do**

  Let $\pi(i) \xrightarrow{\neg P[\pi(i)+1],d} j$ outgoing from $\pi(i)$

  If $P[i+1] = P[\pi(i)+1]$ **then**

    Add $i \xrightarrow{\neg P[i+1],d} j$

  Else

    Add $i \xrightarrow{\neg P[i+1],\uparrow} \pi(i)$

La fonction π'

La fonction π' de $\{0, \ldots, m\}$ vers $\{-1, \ldots, m-1\}$ est définie par :

- $\pi'(m) = \pi(m)$
- Soit $0 \leq i < m$ et la transition $i \xrightarrow{\neg P[i+1], d} j$;
- Si $d = \rightarrow$ (ce qui implique $j = 0$) alors $\pi'(i) = -1$ sinon $\pi'(i) = j$.

k	0	1	2	3	4	5	6	7
$\pi'(k)$	-1	0	-1	0	-1	3	-1	1

Propriétés.

Définitions alternatives.

- $\pi'(m) = \pi(m)$, $\pi'(0) = -1$ et pour tout $0 < i < m$,

1. Si $P[\pi(i) + 1] \neq P[i + 1]$ alors $\pi'(i) = \pi(i)$ sinon $\pi'(i) = \pi'(\pi(i))$
2. $\pi'(i) = \max(\pi^{(k)}(i) \mid P[\pi^{(k)}(i) + 1] \neq P[i + 1])$
   avec par convention $\max(\emptyset) = -1$.

Soit δ la fonction de transition de l'automate du motif, $a \in \Sigma$,

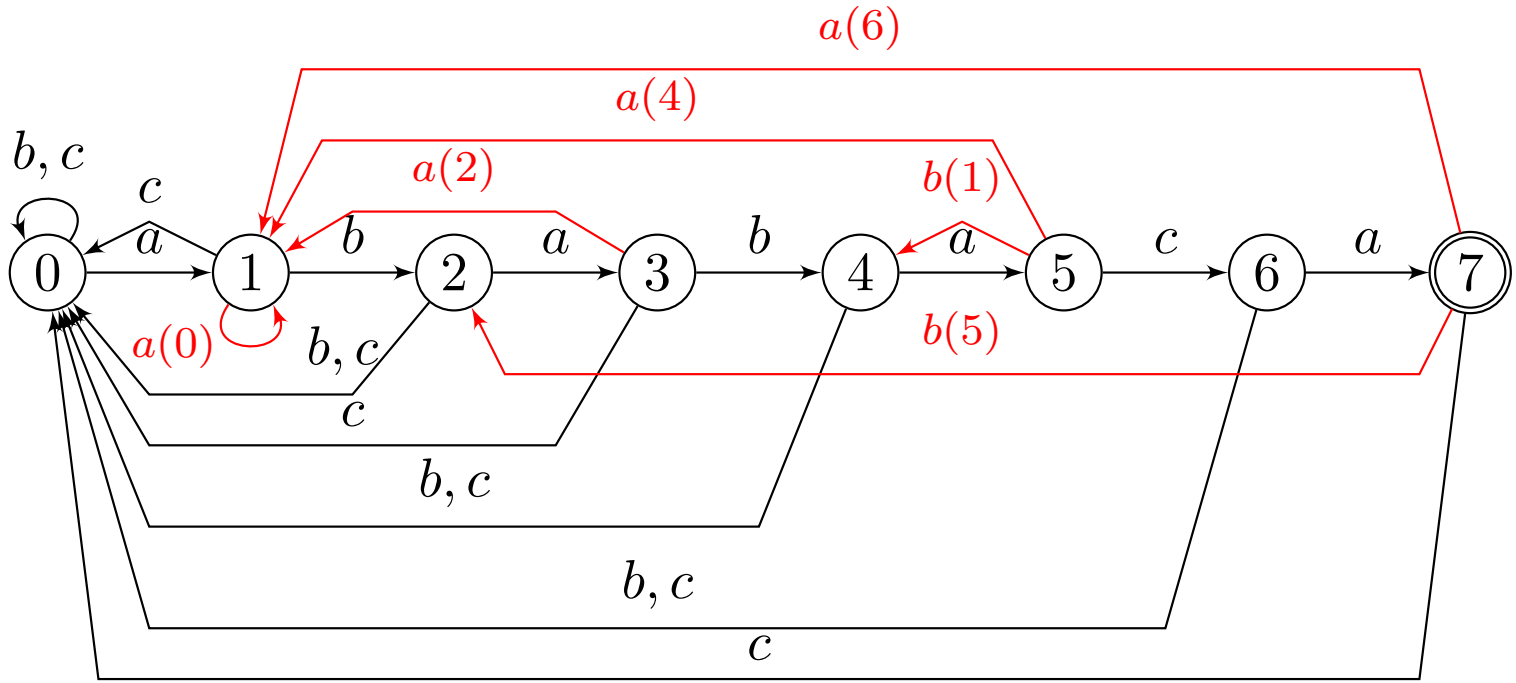
$0 < i \leq m$ un état et ($i = m$ ou $a \neq P[i + 1]$) alors :

Si $\pi'(i) = -1$ alors $\delta(i, a) = 0$ sinon $\delta(i, a) = \delta(\pi'(i), a)$.

Plan

Transitions arrières

Une transition $q \xrightarrow{a} r$ est dite *arrière* si $0 < r \leq q$. Son *amplitude* est $q - r$.



Analyse de l'automate du motif

Une transition $q \xrightarrow{a} r$ est dite *arrière* si $0 < r \leq q$. Son *amplitude* est $q - r$.

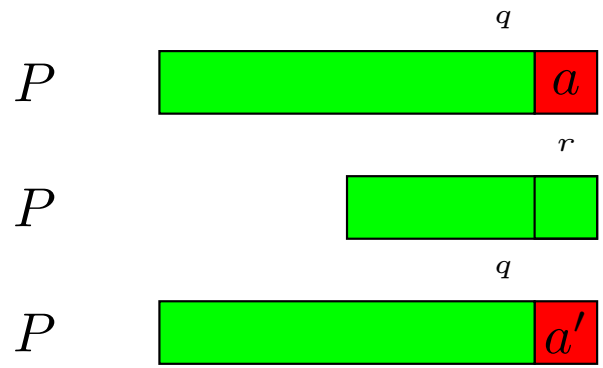
Soient deux transitions arrières $q \xrightarrow{a} r$ et $q' \xrightarrow{a'} r'$. Alors $q' - r' \neq q - r$.

Preuve. (sans perte de généralité $q \leq q'$)
Puisque $r > 0$, $P[1, r] = P[q - r + 2, q]a$ et par conséquent, $P[r] = a$.

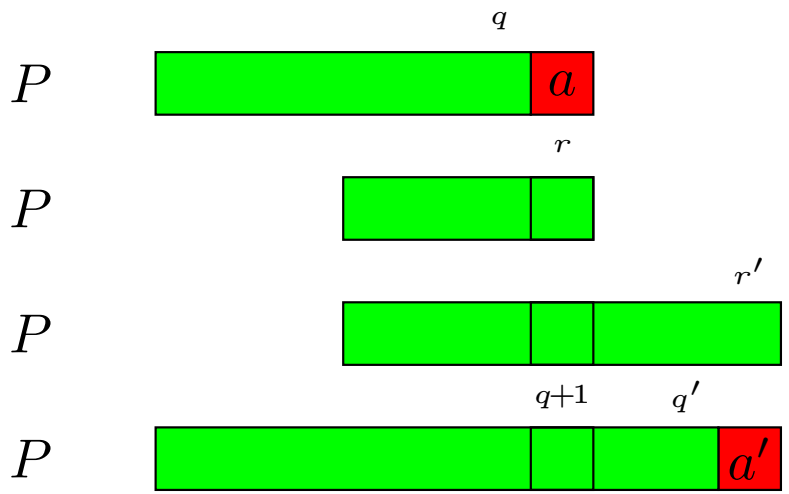
- Si $q = q'$ alors $a \neq a'$.
  Si $q' - r' = q - r$ alors $r = r'$.
  D'où $P[r] = a' \neq a = P[r]$, ce qui est contradictoire.
- Si $q < q' \leq m$, alors $P[q + 1] \neq a$.
  Puisque $r > 0$, $P[1, r'] = P[q' - r' + 2, q']a'$.
  Si $q' - r' = q - r$ alors $r' > r$.
  D'où $a = P[r] = P[r + (q' - r' + 1)] = P[q + 1] \neq a$, ce qui est contradictoire.

Illustration de la preuve

- $q = q'$



- $q < q'$

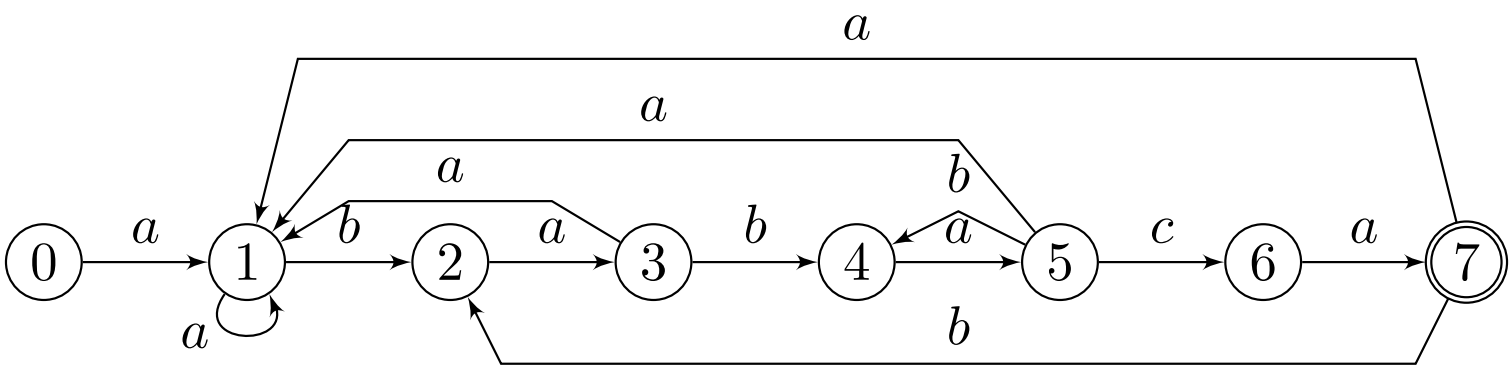


L'automate de Simon

L'automate de Simon est l'automate du motif privé des transitions conduisant à l'état 0.

L'automate de Simon a au plus $2m$ transitions.

Illustration $P = ababaca$



Calcul des transitions arrières

Soit $0 < i \leq m$. Si $\pi'(i) = -1$ alors il n'y a pas de transition arrière issue de i.

Supposons $\pi'(i) \geq 0$.

Soit $a \in \Sigma$ tel que ($i = m$ ou $a \neq P[i+1]$). Notons $j = \delta(i, a) = \delta(\pi'(i), a)$.

- soit $a = P[\pi'(i)+1]$ et $i \xrightarrow{a} \pi'(i)+1$ est une transition arrière ;
- soit $a \neq P[\pi'(i)+1]$ et $i \xrightarrow{a} j$ est une transition arrière ssi $\pi'(i) \xrightarrow{a} j$ est une transition arrière.

$Listarr[0] = \varepsilon$
For i **from** 1 **do** m **do**
  If $\pi'(i) = -1$ **then** $Listarr[i] \leftarrow \varepsilon$
  Else
    Copy$(Listarr[\pi'(i)], Listarr[i])$
    If $i < m$ **and** $\exists i' \xrightarrow{P[i+1]} j \in Listarr[i]$ **then**
      Extract$(i' \xrightarrow{P[i+1]} j, Listarr[i])$
    Insert$(i \xrightarrow{P[\pi'(i)+1]} \pi'(i)+1, Listarr[i])$

Ce calcul est en temps linéaire par rapport à la taille des listes donc en $\Theta(m)$.

Deux choix d'implémentation

Construction en $\Theta(m)$ de l'automate de Simon indépendamment du choix.

- Une « matrice » $\{0, \ldots, m\} \times \Sigma$ à valeurs dans $\{0, \ldots, m\}$ avec initialisation paresseuse à 0.
  - **Avantage.** Le traitement de chaque caractère se fait en temps constant.
  - **Inconvénient.** L'espace mémoire est en $\Theta(m|\Sigma|)$.

- Une liste des transitions (avant et arrière) par état triée par amplitude croissante.
  - **Avantage.** L'espace mémoire est en $\Theta(m)$
  - **Inconvénient.** Le traitement de chaque caractère ne se fait plus en temps constant ... mais est toujours au moins aussi rapide que par KMP !

Plan

Une autre version de l'algorithme naïf

$L \leftarrow \emptyset$;
For i **from** 0 **do** $n - m$ **do**
  $j \leftarrow m$; Comparaison de droite à gauche
  While $1 \leq j \leq m$ **do**
    If $T[i + j] = P[j]$ **then** $j \leftarrow j - 1$ **else** $j \leftarrow m + 1$;
  If $j = 0$ **then** $L \leftarrow L \cup \{i + 1\}$;

Observation.

Le caractère $T[i + m]$ doit s'apparier avec un caractère de $P[1, m - 1]$ lors des $m - 1$ comparaisons suivantes.

Notons $d_1(a) = m - \max(i \mid i < m \wedge P[i] = a)$ avec $\max(\emptyset) = 0$.

Alors les $d_1(T[i + m]) - 1$ comparaisons avec le motif seront infructeuses.

L'algorithme avec déplacement variable

$L \leftarrow \emptyset$;

$i \leftarrow 0$

While $i \leq n - m$ **do**

  $j \leftarrow m$;

  While $1 \leq j \leq m$ **do**

    If $T[i + j] = P[j]$ **then** $j \leftarrow j - 1$ **else** $j \leftarrow m + 1$;

  If $j = 0$ **then** $L \leftarrow L \cup \{i + 1\}$;

  $i \leftarrow i + d_1(T[i + m])$

Complexité au pire cas inchangée : $\Theta(nm)$

avec $T = a^n$ et $P = a^m$.

Complexité au meilleur cas sous-linéaire : $\Theta(\frac{n}{m})$

avec $T = (a^{m-1}b)^{\frac{n}{m}}$ et $P = a^m$.

Impossible à atteindre avec les approches par automates.

Complexité en moyenne (1)

Hypothèses.

On suppose que $m \leq |\Sigma|$.

Les caractères de T sont tirés uniformément et de manière indépendante dans Σ.

Il n'y a pas d'hypothèse sur les caractères de P.

Déplacement moyen.

Il y a plus m caractères qui apparaissent dans P.

D'où un déplacement moyen minoré par :

$$dmin = m\left(1 - \frac{m}{|\Sigma|}\right) + \frac{m}{|\Sigma|} = m - (m-1)\frac{m}{|\Sigma|}$$

Illustration. Si $2m \leq |\Sigma|$ alors :

$$dmin \geq \frac{m+1}{2}$$

Complexité en moyenne (2)

Nombre moyen de comparaisons lors d'un appariement possible

Hormis l'éventuel caractère du texte qui correspond au déplacement minimal,

la probabilité d'un ajustement entre un caractère du texte et du motif est $\frac{1}{|\Sigma|}$

On a donc affaire à un tirage répété jusqu'à échec

avec probabilité $1 - \frac{1}{|\Sigma|}$ d'échec.

D'où un nombre moyen (incluant le caractère qui s'ajuste) majoré par :

$$1 + \frac{|\Sigma|}{|\Sigma| - 1} \leq 3$$

(dès que $|\Sigma| \geq 2$)

Calcul de d_1

Recherche de la *dernière* occurence de tout a dans $P[1, m-1]$.

For $a \in \Sigma$ **do** $d_1(a) \leftarrow m$
For i **from** 1 **to** $m-1$ **do** $d_1(P[i]) \leftarrow m-i$

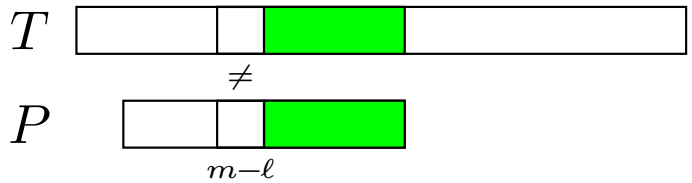
Complexité. $\Theta(m + |\Sigma|)$ ou même $\Theta(m)$ avec initialisation paresseuse.

Illustration. $P = abababaca$

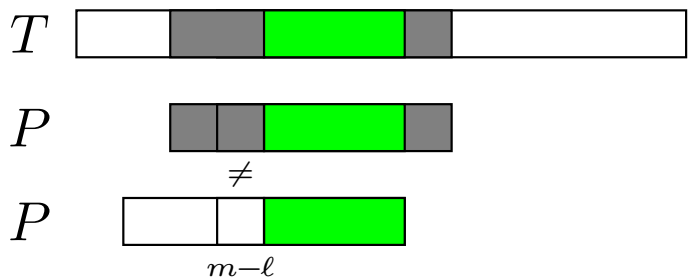
- $d_1(a) = 2$;
- $d_1(b) = 3$;
- $d_1(c) = 1$;
- Pour tout $x \in \Sigma \setminus \{a, b, c\}$, $d_1(x) = 7$.

Le déplacement d_2

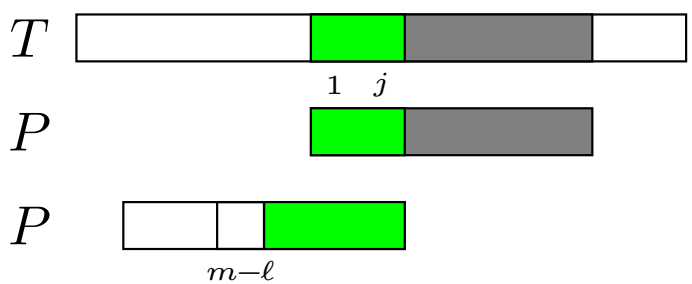
Soit ℓ, le nombre de caractères appariés lors de la dernière comparaison avec P.



Cas d'un petit déplacement avec appariement du motif :
$P[m-\ell+1, m]$ doit apparaître ailleurs dans le motif.



Cas d'un grand déplacement avec appariement du motif : $j < \ell$ et $j \in \pi^*(m)$.



Spécification de d_2

- La fonction $Suff(j)$ associe à une position j de P
  la plus grande longueur d'un suffixe de P se terminant en $j < m$:

$$Suff(j) = \max(k \mid 0 \leq k \leq j \wedge P[j-k+1, j] = P[m-k+1, m])$$

- d_2^a correspond au cas d'un déplacement sans troncature :

$$d_2^a(\ell) = m - \max(j < m \mid Suff(j) = \ell) \text{ avec } \max(\emptyset) = 0$$

- d_2^b correspond au cas d'un déplacement avec troncature :

$$d_2^b(\ell) = m - \max(\pi^{(k)}(m) \mid \pi^{(k)}(m) < \ell) \text{ avec } \max(\emptyset) = 0$$

$$\boxed{d_2(\ell) = \min(d_2^a(\ell), d_2^b(\ell))}$$

Illustration

$$P = abababa$$

j	0	1	2	3	4	5	6
$Suff(j)$	0	1	0	3	0	5	0

ℓ	0	1	2	3	4	5	6	7
$d_2^a(\ell)$	1	6	7	4	7	2	7	7

i	1	2	3	4	5	6	7
$\pi(i)$	0	0	1	2	3	4	5

ℓ	0	1	2	3	4	5	6	7
$d_2^b(\ell)$	7	7	6	6	4	4	2	2

ℓ	0	1	2	3	4	5	6	7
$d_2(\ell)$	1	6	6	4	4	2	2	2

L'algorithme avec déplacement variable

$L \leftarrow \emptyset$

$i \leftarrow 0$

While $i \leq n - m$ **do**

  $j \leftarrow m$

  $\ell \leftarrow 0$

  While $1 \leq j \leq m$ **do**

    If $T[i + j] = P[j]$ **then**

      $\ell \leftarrow \ell + 1$

      $j \leftarrow j - 1$

    Else

      $j \leftarrow m + 1$

  If $j = 0$ **then** $L \leftarrow L \cup \{i + 1\}$

  $i \leftarrow i + \max(d_1(T[i + m]), d_2(\ell))$

Calcul de d_2^a et d_2^b

Complexité en $\Theta(m)$ une fois $Suff$ calculée.

For i **from** 1 **to** m **do**
  $d_2^a(i) \leftarrow m$
For i **from** 1 **to** $m-1$ **do**
  If $Suff(i) \neq 0$ **then** $d_2^a(Suff(i)) \leftarrow m - i$

Complexité en $\Theta(m)$.

$j \leftarrow m$
While $j > 0$
  For i **from** $\pi(j)+1$ **to** j **do** $d_2^b(i) \leftarrow m - \pi(j)$
  $j \leftarrow \pi(j)$
$d_2^b(0) \leftarrow m$

Calcul de $Suff$ via $Pref$

- La fonction $Pref(j)$ associe à une position j de P
  la plus grande longueur d'un préfixe de P commençant en j :

$$Pref(j) = \max(k \mid 0 \leq k \leq m - j + 1 \wedge P[j, j + k - 1] = P[1, k])$$

- Calculer $Suff$ revient à calculer $Pref$ pour $\tilde{P}$, le miroir de P.
- Un algorithme naïf opère en $\Theta(m^2)$. Comment obtenir une complexité en $\Theta(m)$?
- La fonction auxiliaire **Scan** renvoie la taille
  du plus grand sous-mot commun démarrant en x et y avec $x > y$.

Scan(x, y) : une longueur
$z \leftarrow 0$
While $x \leq m$ **do**
  If $P[x] = P[y]$ **then** $x \leftarrow x + 1$; $y \leftarrow y + 1$; $z \leftarrow z + 1$
  Else $x \leftarrow m + 1$
Return(z)

On dit que l'indice x est *touché* lorsque la comparaison $P[x] = P[y]$ renvoie vrai.

Un algorithme efficace

L'algorithme calcule $Pref(i)$ par valeurs de i croissantes.

Il maintient s qui maximise $i + Pref(i)$ pour les i déjà évalués tels que $Pref(i) > 0$.

Tant qu'il n'existe pas un tel i, $s = 1$ avec $Pref(1) = 0$ par convention.

$Pref(1) \leftarrow 0\,;\ s \leftarrow 1$

For i **from** 2 **to** m **do**

  $k \leftarrow i - s + 1\,;\ r \leftarrow s + Pref(s)$

  If $r \leq i$ **then** cas 1

    $Pref(i) \leftarrow \mathbf{Scan}(i, 1)$

    If $Pref(i) > 0$ **then** $s \leftarrow i$

  Else If $Pref(k) + k - 1 \neq Pref(s)$ **then** cas 2 et 3

    $Pref(i) \leftarrow \min(Pref(k), r - i)$

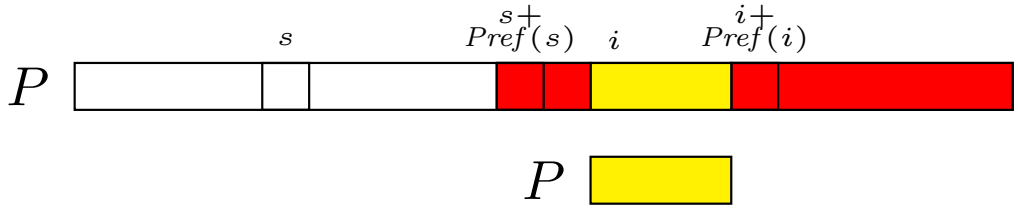
  Else cas 4

    $t \leftarrow \mathbf{Scan}(r, r - i + 1)$

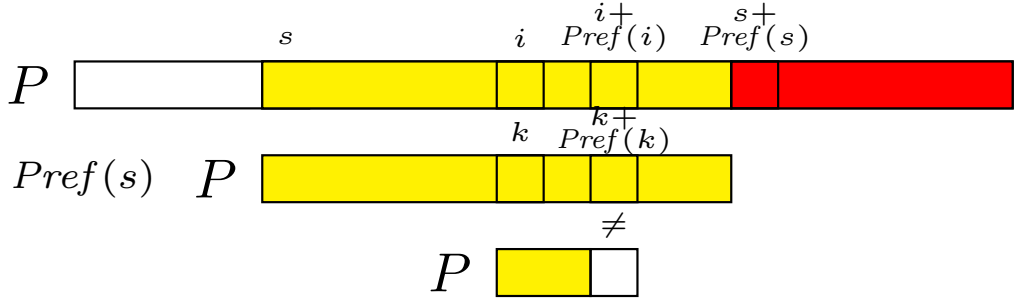
    $Pref(i) \leftarrow r - i + t\,;\ s \leftarrow i$

Illustration des quatre cas

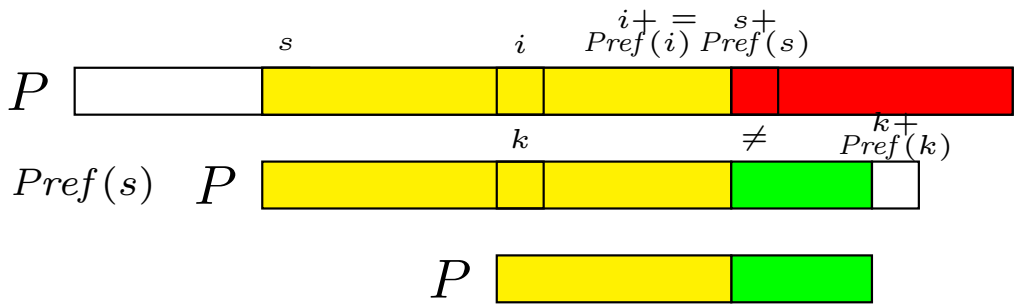
$s + Pref(s) \leq i$



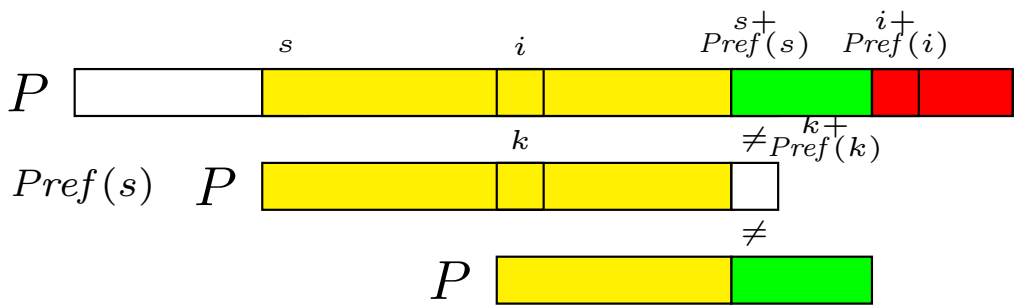
$s + Pref(s) > i$
$k + Pref(k) - 1 < Pref(s)$



$s + Pref(s) > i$
$k + Pref(k) - 1 > Pref(s)$



$s + Pref(s) > i$
$k + Pref(k) - 1 = Pref(s)$



Invariant de boucle

Invariant. Les indices inférieurs à $s + Pref(s)$ ont été touchés au plus une fois et les indices supérieurs ou égaux n'ont pas été touchés.

Preuve. Examinons les quatre cas de l'algorithme lorsqu'il évalue $Pref(i)$.

1. $s + Pref(s) \leq i$. Les caractères en jaune sont les caractères touchés.
   Ils n'étaient pas touchés auparavant.
   Si $Pref(i) > 0$ alors s devient i. Donc l'invariant de boucle reste satisfait.
2. $s + Pref(s) > i$ et $k - 1 + Pref(k) < Pref(s)$. Aucune comparaison n'est nécessaire car le préfixe associé à k est celui associé à i.
   s est inchangé et l'invariant de boucle reste satisfait.
3. $s + Pref(s) > i$ et $k - 1 + Pref(k) > Pref(s)$. Aucune comparaison n'est nécessaire car le préfixe associé à i est celui associé à k tronqué aux $s + Pref(s) - i - 1$ premiers caractères.
   s est inchangé et l'invariant de boucle reste satisfait.
4. $s + Pref(s) > i$ et $k - 1 + Pref(k) = Pref(s)$. Les caractères en jaune sont les caractères touchés et ils n'étaient pas touchés auparavant.
   s devient i. L'invariant de boucle reste satisfait.

Analyse de complexité

Une itération s'exécute en $\Theta(1)$ excepté l'appel éventuel à **Scan**.

Il y au plus une comparaison négative par appel

donc au plus m comparaisons négatives.

Le temps du reste de l'appel est proportionnel au nombre de comparaisons positives i.e. au nombre d'indices touchés.

En cumulant sur tous les appels et en prenant en compte le fait qu'un indice est touché au plus une fois, on obtient un temps en $\Theta(m)$.

Plan

Principe de la recherche

Soit P une expression régulière de taille m et T un texte de taille n.

Construction d'un ε-automate *normalisé* reconnaissant $\Sigma^* \cdot P$ en $\Theta(m)$

Maintien du sous-ensemble d'états atteints par $T[1, i]$ pour $i \leq n$ en $\Theta(m)$

Affichage des i tels que l'état final appartienne au sous-ensemble d'états.

D'où une complexité en $\Theta(mn)$

ε-Automate normalisé

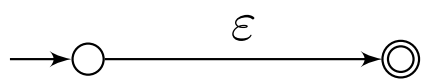
- Un unique état initial et un unique état final distincts.
- Pas de transition entrante dans l'état initial.
- Pas de transition sortante de l'état final.
- L'ensemble des transitions sortantes de tout état est
  - soit un singleton étiqueté par une lettre de Σ ;
  - soit un ensemble d'au plus deux transitions étiquetées par ε.

Pour tout expression de taille m, on peut construire en $O(m)$ un ε-automate normalisé avec au plus $2m$ états (et donc au plus $4m$ transitions).

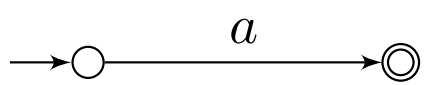
Construction inductive (1)

- $\mathcal{A}(\emptyset)$

- $\mathcal{A}(\varepsilon)$

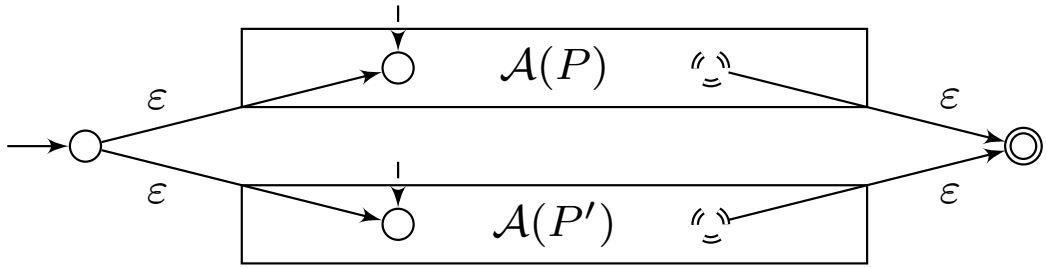


- $\mathcal{A}(a)$ pour $a \in \Sigma$

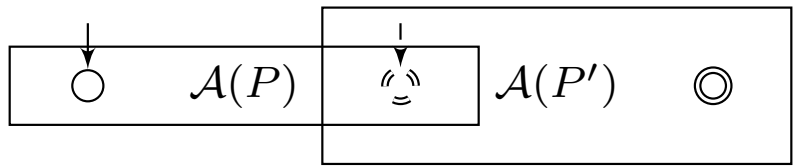


Construction inductive (2)

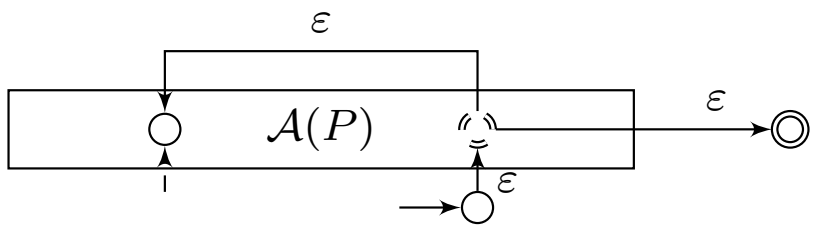
- $\mathcal{A}(P + P')$



- $\mathcal{A}(P \cdot P')$



- $\mathcal{A}(P^*)$

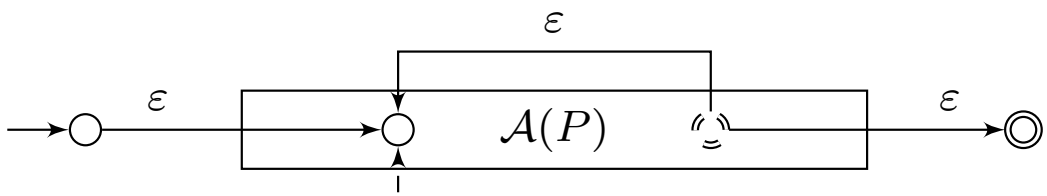


Construction inductive (3)

Du sucre syntaxique ...

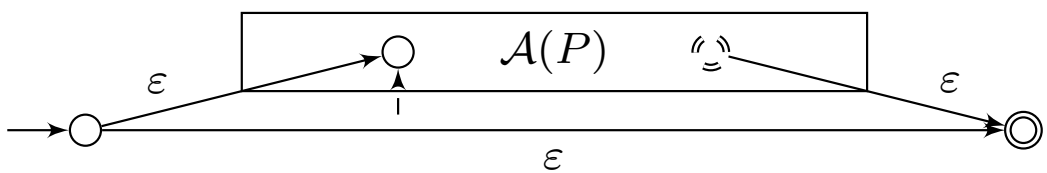
- $P^! = P.P^*$, i.e. au moins une fois P

$\mathcal{A}(P^!)$



- $P^? = P + \varepsilon$, i.e. au plus une fois P

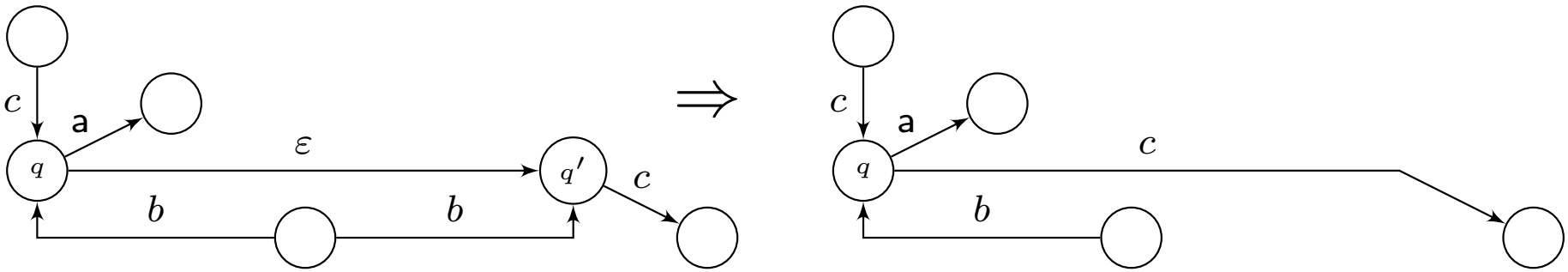
$\mathcal{A}(P^?)$



- Soit $i \neq j \in \{?, !, *\}$ alors $(P^i)^i = P^i$ et $(P^i)^j = P^*$.

Elimination des ε-transitions (1)

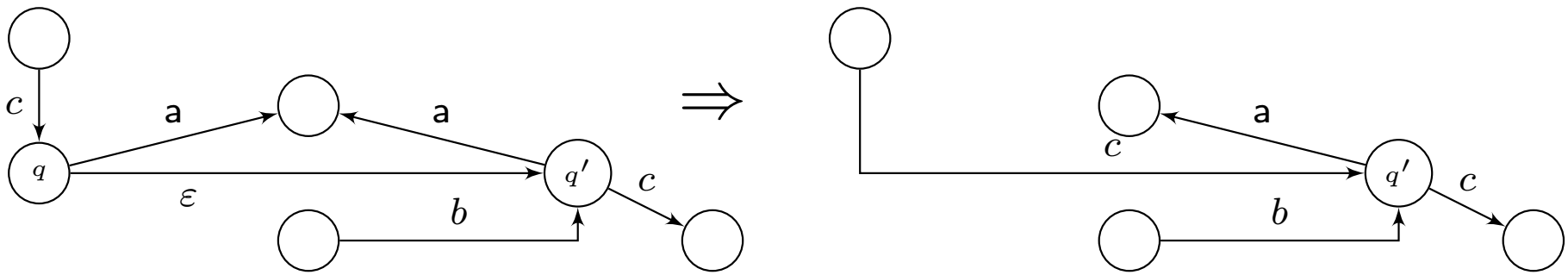
- Pré-inclusion :
  - $\exists q \xrightarrow{\varepsilon} q' \wedge q' \neq initial$
  - $\forall q'' \neq q, x \; q'' \xrightarrow{x} q' \Rightarrow q'' \xrightarrow{x} q$



- ε-Pré-inclusion :
  - $\exists q \xrightarrow{\varepsilon} q' \wedge q' \neq initial$
  - $\forall q'' \neq q, x \neq \varepsilon \; q'' \xrightarrow{x} q' \Rightarrow q'' \xrightarrow{x} q$
  - $\forall q'' \neq q \; q'' \xrightarrow{\varepsilon} q' \Rightarrow q'' \xrightarrow{\varepsilon}_* q$

Elimination des ε-transitions (2)

- Post-inclusion :
  - $\exists q \xrightarrow{\varepsilon} q' \wedge q \neq final$
  - $\forall q'' \neq q', x \; q \xrightarrow{x} q'' \Rightarrow q' \xrightarrow{x} q''$



- ε-Post-inclusion :
  - $\exists q \xrightarrow{\varepsilon} q' \wedge q \neq final$
  - $\forall q'' \neq q', x \neq \varepsilon \; q \xrightarrow{x} q'' \Rightarrow q' \xrightarrow{x} q''$
  - $\forall q'' \neq q' q \xrightarrow{\varepsilon} q'' \Rightarrow q' \xrightarrow{\varepsilon}_* q''$

Autres réductions

- Suppression d'arc :
  - $\exists q \xrightarrow{a} q'$
  - $\exists q \xrightarrow{\varepsilon}_* q_1 \xrightarrow{a} q'_1 \xrightarrow{\varepsilon}_* q'$
  - $q \xrightarrow{a} q' \neq q_1 \xrightarrow{a} q'_1$
    (couvre le cas de deux transitions identiques)

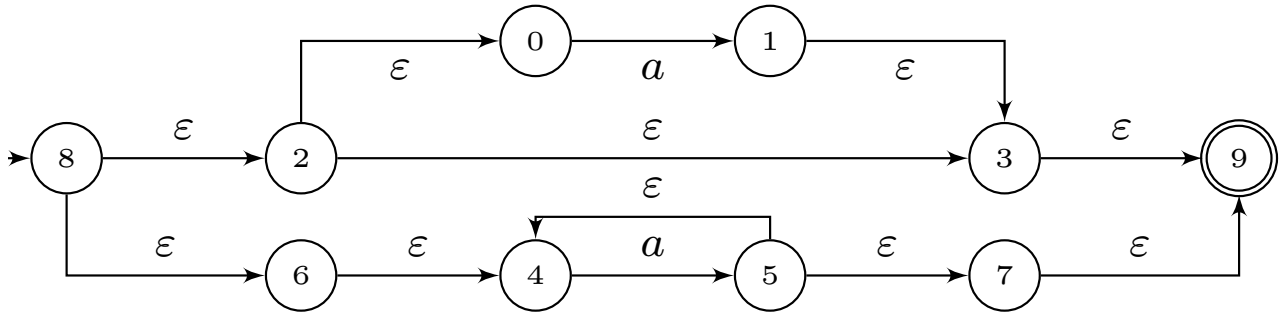
Alors $q \xrightarrow{a} q'$ peut être supprimé.

- Fusion d'états équivalents :
  - Soit $Suiv(q) = \{(\alpha, q') \mid q \xrightarrow{\alpha} q'\}$
  - $\exists q \neq q'\ Suiv(q) = Suiv(q')$

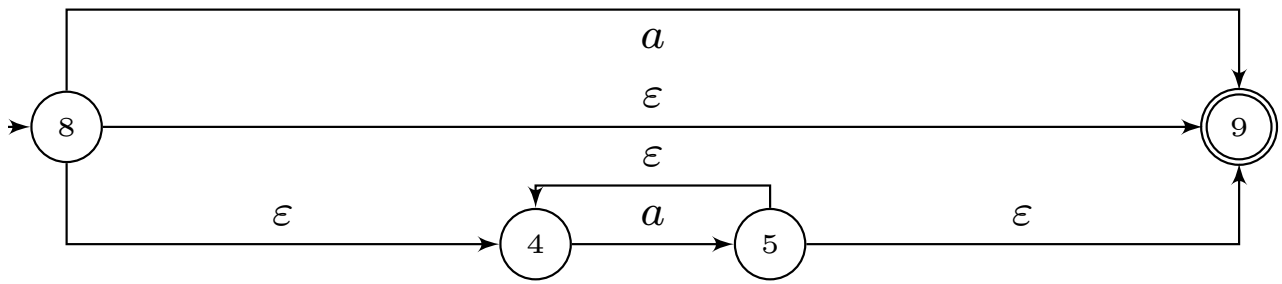
Alors q peut être supprimé et ses arcs entrants redirigés vers q'.

Illustration

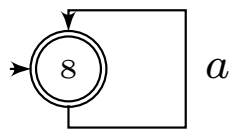
Soit le motif $a? + a!$ et son automate :



Après les pré-inclusions $(2, 0)$, $(5, 7)$, $(8, 2)$, $(8, 6)$
et les post-inclusions $(1, 3)$ et $(3, 9)$:



Après la suppression d'arc $8 \xrightarrow{a} 9$, l'équivalence d'états $(8, 5)$
et les pré-inclusions $(8, 9)$ et $(8, 4)$:



Pas de garantie de minimalité (le problème de l'universalité est PSPACE-complet).

Reconnaissance du motif

- Le tableau $Current$ décrit le sous-ensemble d'états atteint par $T[1, i]$
- Il est initialisé à q_0 et enrichi par les états atteints via des ε-transitions depuis q_0.
- A l'étape i, on le construit à partir des transitions étiquetées par $T[i]$

  puis enrichi par les états atteints via des ε-transitions.

For $q \in Q$ **do** $Current[q] \leftarrow \bot$
$Current[q_0] \leftarrow \top$; Complete$(Current)$; **If** $Current[q_f]$ **then** Print(0)
For i **from** 1 **do** n **do**
  For $q \in Q$ **do** $Prev[q] \leftarrow Current[q]$; $Current[q] \leftarrow \bot$
  For $q \in Q$ **do**
    If $Prev[q]$ **then For all** $q \xrightarrow{T[i]} q'$ **do** $Current[q'] \leftarrow \top$
  Complete$(Current)$; **If** $Current[q_f]$ **then** Print(i)

D'après les hypothèses sur les transitions sortantes par état,

le parcours des transitions issues de q ($q \xrightarrow{T[i]} q'$) se fait en temps constant.

ε-Complétion

Cette étape consiste en un algorithme d'accessibilité en temps linéaire par rapport au nombre d'états et d'ε-transitions soit en $\Theta(m)$.

Ici on a choisi de l'implémenter par des parcours en profondeur, d'où le choix d'une pile.

Complete($Current$)
  $Stack \leftarrow \emptyset$
  For $q \in Q$ **do If** $Current[q]$ **then** Push($Stack, q$)
  While $Stack \neq \emptyset$ **do**
    $q \leftarrow$ Pop($Stack$)
    For all $q \xrightarrow{\varepsilon} q'$ **do**
      If not $Current[q']$ **then** $Current[q'] \leftarrow \top$; Push($Stack, q'$)

Affichage des fragments appariés

Observations.

- Afficher les positions de fin n'est pas assez informatif ;
- Afficher tous les fragments appariés est trop couteux ;
- avec $P = a^*$ et $T = a^n$ tout fragment convient.

Objectif.

- Afficher un fragment par position de fin
- mais lequel ?

Le fragment le plus court est généralement le plus interprétable.

... mais le fragment le plus long a aussi des applications.

Extension des structures de données

Le tableau $Current$ est maintenant un tableau d'entiers.

Après avoir lu $T[1, i]$,

- $Current[q] = 0$ s'il n'existe pas de $j \leq i + 1$
  et un calcul de l'automate du motif $q_0 \xrightarrow{T[j,i]} q$;
- Sinon $Current[q]$ est le plus grand j associé à un tel calcul.

Pour des raisons d'efficacité, on maintient une pile/file $Stcurrent$
qui contient les paires $(q, Current[q])$ tels que $Current[q] > 0$.
Les opérations sont Push (pile), Insert (file) et Pop (pile/file).

Les valeurs de $Current[q]$ décroissent
en descendant du sommet à la base de la pile.

L'algorithme revisité

For $q \in Q$ **do** $Current[q] \leftarrow 0$

$Current[q_0] \leftarrow 1$; Push$(Stcurrent, (q_0, 1))$

Complete$(Current, Stcurrent)$

If $Current[q_f] > 0$ **then** Print$((1, 0))$

For i **from** 1 **do** n **do**

  $Stprev \leftarrow Stcurrent$; $Stcurrent \leftarrow \emptyset$

  For $q \in Q$ **do** $Current[q] \leftarrow 0$

  While $Stprev \neq \emptyset$

    $(q, ind) \leftarrow$ Pop$(Stprev)$

    For all $q \xrightarrow{T[i]} q'$

      If $Current[q'] = 0$ **then** $Current[q'] \leftarrow ind$; Insert$(Stcurrent, (q', ind))$

  Push$(Stcurrent, (q_0, i + 1))$

  Complete$(Current, Stcurrent)$

  If $Current[q_f] > 0$ **then** Print$((Current[q_f], i))$

ε-Complétion revisitée

Complete$(Current, Stcurrent)$

$Stprev \leftarrow Stcurrent$; $Stcurrent \leftarrow \emptyset$

For $q \in Q$ **do** $Current[q] \leftarrow 0$

While $Stprev \neq \emptyset$ **do**

$(q, ind) \leftarrow$ Pop$(Stprev)$

If $Current[q] < ind$ **then**

$Current[q] \leftarrow ind$; Insert$(Stcurrent, (q, ind))$

For all $q \xrightarrow{\varepsilon} q'$ **do** Push$(Stprev, (q', ind))$

Cet algorithme opère en $\Theta(m)$

car le test $Current[q] < ind$ ne sera vrai qu'une fois par état q.

Plan

Distance d'édition

Opérations standard.

- substitution d'un caractère par un autre
- suppression d'un caractère
- insertion d'un caractère

Illustration.

$$
\begin{aligned}
COMPUTER &\rightarrow COMPUTEUR \\
&\rightarrow OMPUTEUR \\
&\rightarrow ORPUTEUR \\
&\rightarrow ORDUTEUR \\
&\rightarrow ORDITEUR \\
&\rightarrow ORDINTEUR \\
&\rightarrow ORDINATEUR
\end{aligned}
$$

Distance d'édition

Soit x et y deux mots. Alors $edit(x, y)$ la *distance d'édition* est définie par : le nombre minimal d'opérations à effectuer sur x pour obtenir y.

$$\boxed{edit(x, y) \text{ est une distance.}}$$

- Pour tout x, y, $edit(x, y) = 0$ ssi $x = y$;
- Pour tout x, y, $edit(x, y) = edit(y, x)$;
- Pour tout x, y, z, $edit(x, z) \leq edit(x, y) + edit(y, z)$.

Observation. Puisque les opérations sont réversibles, $edit(x, y)$ est aussi le nombre minimal d'opérations à effectuer sur x et y pour obtenir un mot identique.

Illustration.

$$edit(COMPUTER, ORDINATEUR) = 7$$

Propriété de la distance d'édition

Soit u un mot de longueur $m > 0$ et v un mot de longueur $n > 0$. Alors :

$$edit(u,v) = \min(edit(u[1,m-1],v)+1, edit(u,v[1,n-1])+1, edit(u[1,m-1],v[1,n-1]) + \mathbf{1}_{u[m]\neq v[n]})$$

Preuve.

L'inégalité $\leq$. Pour apparier u et v, on peut :

- apparier $u[1,m-1]$ et v puis supprimer $u[m]$;
- apparier u et $v[1,n-1]$ puis créer $v[n]$;
- apparier $u[1,m-1]$ et $v[1,n-1]$
  puis éventuellement substituer $u[m]$ par $v[n]$.

L'inégalité $\geq$. Soit ρ une suite minimale d'opérations qui conduit de u à v.

(selon le devenir de $u[m]$ après ρ)

- Si $u[m]$ a été supprimé alors ρ sans la suppression apparie $u[1,m-1]$ à v ;
- Si $u[m]$ n'est plus le dernier caractère alors un nouveau caractère s'apparie avec $v[n]$ et ρ sans cette création apparie u à $v[1,n-1]$;
- Sinon $u[m]$ s'apparie avec $v[n]$ avec substitution si $u[m] \neq v[n]$
  alors ρ sans l'éventuelle substitution apparie $u[1,m-1]$ à $v[1,n-1]$.

Calcul de la distance d'édition

Programmation dynamique.

Les sous-problèmes sont $\{edit(u[1,i], v[1,j])\}_{i \leq m, j \leq n}$.

Pour tout i, j, $edit(u[1,i], v[1,0]) = i$ et $edit(u[1,0], v[1,j]) = j$.

Pour tout $0 < i, j$, on applique la relation établie précédemment.

On stocke les résultats dans le tableau d (une optimisation mémoire est possible).

For i **from** 0 **to** m **do** $d[i,0] \leftarrow i$

For j **from** 1 **to** n **do** $d[0,j] \leftarrow j$

For i **from** 1 **to** m **do**

  For j **from** 1 **to** n **do**

    If $u[i] = v[j]$ **then** $d[i,j] \leftarrow d[i-1,j-1]$ **else** $d[i,j] \leftarrow d[i-1,j-1] + 1$

    If $d[i,j] > d[i,j-1] + 1$ **then** $d[i,j] \leftarrow d[i,j-1] + 1$

    If $d[i,j] > d[i-1,j] + 1$ **then** $d[i,j] \leftarrow d[i-1,j] + 1$

Calcul de la suite d'opérations (1)

On part de $d[i, j]$ avec $i = m$ et $j = n$ on déduit de $d[m-1, n]$, $d[n-1, m]$

et $d[m-1, n-1]$ quelle est l'éventuelle dernière opération à effectuer.

On met à jour i et j en fonction du sous-problème choisi.

On itère le procédé jusqu'à ce que $i = 0$ ou $j = 0$.

On débute alors par des suppressions ou des insertions.

Calcul de la suite d'opérations (2)

$i \leftarrow m\,;\ j \leftarrow n\,;\ L \leftarrow \varepsilon\,;$

While $i > 0$ **and** $j > 0$ **do**

$\{v[1,j]u[i+1,m] \xrightarrow{L} v\}$

If $d[i,j] = d[i-1,j] + 1$ **then** $L \leftarrow del(j+1) \cdot L\,;\ i \leftarrow i-1$

$v[1,j]u[i,m] \xrightarrow{del(j+1)} v[1,j]u[i+1,m] \xrightarrow{L} v$

Else If $d[i,j] = d[i,j-1] + 1$ **then** $L \leftarrow ins(j, v[j]) \cdot L\,;\ j \leftarrow j-1$

$v[1,j-1]u[i+1,m] \xrightarrow{ins(j,v[j])} v[1,j]u[i+1,m] \xrightarrow{L} v$

Else

If $u[i] \neq v[j]$ **then** $L \leftarrow upd(j, v[j]) \cdot L$

$i \leftarrow i-1\,;\ j \leftarrow j-1$

$v[1,j-1]u[i,m] \xrightarrow{upd(j,v[j])^?} v[1,j]u[i+1,m] \xrightarrow{L} v$

If $i = 0$ **then for** k **from** j **downto** 1 **do** $L \leftarrow ins(k, v[k]) \cdot L$

$u \xrightarrow{ins(1,v[1]) \cdots ins(j,v[j])} v[1,j]u \xrightarrow{L} v$

Else if $j = 0$ **then for** k **from** 1 **to** i **do** $L \leftarrow del(1) \cdot L$

$u \xrightarrow{del(1)^i} u[i+1,m] \xrightarrow{L} v$

Recherche approchée de motif

On cherche un facteur du texte qui minimise la distance d'édition avec le motif.

On définit $se(i,j) = \min(edit(T[k,i], P[1,j]) \mid 1 \leq k \leq i+1)$.

$$se(i,j) = \min(se(i-1,j)+1, se(i,j-1)+1, se(i-1,j-1)+\mathbf{1}_{T[i]\neq P[j]})$$

Preuve.

- Soit $T[k,i]$ le suffixe qui minimise la distance à $P[1,j-1]$ par création de $P[j]$, on obtient $se(i,j) \leq se(i,j-1)+1$;
- Soit $T[k',i-1]$ le suffixe qui minimise la distance à $P[1,j]$ par suppression de $T[i]$, on obtient $se(i,j) \leq se(i-1,j)+1$;
- Soit $T[k'',i-1]$ le suffixe qui minimise la distance à $P[1,j-1]$ par substitution éventuelle de $T[i]$ par $P[j]$, on obtient : $se(i,j) \leq se(i-1,j-1)+\mathbf{1}_{T[i]\neq P[j]}$.

Soit ρ une suite d'opérations qui conduit de $T[k,i]$ à $P[1,j]$.
(selon le devenir de $T[i]$ après ρ)

- Si $T[i]$ a été supprimé alors ρ sans la suppression apparie $T[k,i-1]$ à $P[1,j]$;
- Si Si $T[i]$ n'est plus le dernier caractère alors un nouveau caractère s'apparie avec $P[j]$ alors ρ sans cette création apparie $T[k,i]$ à $P[1,j-1]$;
- Sinon $T[i]$ s'apparie avec $P[j]$ avec substitution si $T[i] \neq P[j]$ et ρ sans l'éventuelle substitution apparie $T[k,i-1]$ à $P[1,j-1]$.

Calcul de se

Pour tout i, j, $se(i, 0) = 0$ et $se(0, j) = j$.

Pour tout $0 < i, j$, on applique la relation établie précédemment.

opt est la première position optimale.

For i **from** 1 **to** n **do** $se[i, 0] \leftarrow 0$
For j **from** 1 **to** m **do** $se[0, j] \leftarrow j$
For i **from** 1 **to** n **do**
  For j **from** 1 **to** m **do**
    If $T[i] = P[j]$ **then** $se[i, j] \leftarrow se[i-1, j-1]$ **else** $se[i, j] \leftarrow se[i-1, j-1] + 1$
    If $se[i, j] > se[i, j-1] + 1$ **then** $se[i, j] \leftarrow se[i, j-1] + 1$
    If $se[i, j] > se[i-1, j] + 1$ **then** $se[i, j] \leftarrow se[i-1, j] + 1$
$opt \leftarrow 0$; $vopt \leftarrow m$
For i **from** 1 **to** n **do** **if** $se[i, m] < vopt$ **then** $opt \leftarrow i$; $vopt \leftarrow se[i, m]$

Observation. On peut mémoriser sans surcoût toutes les positions optimales.

Calcul d'un facteur optimal

Principe.

- On calcule la suite d'opérations comme pour la distance d'édition,
- excepté la suppression des caractères de $T[1, i]$.
- $T[i+1, opt]$ est un facteur optimal.

$i \leftarrow opt$; $j \leftarrow m$; $L \leftarrow \varepsilon$;
While $i > 0$ **and** $j > 0$ **do**
  If $se[i, j] = se[i-1, j] + 1$ **then** $L \leftarrow ins(j, P[j]) \cdot L$; $i \leftarrow i - 1$
  Else If $se[i, j] = se[i, j-1] + 1$ **then** $L \leftarrow del(j+1) \cdot L$; $j \leftarrow j - 1$
  Else
    If $T[i] \neq P[j]$ **then** $L \leftarrow upd(j, P[j]) \cdot L$
    $i \leftarrow i - 1$; $j \leftarrow j - 1$
If $i = 0$ **then for** k **from** j **downto** 1 **do** $L \leftarrow ins(k, P[k]) \cdot L$
$res \leftarrow T[i+1, opt]$

Distance d'édition généralisée

On associe un coût à chaque opération

- i pour une insertion ;
- d pour une suppression ;
- r pour une substitution.

Par un raisonnement similaire, on obtient :

$$edit(u,v) = \min(edit(u[1,m-1],v)+d, edit(u,v[1,n-1])+i, edit(u[1,m-1],v[1,n-1])+r\mathbf{1}_{u[m]\neq v[n]})$$

Cas particulier. $edit_{di}$: $i = d = 1$ et $r \geq 2$ *(valable aussi pour $i + d \leq r$)*.

Ici, la substitution peut-être remplacée par une suppression et une création. D'où une nouvelle formule :

$$edit_{di}(u,v) = \min(edit_{di}(u[1,m-1],v)+1, edit_{di}(u,v[1,n-1])+1, edit_{di}(u[1,m-1],v[1,n-1])+\infty\mathbf{1}_{u[m]\neq v[n]})$$

Une autre équation pour $edit_{di}$ (1)

Si $u[m] = v[n]$ alors $edit_{di}(u, v) = edit_{di}(u[1, m-1], v[1, n-1])$
Sinon $edit_{di}(u, v) = 1 + \min(edit_{di}(u[1, m-1], v), edit_{di}(u, v[1, n-1]))$.

Preuve.

- Supposons $u[m] \neq v[n]$. Le résultat découle de l'équation précédente pour $edit_{di}$.
- Supposons $u[m] = v[n]$. Soit une suite d'opérations ρ qui conduit de u à v.

Cas 1. Dans ρ, le caractère qui s'apparie à $v[n]$ est un nouveau caractère et $u[m]$ n'a pas été supprimé.

$$u[1, m-1]u[m] \rightarrow^* v_1 u[m] v_2 \rightarrow^* v_1 u[m] v_2 u[m]$$

Soit ρ' définie ainsi :

$$u[1, m-1]u[m] \rightarrow^* v_1 v_2 u[m] \rightarrow^* v_1 u[m] v_2 u[m]$$

conduit de u à v et $|\rho'| = |\rho|$.

Une autre équation pour $edit_{di}$ (2)

Preuve (suite).

Cas 2. Dans ρ, le caractère qui s'apparie à $v[n]$ est un nouveau caractère et $u[m]$ a été supprimé.

$$u \to u[1, m-1] \to^* v[1, n-1] \to^* v[1, n-1]u[m]$$

Alors ρ' définie ainsi :

$$u \to^* v[1, n-1]u[m]$$

conduit de u à v et $|\rho'| = |\rho| - 2$.

Cas 3. Dans ρ', le caractère qui s'apparie à $v[n]$ est un caractère $u[i]$. Donc $u[m]$ a été supprimé.

$$u \to u[1, i-1]u[i] \to^* v[1, n-1]u[i]$$

Alors ρ' définie ainsi :

$$u \to u[1, i-1]u[m] \to^* v[1, n-1]u[m]$$

conduit de u à v et $|\rho'| = |\rho|$.

Texte et motif avec caractère joker

Soit $\sharp$ un caractère joker. Soit $\Sigma_\sharp = \Sigma \uplus \{\sharp\}$.

On définit l'appariement de caractères $a \approx b$ par $\sharp \in \{a, b\} \vee a = b$.

Soit $P \in \Sigma_\sharp^m$ et $T \in \Sigma_\sharp^n$.

On cherche toutes les positions du texte où le motif et le texte s'apparient.

Tout algorithme d'appariement par des comparaisons d'un caractère du texte et d'un caractère du motif nécessite $\Omega(n^2)$ comparaisons.

Preuve par l'absurde.

Soit n impair, $P = \sharp^m$ et $T = \sharp^n$ avec $m = \frac{n+1}{2}$.

Supposons que l'algorithme effectue moins de m^2 comparaisons.

Alors l'une des m^2 comparaisons '$T[i + j - 1] \approx P[j]$?'
pour $1 \leq i \leq m$ et $1 \leq j \leq m$ n'est pas effectuée.

Soit $a \neq b \in \Sigma$. En remplaçant $T[i + j - 1]$ par a et $P[j]$ par b,

l'algorithme fournit le même résultat bien que $T[i, i + m - 1] \not\approx P$.

Produit logique

Soit v, w des vecteurs de n booléens indicés de 0 à $n-1$.

Le produit logique $z = v \bullet w$ de $2n-1$ booléens est défini par :

$$z[k] = \bigvee_{0 \leq i,j < n, i+j=k} v[i] \wedge w[j]$$

Le produit logique via le produit de polynômes.

Soit P, Q polynômes de degré inférieurs à n avec $P[i]$, $Q[i]$, coefficients de X^i.

$P \leftarrow 0$; $Q \leftarrow 0$
For i **from** 0 **to** $n-1$ **do** $P[i] \leftarrow \mathbf{1}_{v[i]=\top}$; $Q[i] \leftarrow \mathbf{1}_{w[i]=\top}$
$R \leftarrow P \cdot Q$
For i **from** 0 **to** $2n-2$ **do if** $R[i] > 0$ **then** $z[i] \leftarrow \top$ **else** $z[i] \leftarrow \bot$

Le produit efficace de polynômes entiers s'effectue en $O(n \log(n) \log(\log(n)))$.

Par conséquent le produit logique s'effectue en $O(n \log(n) \log(\log(n)))$.

Illustration

Soit

$$v = (\bot, \top, \top) \text{ et } w = (\top, \top, \top)$$

Alors :

$$P = X + X^2 \text{ et } Q = 1 + X + X^2$$

D'où :

$$P \cdot Q = X + 2X^2 + 2X^3 + X^4$$

Et finalement :

$$v \bullet w = (\bot, \top, \top, \top, \top)$$

Recherche du motif par le produit logique

On considère P et T indicés à partir de 0.

On note $\tilde{P}$ le mot miroir de P.

Pour $m-1 \leq k < n$, soit $res[k]$ défini par $res[k] = \top$ si $T[k-m+1,k] \approx P$.

$$res[k] = \bigwedge_{0\leq i<m, i+j=k} \tilde{P}[i] \approx T[j] = \neg \bigvee_{0\leq i<m, i+j=k} \tilde{P}[i] \not\approx T[j]$$

$$= \neg \bigvee_{0\leq i<m, i+j=k} \bigvee_{a\neq b\in\Sigma} (\tilde{P}[i] = a \wedge T[j] = b) = \neg \bigvee_{a\neq b\in\Sigma} \bigvee_{0\leq i<m, i+j=k} (\tilde{P}[i] = a \wedge T[j] = b)$$

Un autre algorithme

Soit u un mot. Alors le vecteur logique $\top_{u,a}$ est défini par $\top_{u,a}[i] = \top$ si $u[i] = a$.

D'où pour $m-1 \leq k < n$, $res[k] = \left(\neg \bigvee_{a\neq b\in\Sigma} \top_{\tilde{P},a} \bullet \top_{T,b}\right)[k]$.

avec une complexité en $O(n \log(n) \log(\log(n))|\Sigma|^2)$ (versus $\Omega(n^2 \log(|\Sigma|))$.

Plus longue sous-séquence commune

Soit u un mot de longueur m et v un mot de longueur n.

w un mot de longueur p est une sous-séquence commune de u et v si :

- Il existe α une fonction strictement croissante de $\{1, \ldots, p\}$ dans $\{1, \ldots, m\}$ telle que pour tout i, $w[i] = u[\alpha(i)]$;
- Il existe β une fonction strictement croissante de $\{1, \ldots, p\}$ dans $\{1, \ldots, n\}$ telle que pour tout i, $w[i] = v[\beta(i)]$

Illustration.

$u =$ COMPUTER, $v =$ ORDINATEUR, $w =$ OTER

- $\alpha = \langle 2, 6, 7, 8 \rangle$;
- $\beta = \langle 1, 7, 8, 10 \rangle$.

Objectif. Calcul d'une plus longue sous-séquence commune (PLSC).

Sous-problèmes

En raisonnant sur la dernière lettre appariée, une plus longue sous-séquence commune de u et v est :

- soit une plus longue sous-séquence commune de $u[1, m-1]$ et v ;
- soit une plus longue sous-séquence commune de u et $v[1, n-1]$;
- soit une plus longue sous-séquence commune de $u[1, m-1]$ et $v[1, n-1]$ étendue par $u[m] = v[n]$.

Les sous-problèmes sont donc les $(m+1)(n+1)$ sous-mots $u[1, i]$ et $v[1, j]$.

Un ordre partiel approprié peut être défini par $(i, j) < (i', j')$ si :

- $i < i'$ et $j \leq j'$;
- ou $i \leq i'$ et $j < j'$.

Relation entre sous-problèmes

Soit le sous-problème défini par $u[1,i]$ et $v[1,j]$.
Si $u[i] = v[j]$ alors il existe une PLSC qui apparie $u[i]$ et $v[j]$.

Preuve.

Soit w une PLSC.

- Si w n'apparie ni $u[i]$ ni $v[j]$ alors on peut l'etendre par $u[i] = v[j]$ d'où une contradiction.
- Supposons que w apparie $u[i]$ et $v[k]$ pour $k < j$ alors w est aussi obtenue en appariant $u[i]$ et $v[j]$.

D'où en notant $\ell[i,j]$ la longueur de la PSLC de $u[1,i]$ et $v[1,j]$:

- Si $u[i] = v[j]$ alors $\ell[i,j] = \ell[i-1,j-1] + 1$;
- Sinon $\ell[i,j] = \max(\ell[i-1,j], \ell[i,j-1])$.

Observation. $2\ell[i,j] = i + j - edit_{di}(u[1,i], v[1,j])$.

(preuve soit directe soit par induction)

Calcul d'une PLSC

Calcul des longueurs et préparation pour la PLSC

For i **from** 0 **to** m **do** $\ell[i,0] \leftarrow 0$; **For** j **from** 1 **to** n **do** $\ell[0,j] \leftarrow 0$;
For i **from** 1 **to** m **do**
  For j **from** 1 **to** n **do**
    If $u[i] = v[j]$ **then** $\ell[i,j] \leftarrow \ell[i-1,j-1] + 1$;
    Else
      $\ell[i,j] \leftarrow \ell[i-1,j]$;
      If $\ell[i,j] < \ell[i,j-1]$ **then** $\ell[i,j] \leftarrow \ell[i,j-1]$;

Ecriture de la PLSC

$i \leftarrow m$; $j \leftarrow n$; $k \leftarrow \ell[i,j]$;
While $k > 0$ **do**
  If $\ell[i,j] = \ell[i-1,j]$ **then** $i \leftarrow i-1$;
  Else If $\ell[i,j] = \ell[i,j-1]$ **then** $j \leftarrow j-1$;
  Else $w[k] \leftarrow u[i]$; $i \leftarrow i-1$; $j \leftarrow j-1$; $k \leftarrow k-1$;

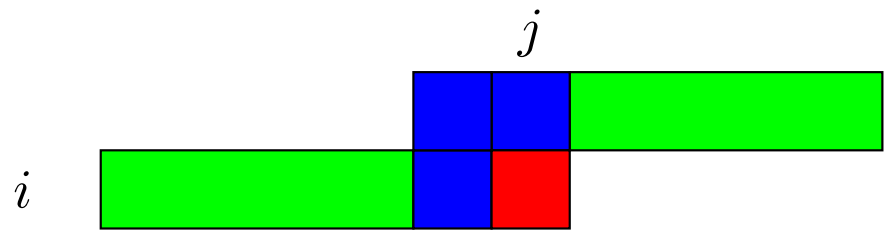
Considérations de mémoire

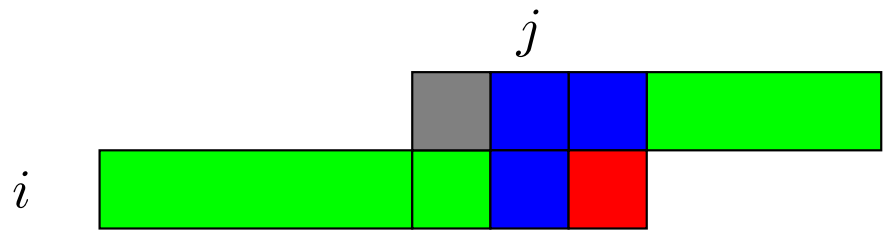
La mémoire utilisée est en $\Theta(mn)$.

Si on cherche uniquement à connaître la longueur, on peut économiser la mémoire.

Le problème (i, j) ne dépend que des problèmes $(i-1, j-1)$, $(i-1, j)$ et $(i, j-1)$.

Il suffit de conserver $\ell[i, 1], \ldots, \ell[i, j-1], \ell[i-1, j-1], \ell[i-1, j], \ldots, \ell[i-1, n]$ qu'on notera $\ell[1], \ldots, \ell[j-1], old\ell, \ell[j], \ldots, \ell[n]$.





Calcul des longueurs

Length(u, m, v, n, ℓ)

For j **from** 0 **to** n **do** $\ell[j] \leftarrow 0$;

For i **from** 1 **to** m **do**

  $old\ell \leftarrow 0$;

  For j **from** 1 **to** n **do**

    $temp \leftarrow \ell[j]$;

    If $u[i] = v[j]$ **then** $\ell[j] \leftarrow old\ell + 1$;

    Else If $\ell[j] < \ell[j-1]$ **then** $\ell[j] \leftarrow \ell[j-1]$;

    $old\ell \leftarrow temp$;

Observations

- A la fin de l'algorithme pour tout j, $\ell[j] = |PLSC(u, v[1, j])|$.
- En permutant si nécessaire u et v, la mémoire utilisée est en $\Theta(\min(m, n))$.

Diviser pour régner

On note $\tilde{u}$ le mot miroir de u et ε le mot vide.

ℓ et $\tilde{\ell}$ sont des tableaux indicés de 0 à n, initialisés à 0.

PLSC(u, m, v, n)

If $n = 0$ **then return**(ε) ;

If $m = 1$ **then**

  For j **from** 1 **to** n **do** **if** $u[1] = v[j]$ **then return**(u) ;

  return(ε) ;

$mid \leftarrow \lfloor \frac{m}{2} \rfloor$;

Détermination des longueurs des PLSC pour $(u[1, mid], v)$ et $(\tilde{u}[1, m - mid], \tilde{v})$

Length$(u[1, mid], mid, v, n, \ell)$; Length$(\tilde{u}[1, m - mid], m - mid, \tilde{v}, n, \tilde{\ell})$;

Recherche de la meilleure partition

$max\ell \leftarrow -\infty$;

For j **from** 0 **to** n **do** **if** $\ell[j] + \tilde{\ell}[n - j] > max\ell$ **then** $max\ell \leftarrow \ell[j] + \tilde{\ell}[n - j]$; $k \leftarrow j$;

Appels récursifs et concaténation

If $k > 0$ **then** $w_1 \leftarrow$ PLSC$(u[1, mid], mid, v[1, k], k)$ **else** $w_1 \leftarrow \varepsilon$;

If $k < n$ **then** $w_2 \leftarrow$ PLSC$(u[mid + 1, m], m - mid, v[k + 1, n], n - k)$ **else** $w_2 \leftarrow \varepsilon$;

return CONCAT(w_1, w_2) ;

Illustration (1)

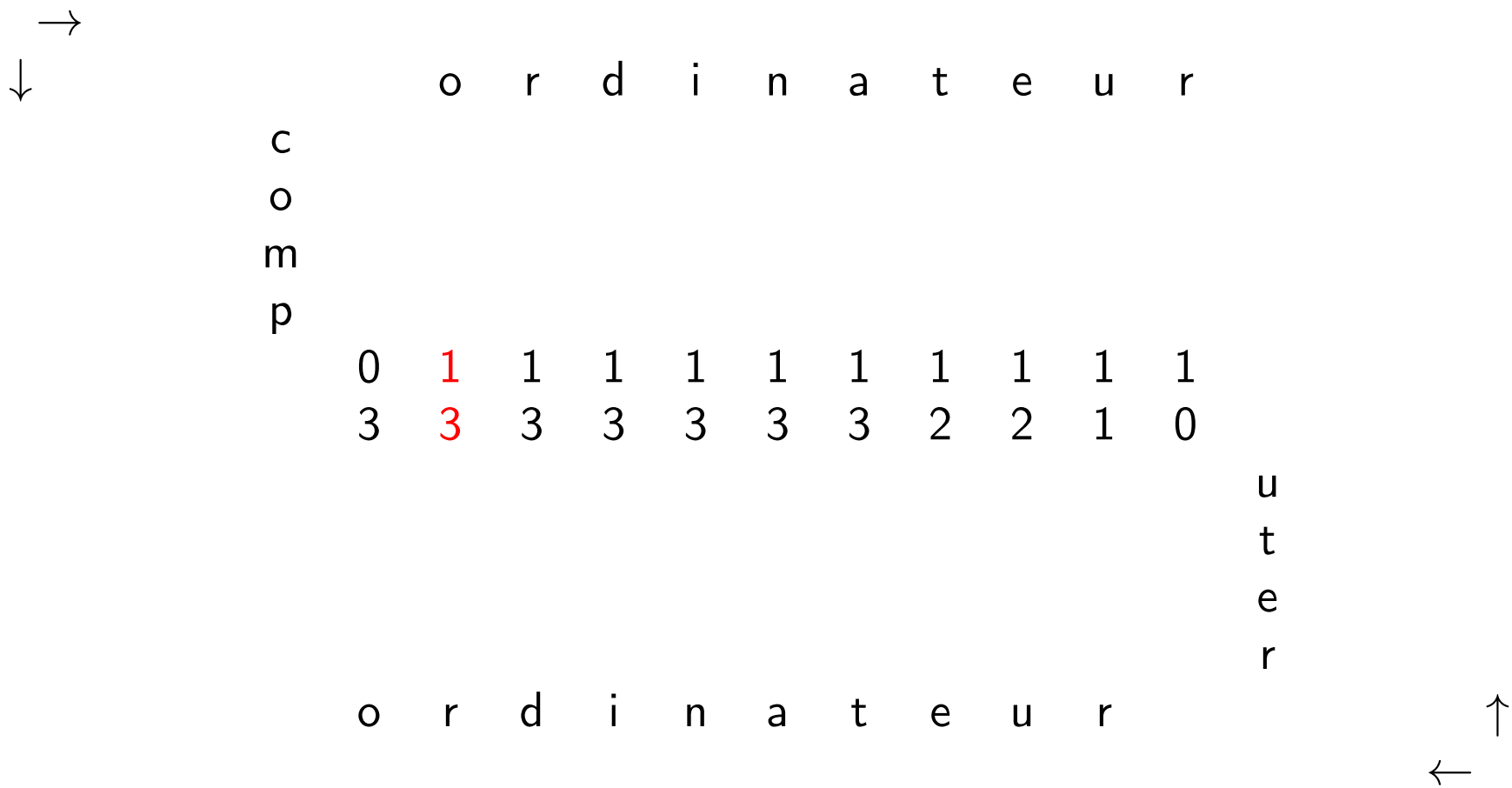
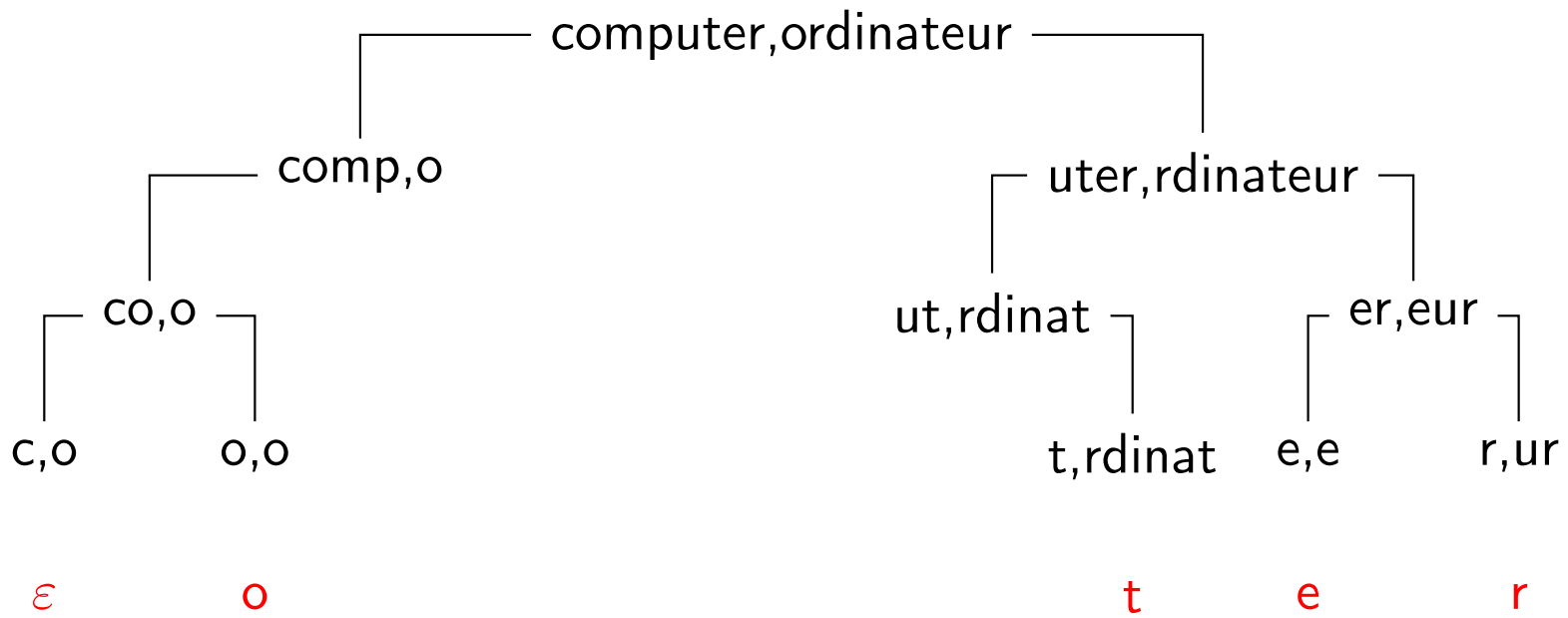


Illustration (2)



Analyse de complexité

Espace

ℓ et $\tilde{\ell}$ peuvent être des variables globales, d'où $\Theta(n)$.

Les variables locales vérifient $|w_1| \leq m$ et $|w_2| \leq m$.

A chaque appel récursif m est divisé par deux, d'où $\Theta(m)$.

En sommant, on obtient $\Theta(m+n)$.

Temps

$$T(m,n) \leq \max_{k \leq n}(T(\frac{m}{2},k) + T(\frac{m}{2},n-k)) + Cmn$$

$$T(i,n) \leq Cn \text{ pour } i \in \{0,1\}$$

$$T(m,i) \leq Cm \text{ pour } i \in \{0,1\}$$

Par induction pour $m \geq 1$ and $n \geq 1$,

$$T(m,n) \leq 2Cmn$$

Développements possibles

- Calcul de la fonction π en $\Theta(m)$
- Complexité de l'algorithme de Morris-Pratt (au plus $2n - 1$ comparaisons)
- Construction et propriétés de l'ε-automate d'une expression régulière
- Recherche d'un motif dans un texte avec caractères joker
- Calcul d'une PLSC