Recherche exacte de motifs

Synthèse et conclusion

Sèverine Bérard

8 octobre 2020

ISE-M – FDS, Université de Montpellier

Plan du cours

Plan du cours

- Survol rapide de deux autres méthodes de recherche exacte de motif :
  - Recherche au moyen d'automates finis (Réf. Intro algo)
  - Algorithme Shift-Or (Réf. Gusfield)
- Synthèse basée sur un article de review de Faro et Lecroq

Plan du cours

Principes

- Construire un *automate fini* qui balaye le texte T à la recherche de toutes les occurrences du motif P

Principes

- Construire un *automate fini* qui balaye le texte T à la recherche de toutes les occurrences du motif P

- Efficace : besoin d'examiner chaque caractère du texte une et une seule fois

Principes

- Construire un *automate fini* qui balaye le texte T à la recherche de toutes les occurrences du motif P
- Efficace : besoin d'examiner chaque caractère du texte une et une seule fois
- Recherche en $O(n)$

Principes

- Construire un *automate fini* qui balaye le texte T à la recherche de toutes les occurrences du motif P

- Efficace : besoin d'examiner chaque caractère du texte une et une seule fois

- Recherche en $O(n)$

- Mais il faut construire l'automate avant, temps potentiellement grand si l'alphabet est grand

Automate fini

Un automate fini M est un quintuplet $(\mathcal{Q}, q_0, A, \Sigma, \delta)$ où :

- $\mathcal{Q}$ est un ensemble fini d'*états*
- $q_o \in \mathcal{Q}$ est l'*état initial*
- $A \subseteq \mathcal{Q}$ est un ensemble distingués d'*états terminaux*
- Σ est un *alphabet* fini
- δ est une fonction de $\mathcal{Q} \times \Sigma$ vers $\mathcal{Q}$, appelée *fonction de transition* de M

Automate fini

Un automate fini M est un quintuplet $(\mathcal{Q}, q_0, A, \Sigma, \delta)$ où :

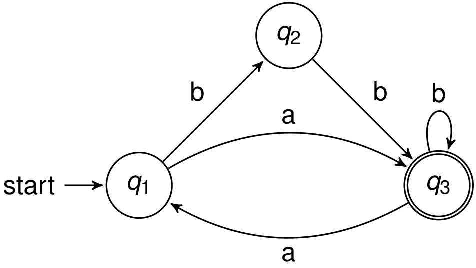
- $\mathcal{Q}$ est un ensemble fini d'*états*
- $q_o \in \mathcal{Q}$ est l'*état initial*
- $A \subseteq \mathcal{Q}$ est un ensemble distingués d'*états terminaux*
- Σ est un *alphabet* fini
- δ est une fonction de $\mathcal{Q} \times \Sigma$ vers $\mathcal{Q}$, appelée *fonction de transition* de M

L'automate en action

- L'automate fini démarre à l'état q_0 et lit les caractères de la chaîne d'entrée un par un
- S'il se trouve dans l'état q et lit le caractère a, il passe à l'état $\delta(q, a)$
- À chaque fois que l'état courant $q \in A$, on dit que M a *accepté* la chaîne lue (sinon elle est *rejetée*)

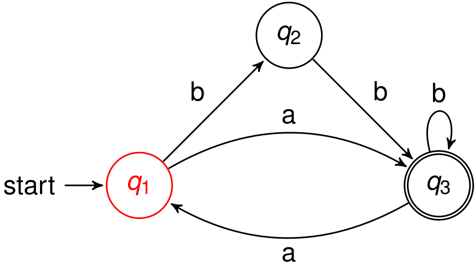
Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



Automate fini : un exemple

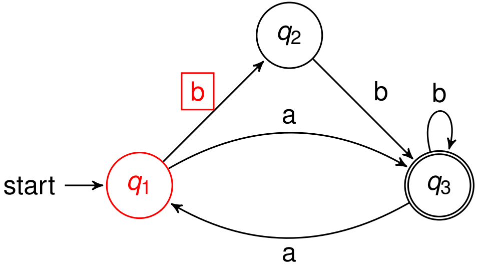
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



w_1 = bba

Automate fini : un exemple

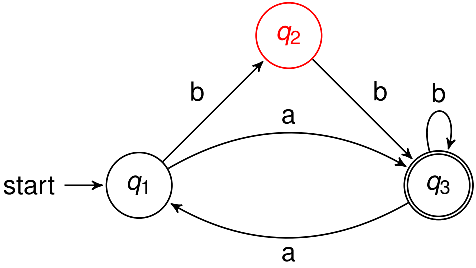
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



w_1 = bba

Automate fini : un exemple

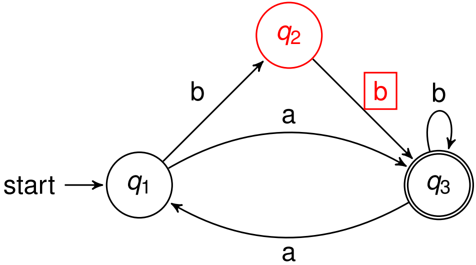
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



w_1 = bba

Automate fini : un exemple

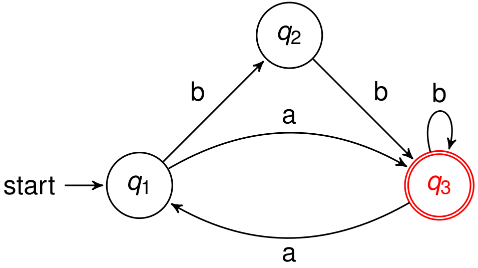
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \ldots$



w_1 = bba

Automate fini : un exemple

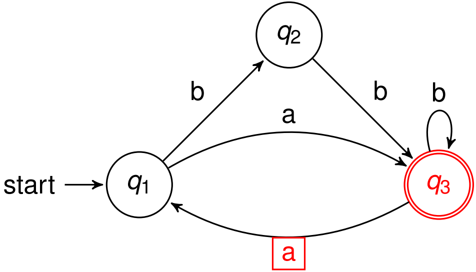
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



w_1 = bba

Automate fini : un exemple

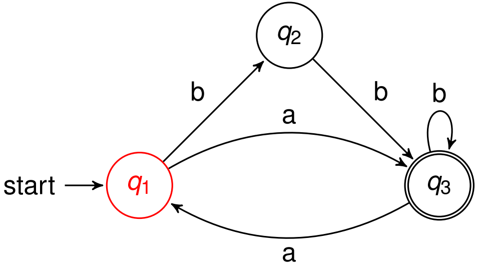
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \ldots$



w_1 = bba

Automate fini : un exemple

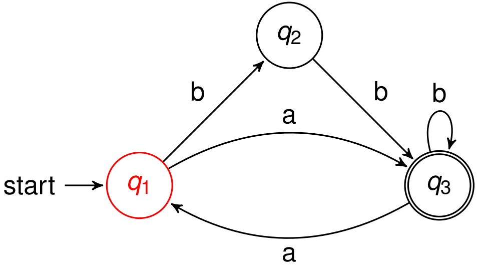
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \ldots$



w_1 = bba

Automate fini : un exemple

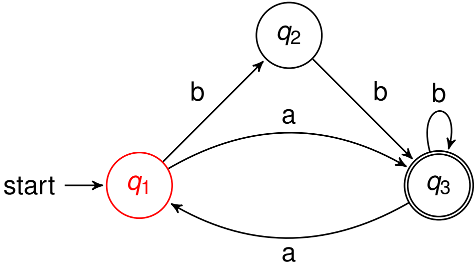
- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



w_1 = bba : rejetée

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \dots$

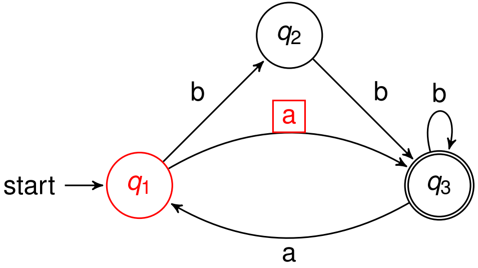


w_1 = bba : rejetée

w_2 = aaa

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...

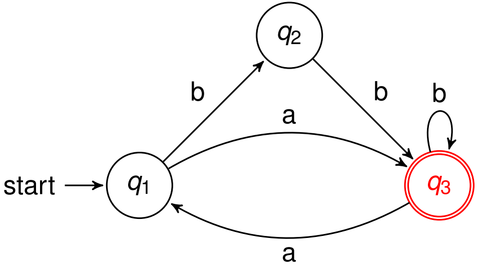


w_1 = bba : rejetée

w_2 = aaa

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \ldots$

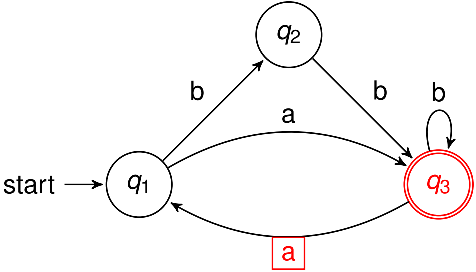


w_1 = bba : rejetée

w_2 = aaa

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \ldots$

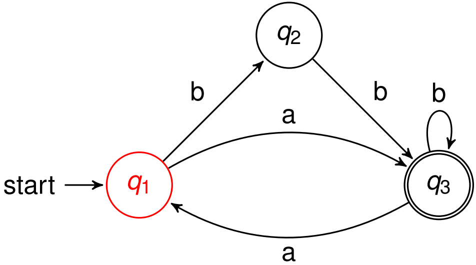


w_1 = bba : rejetée

w_2 = aaa

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...

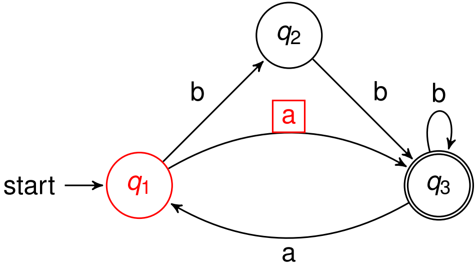


w_1 = bba : rejetée

w_2 = aaa

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \ldots$

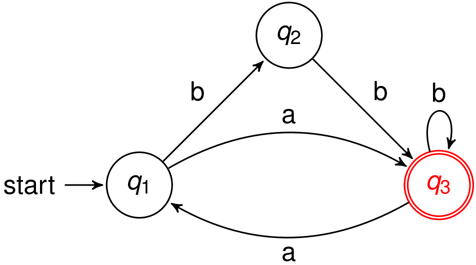


w_1 = bba : rejetée

w_2 = aaa

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...

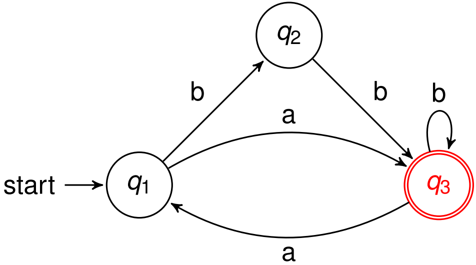


w_1 = bba : rejetée

w_2 = aaa

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...

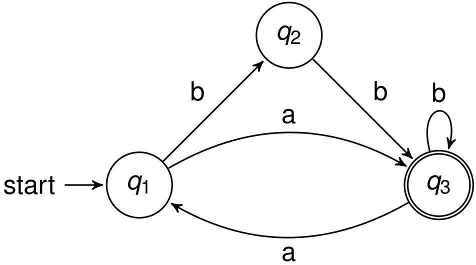


w_1 = bba : rejetée

w_2 = aaa : acceptée

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



w_1 = bba : rejetée

w_2 = aaa : acceptée

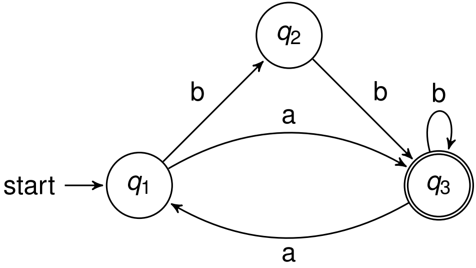
Fonction d'état final

Fonction Φ de Σ^* vers Q telle que $\Phi(w)$ est l'état dans lequel se trouve M après avoir traité la chaîne w. Définition récursive :

- $\Phi(\varepsilon) = q_0$
- $\Phi(wa) = \delta(\Phi(w), a)$, pour $w \in \Sigma^*$, $a \in \Sigma$

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2, \ldots$



w_1 = bba : rejetée
w_2 = aaa : acceptée

$\Phi(w_1) = q_1$

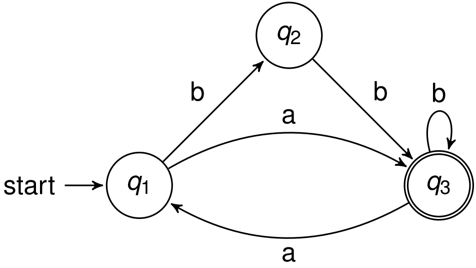
Fonction d'état final

Fonction Φ de Σ^* vers Q telle que $\Phi(w)$ est l'état dans lequel se trouve M après avoir traité la chaîne w. Définition récursive :

- $\Phi(\varepsilon) = q_0$
- $\Phi(wa) = \delta(\Phi(w), a)$, pour $w \in \Sigma^*$, $a \in \Sigma$

Automate fini : un exemple

- $Q = \{q_1, q_2, q_3\}$
- q_1 est l'état initial
- $A = \{q_3\}$
- $\Sigma = \{a, b\}$
- $\delta(q_1, a) = q_3$,
  $\delta(q_1, b) = q_2$, ...



w_1 = bba : rejetée
w_2 = aaa : acceptée

$\Phi(w_1) = q_1$
$\Phi(w_2) = q_3$

Fonction d'état final

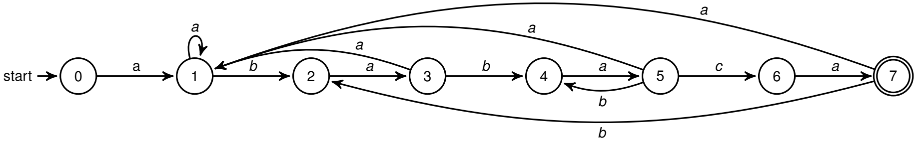
Fonction Φ de Σ^* vers Q telle que $\Phi(w)$ est l'état dans lequel se trouve M après avoir traité la chaîne w. Définition récursive :

- $\Phi(\varepsilon) = q_0$
- $\Phi(wa) = \delta(\Phi(w), a)$, pour $w \in \Sigma^*$, $a \in \Sigma$

But : construire un automate pour un motif $P = ababaca$

Automates de recherche de motif

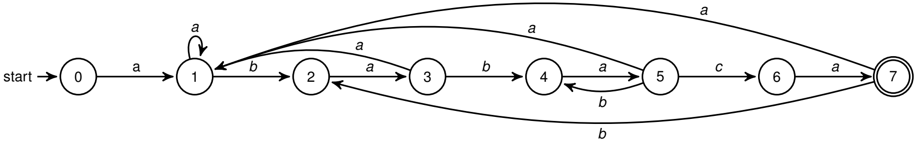
But : construire un automate pour un motif $P = ababaca$



Tous les arcs non montrés pointent vers l'état initial

Automates de recherche de motif

But : construire un automate pour un motif $P = ababaca$

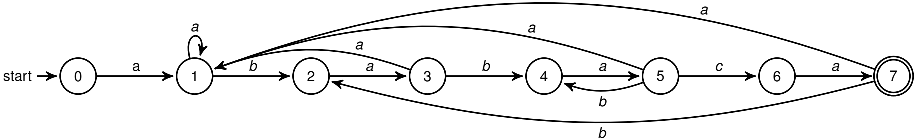


Tous les arcs non montrés pointent vers l'état initial

Cet automate accepte toutes les chaînes finissant par P

Automates de recherche de motif

But : construire un automate pour un motif $P = ababaca$



Tous les arcs non montrés pointent vers l'état initial

Cet automate accepte toutes les chaînes finissant par P

Exemple avec le texte $T = ababababacaba$

i	–	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	–	a	b	a	b	a	b	a	c	a	b	a
état $\Phi(T[1..i])$	0	1	2	3	4	5	4	5	6	7	2	3

Construire l'automate pour un motif P

La fonction suffixe σ associée à P

C'est une application de Σ^* vers $[0..m]$ telle que $\sigma(w)$ est la longueur du plus long préfixe de P qui est un suffixe de w.

Pour une chaîne P de longueur m, on a $\sigma(w) = m$ ssi P est suffixe de w

Si v est suffixe de w, alors $\sigma(v) \leqslant \sigma(w)$

Construire l'automate pour un motif P

La fonction suffixe σ associée à P

C'est une application de Σ^* vers $[0..m]$ telle que $\sigma(w)$ est la longueur du plus long préfixe de P qui est un suffixe de w.
Pour une chaîne P de longueur m, on a $\sigma(w) = m$ ssi P est suffixe de w
Si v est suffixe de w, alors $\sigma(v) \leqslant \sigma(w)$

Définition de l'automate de recherche de P :

- L'ensemble de états $\mathcal{Q}$ est $[0..m]$
- L'état initial est l'état 0 et le seul état final est m
- Fonction de transition : $\forall q \in \mathcal{Q}, \forall a \in \Sigma, \delta(q, a) = \sigma(P[1..q].a)$

Construire l'automate pour un motif P

La fonction suffixe σ associée à P

C'est une application de Σ^* vers $[0..m]$ telle que $\sigma(w)$ est la longueur du plus long préfixe de P qui est un suffixe de w.
Pour une chaîne P de longueur m, on a $\sigma(w) = m$ ssi P est suffixe de w
Si v est suffixe de w, alors $\sigma(v) \leqslant \sigma(w)$

Définition de l'automate de recherche de P :

- L'ensemble de états $\mathcal{Q}$ est $[0..m]$
- L'état initial est l'état 0 et le seul état final est m
- Fonction de transition : $\forall q \in \mathcal{Q},\ \forall a \in \Sigma,\ \delta(q, a) = \sigma(P[1..q].a)$

Invariant

$$\Phi(T[1..i]) = \sigma(T[1..i])$$

Prétraitement

Algorithme : Calcul de la fonction de transition

Données : P de longueur m et Σ l'alphabet

pour *(q de 0 à m)* **faire**
    pour chaque $a \in \Sigma$ **faire**
        $k := \min(m+1, q+2)$;
        répéter
            $k := k - 1$;
        jusqu'à *($P[1..k]$ suffixe de $P[1..q].a$)*;
        $\delta(q, a) := k$;

Retourner δ;

Prétraitement

Algorithme : Calcul de la fonction de transition

Données : P de longueur m et Σ l'alphabet

pour *(q de 0 à m)* **faire**
  pour chaque $a \in \Sigma$ **faire**
    $k := \min(m+1, q+2)$;
    répéter
      $k := k - 1$;
    jusqu'à *($P[1..k]$ suffixe de $P[1..q].a$)*;
    $\delta(q, a) := k$;

Retourner δ;

- Complexité en temps de cet algorithme :

Prétraitement

Algorithme : Calcul de la fonction de transition

Données : P de longueur m et Σ l'alphabet

pour *(q de 0 à m)* **faire**
  pour chaque $a \in \Sigma$ **faire**
    $k := \min(m+1, q+2)$;
    répéter
      $k := k-1$;
    jusqu'à *($P[1..k]$ suffixe de $P[1..q].a$)*;
    $\delta(q,a) := k$;

Retourner δ;

- Complexité en temps de cet algorithme : $O(m^3|\Sigma|)$

Prétraitement

Algorithme : Calcul de la fonction de transition

Données : P de longueur m et Σ l'alphabet

pour *(q de 0 à m)* **faire**
  pour chaque $a \in \Sigma$ **faire**
    $k := \min(m+1, q+2)$;
    répéter
      $k := k-1$;
    jusqu'à *(*$P[1..k]$ *suffixe de* $P[1..q].a$*)*;
    $\delta(q,a) := k$;

Retourner δ;

- Complexité en temps de cet algorithme : $O(m^3|\Sigma|)$
- Des procédures plus rapides existent en $O(m|\Sigma|)$

Phase de recherche

Algorithme : Recherche avec automate fini

Données : T de longueur n, δ la fonction de transition associée à P de longueur m

$q := 0$;

pour *(i de 1 à n)* **faire**

  $q := \delta(q, T[i])$;

  si $q = m$ **alors**

    $s := i - m$;

    Écrire("P apparaît à la position ", s);

Phase de recherche

Algorithme : Recherche avec automate fini

Données : T de longueur n, δ la fonction de transition associée à P de longueur m

$q := 0$;

pour *(i de 1 à n)* **faire**

   $q := \delta(q, T[i])$;

   si $q = m$ **alors**

      $s := i - m$;

      Écrire("P apparaît à la position ", s);

- Complexité en temps de cet algorithme :

Phase de recherche

Algorithme : Recherche avec automate fini

Données : T de longueur n, δ la fonction de transition associée à P de longueur m

$q := 0$;

pour *(i de 1 à n)* **faire**

  $q := \delta(q, T[i])$;

  si $q = m$ **alors**

    $s := i - m$;

    Écrire("P apparaît à la position ", s);

- Complexité en temps de cet algorithme : $O(n)$

Phase de recherche

Algorithme : Recherche avec automate fini

Données : T de longueur n, δ la fonction de transition associée à P de longueur m

$q := 0$;

pour *(i de 1 à n)* **faire**

  $q := \delta(q, T[i])$;

  si $q = m$ **alors**

    $s := i - m$;

    Écrire("*P* apparaît à la position ", s);

- Complexité en temps de cet algorithme : $O(n)$
- Ne prend pas en compte le temps de pré-traitement

Plan du cours

Algorithme Shift-Or (S0)

- R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. Commun. ACM, 35(10) :74–82, 1992 (*Gusfield préfère "Shift-**And**"*)
- Méthode simple, basée sur la comparaison de bits, très efficace pour les motifs courts

Algorithme Shift-Or (S0)

- R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. Commun. ACM, 35(10) :74–82, 1992 (*Gusfield préfère “Shift-**And**"*)
- Méthode simple, basée sur la comparaison de bits, très efficace pour les motifs courts

Matrice M de $m \times n$ valeurs binaires

- $M(i,j) = 1$ ssi les i premiers caractères de P sont identiques aux i caractères de T terminant au caractère j
- Sinon, $M(i,j) = 0$

Autrement dit $M(i,j) = 1$ ssi $P[1..i]$ matche exactement $T[j-i+1..j]$

Algorithme Shift-Or (S0)

- R. Baeza-Yates and G. H. Gonnet. A new approach to text searching. Commun. ACM, 35(10) :74–82, 1992 (*Gusfield préfère "Shift-**And**"*)
- Méthode simple, basée sur la comparaison de bits, très efficace pour les motifs courts

Matrice M de $m \times n$ valeurs binaires

- $M(i,j) = 1$ ssi les i premiers caractères de P sont identiques aux i caractères de T terminant au caractère j
- Sinon, $M(i,j) = 0$

Autrement dit $M(i,j) = 1$ ssi $P[1..i]$ matche exactement $T[j-i+1..j]$

- Calculer la dernière ligne de M résout exactement le problème de recherche de P dans T
- Comment va-t-on procéder pour construire M ?

Exemple

$P = for$ et $T = california$, alors M :

$$\begin{array}{ccccccccccc}
 & c & a & l & i & f & o & r & n & i & a \\
f & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
o & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
r & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0
\end{array}$$

Exemple

$P = for$ et $T = california$, alors M :

	c	a	l	i	f	o	r	n	i	a
f	0	0	0	0	1	0	0	0	0	0
o	0	0	0	0	0	1	0	0	0	0
r	0	0	0	0	0	0	1	0	0	0

- $U(a)$, vecteurs binaires de longueur m pour chaque $a \in \Sigma$, $U(a)$ est mis à 1 aux positions de P où a apparaît

Exemple

$P = for$ et $T = california$, alors M :

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*
o	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*
r	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*

- $U(a)$, vecteurs binaires de longueur m pour chaque $a \in \Sigma$, $U(a)$ est mis à 1 aux positions de P où a apparaît
  Ex : $U(f) = 100$ et $U(g) = 000$

Exemple

$P = for$ et $T = california$, alors M :

	c	a	l	i	f	o	r	n	i	a
f	0	0	0	0	1	0	0	0	0	0
o	0	0	0	0	0	1	0	0	0	0
r	0	0	0	0	0	0	1	0	0	0

- $U(a)$, vecteurs binaires de longueur m pour chaque $a \in \Sigma$, $U(a)$ est mis à 1 aux positions de P où a apparaît
  Ex : $U(f) = 100$ et $U(g) = 000$
- Opération `Bit-Shift` décale tous les bits d'un vecteur d'une position vers la fin et positionne le 1er bit à 1

Exemple

$P = for$ et $T = california$, alors M :

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*
o	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*
r	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*

- $U(a)$, vecteurs binaires de longueur m pour chaque $a \in \Sigma$, $U(a)$ est mis à 1 aux positions de P où a apparaît
  Ex : $U(f) = 100$ et $U(g) = 000$
- Opération `Bit-Shift` décale tous les bits d'un vecteur d'une position vers la fin et positionne le 1^{er} bit à 1
  Ex : $\texttt{Bit-Shift}(U(f)) = 110$

Exemple

$P = for$ et $T = california$, alors M :

	c	a	l	i	f	o	r	n	i	a
f	0	0	0	0	1	0	0	0	0	0
o	0	0	0	0	0	1	0	0	0	0
r	0	0	0	0	0	0	1	0	0	0

- $U(a)$, vecteurs binaires de longueur m pour chaque $a \in \Sigma$, $U(a)$ est mis à 1 aux positions de P où a apparaît
  Ex : $U(f) = 100$ et $U(g) = 000$
- Opération `Bit-Shift` décale tous les bits d'un vecteur d'une position vers la fin et positionne le 1^{er} bit à 1
  Ex : $\texttt{Bit-Shift}(U(f)) = 110$
- Notation : $M(j)$ est la j^e colonne de M. $\texttt{Bit-Shift}(M(6)) = 101$

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	a	l	i	f	o	r	n	i	a
f										
o										
r										

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

$$\begin{array}{c|cccccccccc} & c & a & l & i & f & o & r & n & i & a \\ f & & & & & & & & & & \\ o & & & & & & & & & & \\ r & & & & & & & & & & \end{array}$$

Initialisation : $P[1] = f \neq T[1] = c$

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*									
o	*0*									
r	*0*									

Initialisation : $P[1] = f \neq T[1] = c$

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*									
o	*0*									
r	*0*									

$M(2) = ($`Bit-Shift`$(M(1))$ `ET` $U(a))$ = (100 `ET` 000)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	0	0								
o	0	0								
r	0	0								

$M(2) = (\texttt{Bit-Shift}(M(1))\ \texttt{ET}\ U(a))$ = (100 `ET` 000)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\texttt{ET}$ bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*								
o	*0*	*0*								
r	*0*	*0*								

$M(3) = (\texttt{Bit-Shift}(M(2))\ \texttt{ET}\ U(l))$ = (100 $\texttt{ET}$ 000)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\texttt{ET}$ bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	0	0	0							
o	0	0	0							
r	0	0	0							

$M(3) = (\texttt{Bit-Shift}(M(2))\ \texttt{ET}\ U(l))$ = $(100\ \texttt{ET}\ 000)$

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*							
o	*0*	*0*	*0*							
r	*0*	*0*	*0*							

$M(4) =$ (`Bit-Shift`$(M(3))$ `ET` $U(i)$) = (100 `ET` 000)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*	*0*						
o	*0*	*0*	*0*	*0*						
r	*0*	*0*	*0*	*0*						

$M(4) =$ (`Bit-Shift`$(M(3))$ `ET` $U(i)$) = (100 `ET` 000)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	0	0	0	0						
o	0	0	0	0						
r	0	0	0	0						

$M(5) =$ (`Bit-Shift`$(M(4))$ `ET` $U(f)$) = (100 `ET` 100)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\texttt{ET}$ bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	0	0	0	0	1					
o	0	0	0	0	0					
r	0	0	0	0	0					

$M(5) = (\texttt{Bit-Shift}(M(4))\ \texttt{ET}\ U(f))$ = $(100\ \texttt{ET}\ 100)$

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\texttt{ET}$ bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*	*0*	*1*					
o	*0*	*0*	*0*	*0*	*0*					
r	*0*	*0*	*0*	*0*	*0*					

$M(6) = (\texttt{Bit-Shift}(M(5))\ \texttt{ET}\ U(o))$ = (110 $\texttt{ET}$ 010)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\texttt{ET}$ bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*	*0*	*1*	*0*				
o	*0*	*0*	*0*	*0*	*0*	*1*				
r	*0*	*0*	*0*	*0*	*0*	*0*				

$M(6) = (\texttt{Bit-Shift}(M(5))\ \texttt{ET}\ U(o))$ = (110 $\texttt{ET}$ 010)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\texttt{ET}$ bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	0	0	0	0	1	0				
o	0	0	0	0	0	1				
r	0	0	0	0	0	0				

$M(7) = (\texttt{Bit-Shift}(M(6))\ \texttt{ET}\ U(r))$ = (101 $\texttt{ET}$ 001)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*	*0*	*1*	*0*	*0*			
o	*0*	*0*	*0*	*0*	*0*	*1*	*0*			
r	*0*	*0*	*0*	*0*	*0*	*0*	*1*			

$M(7) =$ (`Bit-Shift`$(M(6))$ `ET` $U(r)$) = (101 `ET` 001)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\texttt{ET}$ bit à bit entre $\texttt{Bit-Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	0	0	0	0	1	0	0			
o	0	0	0	0	0	1	0			
r	0	0	0	0	0	0	1			

$M(8) = (\texttt{Bit-Shift}(M(7))\ \texttt{ET}\ U(n))$ = (100 $\texttt{ET}$ 000)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération `ET` bit à bit entre `Bit-Shift`$(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	a	l	i	f	o	r	n	i	a
f	0	0	0	0	1	0	0	0		
o	0	0	0	0	0	1	0	0		
r	0	0	0	0	0	0	1	0		

$M(8) = ($`Bit-Shift`$(M(7))$ `ET` $U(n))$ = (100 `ET` 000)

Construction de M

On procède colonne par colonne :

- Colonne 1 : que des 0 si $T[1] \neq P[1]$, sinon $M(1,1) = 1$ et le reste à 0
- Les autres colonnes $j > 1$ sont obtenues par une opération $\mathtt{ET}$ bit à bit entre $\mathtt{Bit\text{-}Shift}(M(j-1))$ et $U(T[j])$

Exemple : $P = for$, $T = california$, $U(a) = U(c) = U(i) = U(l) = U(n) = 000$, $U(f) = 100$, $U(o) = 010$, $U(r) = 001$

	c	*a*	*l*	*i*	*f*	*o*	*r*	*n*	*i*	*a*
f	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*
o	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*
r	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*

$M(9) = \dots$

Shift-Or conclusion

- Complexité dans le pire des cas $O(m \times n)$

Shift-Or conclusion

- Complexité dans le pire des cas $O(m \times n)$

- Mais devient très efficace si la taille des motifs est inférieure à la longueur d'un mot machine :
  - Chaque colonne de M et chaque vecteur U sont encodés par un seul mot machine
  - `Bit-Shift` et `ET` deviennent des opérations sur un seul mot machine

$\longrightarrow$ quasi-linéaire en pratique

Shift-Or conclusion

- Complexité dans le pire des cas $O(m \times n)$

- Mais devient très efficace si la taille des motifs est inférieure à la longueur d'un mot machine :
  - Chaque colonne de M et chaque vecteur U sont encodés par un seul mot machine
  - `Bit-Shift` et `ET` deviennent des opérations sur un seul mot machine

  $\longrightarrow$ quasi-linéaire en pratique

- Seules 2 colonnes de M ont besoin d'être gardées en mémoire
  $\longrightarrow$ efficace aussi en espace

Plan du cours

Synthèse basée sur un article scientifique

- The exact string matching problem : a comprehensive experimental evaluation, S. Faro, T. Lecroq. arXiv preprint arXiv :1012.2547, 2010.

The Exact String Matching Problem:
a Comprehensive Experimental Evaluation

Simone Faro[†] and Thierry Lecroq[‡]

[†]Università degli Studi di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125, Catania, Italy
faro@dmi.unict.it

[‡]Université de Rouen, LITIS EA 4108, 76821 Mont-Saint-Aignan Cedex, France
thierry.lecroq@univ-rouen.fr

- The exact online string matching problem : A review of the most recent results, S. Faro, T. Lecroq. Journal ACM Computing Surveys (CSUR) Volume 45 Issue 2, February 2013.

A

The Exact Online String Matching Problem:
a Review of the Most Recent Results

SIMONE FARO, Università di Catania
THIERRY LECROQ, Université de Rouen

Synthèse basée sur un article scientifique

- The exact string matching problem : a comprehensive experimental evaluation, S. Faro, T. Lecroq. arXiv preprint arXiv :1012.2547, 2010.

The Exact String Matching Problem:
a Comprehensive Experimental Evaluation

Simone Faro[†] and Thierry Lecroq[‡]

[†]Università degli Studi di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125, Catania, Italy
faro@dmi.unict.it

[‡]Université de Rouen, LITIS EA 4108, 76821 Mont-Saint-Aignan Cedex, France
thierry.lecroq@univ-rouen.fr

- The exact online string matching problem : A review of the most recent results, S. Faro, T. Lecroq. Journal ACM Computing Surveys (CSUR) Volume 45 Issue 2, February 2013.

**The Exact Online String Matching Problem:
a Review of the Most Recent Results**

SIMONE FARO, Università di Catania
THIERRY LECROQ, Université de Rouen

Synthèse basée sur un article scientifique

- The exact string matching problem : a comprehensive experimental evaluation, S. Faro, T. Lecroq. arXiv preprint arXiv :1012.2547, 2010.

The Exact String Matching Problem:
a Comprehensive Experimental Evaluation

Simone Faro[†] and Thierry Lecroq[‡]

[†]Università degli Studi di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125, Catania, Italy
faro@dmi.unict.it

[‡]Université de Rouen, LITIS EA 4108, 76821 Mont-Saint-Aignan Cedex, France
thierry.lecroq@univ-rouen.fr

- The exact online string matching problem : A review of the most recent results, S. Faro, T. Lecroq. Journal ACM Computing Surveys (CSUR) Volume 45 Issue 2, February 2013.

A

**The Exact Online String Matching Problem:
a Review of the Most Recent Results**

SIMONE FARO, Università di Catania
THIERRY LECROQ, Université de Rouen

Synthèse basée sur un article scientifique

- The exact string matching problem : a comprehensive experimental evaluation, S. Faro, T. Lecroq. arXiv preprint arXiv :1012.2547, 2010.

The Exact String Matching Problem:
a Comprehensive Experimental Evaluation

Simone Faro† and Thierry Lecroq‡

†Università degli Studi di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125, Catania, Italy
faro@dmi.unict.it

‡Université de Rouen, LITIS EA 4108, 76821 Mont-Saint-Aignan Cedex, France
thierry.lecroq@univ-rouen.fr

- The exact online string matching problem : A review of the most recent results, S. Faro, T. Lecroq. Journal ACM Computing Surveys (CSUR) Volume 45 Issue 2, February 2013.

The Exact Online String Matching Problem:
a Review of the Most Recent Results

SIMONE FARO, Università di Catania
THIERRY LECROQ, Université de Rouen

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant$ 50 % après 2000)

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant$ 50 % après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant 50$ % après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

String matching problem

Étant donnés un texte T de longueur n et un motif P de longueur m sur un alphabet Σ, trouver toutes les occurrences de P dans T

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant 50$ % après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

String matching problem

Étant donnés un texte T de longueur n et un motif P de longueur m sur un alphabet Σ, trouver toutes les occurrences de P dans T

- Énormément d'applications qui nécessitent des solutions différentes selon qui du motif ou du texte est donné en premier :

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant 50\ \%$ après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

String matching problem

Étant donnés un texte T de longueur n et un motif P de longueur m sur un alphabet Σ, trouver toutes les occurrences de P dans T

- Énormément d'applications qui nécessitent des solutions différentes selon qui du motif ou du texte est donné en premier :
  - *Online string matching* : motif donné en premier et généralement prétraité

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant$ 50 % après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

String matching problem

Étant donnés un texte T de longueur n et un motif P de longueur m sur un alphabet Σ, trouver toutes les occurrences de P dans T

- Énormément d'applications qui nécessitent des solutions différentes selon qui du motif ou du texte est donné en premier :
  - *Online string matching* : motif donné en premier et généralement prétraité
  - *Offline string matching* : texte donné en premier et généralement indexé

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant$ 50 % après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

String matching problem

Étant donnés un texte T de longueur n et un motif P de longueur m sur un alphabet Σ, trouver toutes les occurrences de P dans T

- Énormément d'applications qui nécessitent des solutions différentes selon qui du motif ou du texte est donné en premier :
  - *Online string matching* : motif donné en premier et généralement prétraité
  - *Offline string matching* : texte donné en premier et généralement indexé

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant 50$ % après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

String matching problem

Étant donnés un texte T de longueur n et un motif P de longueur m sur un alphabet Σ, trouver toutes les occurrences de P dans T

- Énormément d'applications qui nécessitent des solutions différentes selon qui du motif ou du texte est donné en premier :
  - *Online string matching* : motif donné en premier et généralement prétraité
  - *Offline string matching* : texte donné en premier et généralement indexé
- Borne de complexité inférieure : $O(n)$ atteinte par [MP70]

Contexte

- Depuis 1970, plus de 80 algorithmes de recherche exacte de motif ($\geqslant$ 50 % après 2000)
- Idée : faire la liste et comparer expérimentalement les performances

String matching problem

Étant donnés un texte T de longueur n et un motif P de longueur m sur un alphabet Σ, trouver toutes les occurrences de P dans T

- Énormément d'applications qui nécessitent des solutions différentes selon qui du motif ou du texte est donné en premier :
  - *Online string matching* : motif donné en premier et généralement prétraité
  - *Offline string matching* : texte donné en premier et généralement indexé
- Borne de complexité inférieure : $O(n)$ atteinte par [MP70]
  Borne inférieure en moyenne : $O(n \log_{|\Sigma|}(m)/m)$ [Yao79]

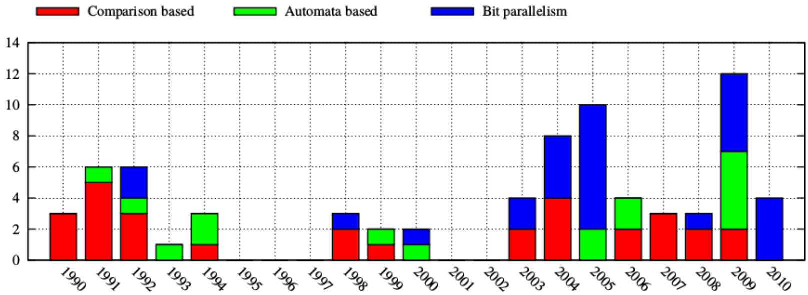
85 algorithmes testés

- Répartis en 3 classes, selon que les algorithmes sont basés sur
  1. les comparaisons de caractères (42)
  2. les automates (17)
  3. le parallélisme de bits (26)

85 algorithmes testés

- Répartis en 3 classes, selon que les algorithmes sont basés sur
  1. les comparaisons de caractères (42)
  2. les automates (17)
  3. le parallélisme de bits (26)

- Répartition en 1990 et 2010 :



Algorithmes basés sur les comparaisons de caractères

Algorithms based on characters comparison

BF	Brute-Force	[CLRS01]	
MP	Morris-Pratt	[MP70]	1970
KMP	Knuth-Morris-Pratt	[KMP77]	1977
BM	Boyer-Moore	[BM77]	1977
HOR	Horspool	[Hor80]	1980
GS	Galil-Seiferas	[GS83]	1983
AG	Apostolico-Giancarlo	[AG86]	1986
KR	Karp-Rabin	[KR87]	1987
ZT	Zhu-Takaoka	[ZT87]	1987
OM	Optimal-Mismatch	[Sun90]	1990
MS	Maximal-Shift	[Sun90]	1990
QS	Quick-Search	[Sun90]	1990
AC	Apostolico-Crochemore	[AC91]	1991
TW	Two-Way	[CP91]	1991
TunBM	Tuned-Boyer-Moore	[HS91]	1991
COL	Colussi	[Col91]	1991
SMITH	Smith	[Smi91]	1991
GG	Galil-Giancarlo	[GG92]	1992
RAITA	Raita	[Rai92]	1992
SMOA	String-Matching on Ordered ALphabet	[Cro92]	1992
NSN	Not-So-Naive	[Han93]	1993
TBM	Turbo-Boyer-Moore	[CCG+94]	1994
RCOL	Reverse-Colussi	[Col94]	1994
SKIP	Skip-Search	[CLP98]	1998
ASKIP	Alpha-Skip-Search	[CLP98]	1998
KMPS	KMP-Skip-Search	[CLP98]	1998
BR	Berry-Ravindran	[BR99]	1999
AKC	Ahmed-Kaykobad-Chowdhury	[AKC03]	2003
FS	Fast-Search	[CF03]	2003

Algorithmes basés sur les automates

Algorithms based on automata

DFA	Deterministic-Finite-Automaton	[CLRS01]	
RF	Reverse-Factor	[Lec92]	1992
SIM	Simon	[Sim93]	1993
TRF	Turbo-Reverse-Factor	[CCG+94]	1994
FDM	Forward-DAWG-Matching	[CR94]	1994
BDM	Backward-DAWG-Matching	[CR94]	1994
BOM	Backward-Oracle-Matching	[ACR99]	1999
DFDM	Double Forward DAWG Matching	[AR00]	2000
WW	Wide Window	[HFS05]	2005
LDM	Linear DAWG Matching	[HFS05]	2005
ILDM1	Improved Linear DAWG Matching	[LWLL06]	2006
ILDM2	Improved Linear DAWG Matching 2	[LWLL06]	2006
EBOM	Extended Backward Oracle Matching	[FL08]	2009
FBOM	Forward Backward Oracle Matching	[FL08]	2009
SEBOM	Simplified Extended Backward Oracle Matching	[FYM09]	2009
SFBOM	Simplified Forward Backward Oracle Matching	[FYM09]	2009
SBDM	Succint Backward DAWG Matching	[Fre09]	2009

Algorithmes basés sur le parallélisme de bits

Algorithms based on bit-parallelism

SO	Shift-Or	[BYR92]	1992
SA	Shift-And	[BYR92]	1992
BNDM	Backward-Nondeterministic-DAWG-Matching	[NR98a]	1998
BNDM-L	BNDM for Long patterns	[NR00]	2000
SBNDM	Simplified BNDM	[PT03,Nav01]	2003
TNDM	Two-Way Nondeterministic DAWG Matching	[PT03]	2003
LBNDM	Long patterns BNDM	[PT03]	2003
SVM	Shift Vector Matching	[PT03]	2003
BNDM2	BNDM with loop-unrolling	[HD05]	2005
SBNDM2	Simplified BNDM with loop-unrolling	[HD05]	2005
BNDM-BMH	BNDM with Horspool Shift	[HD05]	2005
BMH-BNDM	Horspool with BNDM test	[HD05]	2005
FNDM	Forward Nondeterministic DAWG Matching	[HD05]	2005
BWW	Bit parallel Wide Window	[HFS05]	2005
FAOSO	Fast Average Optimal Shift-Or	[FG05]	2005
AOSO	Average Optimal Shift-Or	[FG05]	2005
BLIM	Bit-Parallel Length Invariant Matcher	[Kül08]	2008
FSBNDM	Forward SBNDM	[FL08]	2009
BNDMq	BNDM with q-grams	[DHPT09]	2009
SBNDMq	Simplified BNDM with q-grams	[DHPT09]	2009
UFNDMq	Shift-Or with q-grams	[DHPT09]	2009
SABP	Small Alphabet Bit-Parallel	[ZZMY09]	2009
BP2WW	Bit-Parallel^2 Wide-Window	[CFG10a]	2010
BPWW2	Bit-Parallel Wide-Window^2	[CFG10a]	2010
KBNDM	Factorized BNDM	[CFG10b]	2010
KSA	Factorized Shift-And	[CFG10b]	2010

Jeux de donnée pour les tests

- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur

Jeux de donnée pour les tests

- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur
- 12 textes
  - 8 Randσ pour $\sigma \in [2, 4, 8, 16, 32, 64, 128, 256]$, où chaque Rand$\sigma$ est un texte de 5*Mb* sur un alphabet de taille σ, avec distribution uniforme des caractères

Jeux de donnée pour les tests

- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur
- 12 textes
  - 8 Randσ pour $\sigma \in [2, 4, 8, 16, 32, 64, 128, 256]$, où chaque Rand$\sigma$ est un texte de 5*Mb* sur un alphabet de taille σ, avec distribution uniforme des caractères
  - une séquence de génome de 4 638 690 pb d'*Escherichia coli* ($\sigma = 4$)

Jeux de donnée pour les tests

- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur
- 12 textes
  - 8 Randσ pour $\sigma \in [2, 4, 8, 16, 32, 64, 128, 256]$, où chaque Rand$\sigma$ est un texte de $5Mb$ sur un alphabet de taille σ, avec distribution uniforme des caractères
  - une séquence de génome de 4 638 690 pb d'*Escherichia coli* ($\sigma = 4$)
  - protéines de *Saccharomyces cerevisiae* de lg 3 295 751 ($\sigma = 20$)

Jeux de donnée pour les tests

- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur
- 12 textes
  - 8 Randσ pour $\sigma \in [2, 4, 8, 16, 32, 64, 128, 256]$, où chaque Rand$\sigma$ est un texte de $5Mb$ sur un alphabet de taille σ, avec distribution uniforme des caractères
  - une séquence de génome de 4 638 690 pb d'*Escherichia coli* ($\sigma = 4$)
  - protéines de *Saccharomyces cerevisiae* de lg 3 295 751 ($\sigma = 20$)
  - une version de la bible de 4 047 392 caractères ($\sigma = 63$)

Jeux de donnée pour les tests

- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur

- 12 textes
  - 8 Randσ pour $\sigma \in [2, 4, 8, 16, 32, 64, 128, 256]$, où chaque Rand$\sigma$ est un texte de 5*Mb* sur un alphabet de taille σ, avec distribution uniforme des caractères
  - une séquence de génome de 4 638 690 pb d'*Escherichia coli* ($\sigma = 4$)
  - protéines de *Saccharomyces cerevisiae* de lg 3 295 751 ($\sigma = 20$)
  - une version de la bible de 4 047 392 caractères ($\sigma = 63$)
  - le fichier world192.txt (The CIA World Fact Book) de 2 473 400 caractères ($\sigma = 94$)

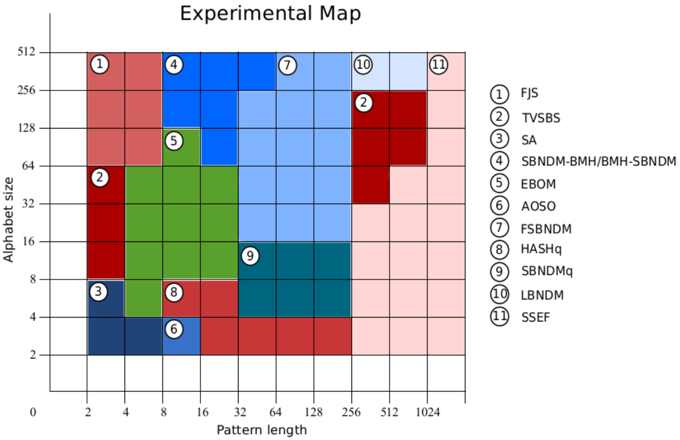
Jeux de donnée pour les tests

- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur
- 12 textes
  - 8 Randσ pour $\sigma \in [2, 4, 8, 16, 32, 64, 128, 256]$, où chaque Rand$\sigma$ est un texte de 5*Mb* sur un alphabet de taille σ, avec distribution uniforme des caractères
  - une séquence de génome de 4 638 690 pb d'*Escherichia coli* ($\sigma = 4$)
  - protéines de *Saccharomyces cerevisiae* de lg 3 295 751 ($\sigma = 20$)
  - une version de la bible de 4 047 392 caractères ($\sigma = 63$)
  - le fichier world192.txt (The CIA World Fact Book) de 2 473 400 caractères ($\sigma = 94$)
- Pour chaque texte, un ensemble de 400 motifs de taille m extraits aléatoirement des textes, avec $m \in [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]$

Jeux de donnée pour les tests

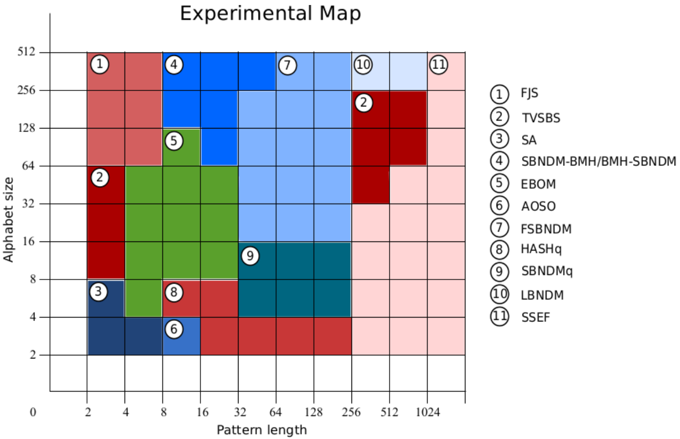
- Tous les algorithmes ré-implémentés en `C` et testés sur le même ordinateur
- 12 textes
  - 8 Randσ pour $\sigma \in [2, 4, 8, 16, 32, 64, 128, 256]$, où chaque Rand$\sigma$ est un texte de $5Mb$ sur un alphabet de taille σ, avec distribution uniforme des caractères
  - une séquence de génome de 4 638 690 pb d'*Escherichia coli* ($\sigma = 4$)
  - protéines de *Saccharomyces cerevisiae* de lg 3 295 751 ($\sigma = 20$)
  - une version de la bible de 4 047 392 caractères ($\sigma = 63$)
  - le fichier world192.txt (The CIA World Fact Book) de 2 473 400 caractères ($\sigma = 94$)
- Pour chaque texte, un ensemble de 400 motifs de taille m extraits aléatoirement des textes, avec $m \in [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]$
- Pour chaque ensemble de motifs, 400 runs pour calculer le temps moyen

Résultats



Experimental Map
512
256
128
64
32
16
8
4
2
0
2
4
8
16
32
64
128
256
512
1024
Alphabet size
Pattern length
1 FJS
2 TVSBS
3 SA
4 SBNDM-BMH/BMH-SBNDM
5 EBOM
6 AOSO
7 FSBNDM
8 HASHq
9 SBNDMq
10 LBNDM
11 SSEF

Résultats



- Les performances dépendent des tailles d'alphabet et des longueurs de motif