Recherche de motifs

Problème : rechercher un mot dans un texte.

Applications :

- éditeur de textes
- génétique (séquence d'ADN)
- WEB (moteur de recherche)
- ...

Vocabulaire :

• un alphabet Σ,	$\{A, C, G, T\}$, $\{0, 1\}$, $\{a, ..., z\}$, ...
• un caractère $c \in \Sigma$	A
• une chaîne de caractères $T \in \Sigma^*$	$ATTTGCCATTA$
• un préfixe de T,	ATT
• un suffixe de T,	$CCATTA$
• ϵ le mot vide	

Définition

Un motif $P[1..m]$ apparait dans le texte $T[1..n]$ avec un décalage s si $T[s+1..s+m] = P[1..m]$.

Pour $s = 2$

$$
\begin{array}{c|c|c|c|c|c|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\cline{2-7}
T & a & b & b & a & b & a \\
\cline{2-7}
\multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{=} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} \\
\cline{4-6}
P & \multicolumn{1}{c}{} & \multicolumn{1}{c|}{} & b & a & b & \multicolumn{1}{c}{} \\
\cline{4-6}
\end{array}
$$

Algorithme naïf

Idée : Vérifier $P[1..m] = T[s+1..s+m]$ pour les $n - m + 1$ décalages possibles.

Recherche Naïve(T, P)

$n \leftarrow longueur[T]$
$m \leftarrow longueur[P]$
pour $s \leftarrow 0$ **à** $n - m$
    faire si $P[1..m] = T[s+1..s+m]$
        alors imprimer “Le motif apparait avec un décalage” s

$s = 0$: échec

a	b	b	a	b	a
$\times$					
b	a	b			

$s = 1$: échec

a	b	b	a	b	a
	$\times$				
	b	a	b		

$s = 2$: succès !

a	b	b	a	b	a
		$=$			
		b	a	b	

Compléxité : $O((n - m + 1) \times m)$

Algorithme de Rabin Karp (1)

Un algorithme utilisant l'arithmétique modulaire.

Même complexité que celle de l'algorithme naïf, plus efficace en pratique.

Notations

On suppose $\Sigma = \{0, 1, 2, ..., 9\}$.

On associe à la chaîne de caractères ‘‘19748’’ l'entier $19\,748$.

p l'entier associé au motif $P[1..m]$

t_s l'entier associé au motif $T[s+1..s+m]$.

Algorithme de Rabin Karp (2)

Supposons momentanément que p et t_s ne sont pas trop grands.

On peut calculer p en $O(m)$ en utilisant le schéma de Horner.

$$p = P[m] + 10(P[m-1] + 10(P[m-2] + ... + 10(P[2] + 10P[1])...)$$

La valeur t_0 peut être calculée en $O(m)$ de la même façon.

Pour calculer $t_1, ..., t_{n-m}$ en $O(n-m)$, connaissant t_s on calcule t_{s+1} en $O(1)$:

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1]. \qquad (1)$$

Algorithme de Rabin Karp (3)

Difficulté. p et t_s sont des grands nombres (à m chiffres).

Solution. on fait toutes les opérations modulo q, avec q tel que qd (en base $d = |\Sigma|$) tienne dans un seul mot machine.

On met à jour (1) pour qu'elle soit valable modulo q :

$$t_{s+1} = (d(t_s - hT[s+1]) + T[s+m+1]) \bmod q). \qquad (2)$$

où $h = d^{m-1} \bmod q$.

Problème. Il peut y avoir des *coups manqués*

$$p \equiv t_s (\bmod q) \text{ sans que } p = t_s.$$

L'efficacité pratique de la méthode repose sur le fait que le nombre de coups manqués est "petit".

Complexité. dans le pire des cas $O(m(n-m+1))$.

Algorithme de Rabin Karp (4)

Rabin-Karp(T, P, d, q)

$n \leftarrow longueur[T]$
$m \leftarrow longueur[P]$
$h \leftarrow d^{m-1} \mod q$
$p \leftarrow 0$
$t_0 \leftarrow 0$
pour $i \leftarrow 1$ **à** m
  faire $p \leftarrow (dp + P[i]) \mod q$
    $t_0 \leftarrow (dt_0 + T[i]) \mod q$
pour $s \leftarrow 0$ **à** $n - m$
  faire si $p = t_s$
    alors si $P[1..m] = T[s+1..s+m]$
      alors imprimer "Le motif apparaît avec le décalage" s
    si $s < n - m$
      alors $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1] \mod q$

Automate fini (1)

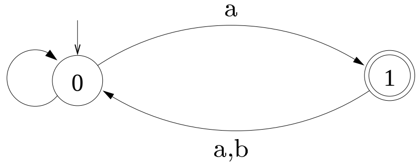
Un *automate fini* M est un quintuplet $(Q, q_0, A, \Sigma, \delta)$, où

- Q est un ensemble fini *d'états*,
- $q_0 \in Q$ est *l'état initial*,
- $A \subseteq Q$ est l'ensemble des *états terminaux*,
- Σ est un *alphabet* fini,
- δ est une fonction de $Q \times \Sigma$ vers Q, appelée *fonction de transition* de M.

Exemple.

$\Sigma = \{a, b\}$, $Q = \{0, 1\}$, $q_0 = 0$, $A = \{1\}$, δ est donné par

	a	b
0	1	0
1	0	0



Automate fini (2)

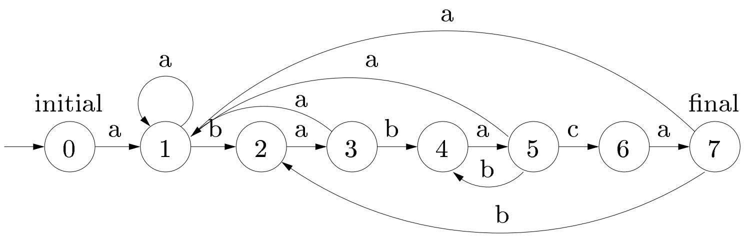
Fonction d'état final $\phi : \Sigma^* \to Q$ définie par : $\begin{cases} \phi(\varepsilon) = q_0 \\ \phi(Ta) = \delta(\phi(T), a) \end{cases}$

On dit que T est reconnu par l'automate M si $\phi(T) \in Q$.

Le *langage reconnu par un automate* est constitué de l'ensemble des chaînes reconnues par cet automate.

Idée. Construire l'automate qui reconnaît le langage formé des chaînes qui se termine par un motif P donné.

Exemple. P = `ababaca`



Automate fini (3)

Recherche-Automate-Fini(T, P, δ, m)

$n \leftarrow$ *longueur*$[T]$
$m \leftarrow$ *longueur*$[P]$
$q \leftarrow 0$
    pour $i \leftarrow 1$ **à** n
        faire $q \leftarrow \delta(q, T[i])$
            si $q = m$
                alors $s \leftarrow i - m$
                    imprimer "Le motif apparait avec le décalage"
s

Complexité. Calcul de la fonction de transition : $O(m^3|\Sigma|)$.

Recherche du motif : $O(n)$.

Algorithme de Knuth-Morris-Pratt (1)

Idée : Faire un prétraitement sur le motif en $O(m)$ pour pouvoir calculer la fonction de transition “à la volée”.

Prétraitement : calcul de la fonction préfixe $\pi : \{1, ..., m\} \longrightarrow \{0, ..., m-1\}$

$$\pi[q] = \max\{k : k < q \text{ et } P[1..k] \text{ est un préfixe de } P[1..q]\ \}$$

$\pi[q]$ est la longueur du plus long préfixe de P qui est aussi suffixe de $P[1..q]$.

Exemple : `ababababca`

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

Algorithme de Knuth-Morris-Pratt (2)

KMP(T, P)

$n \leftarrow$ *longueur*$[T]$
$m \leftarrow$ *longueur*$[P]$
$q \leftarrow 0$
pour $i \leftarrow 1$ **à** n
    faire tant que $q > 0$ et $P[q+1] \neq T[i]$
        faire $q \leftarrow \pi[q]$
      si $P[q+1] = T[i]$
        alors $q \leftarrow q + 1$
      si $q = m$
        alors imprimer “Le motif apparaît avec le décalage” $i - m$
          $q \leftarrow \pi[q]$

Complexité. $O(n)$ (en utilisant une *analyse amortie*)

Algorithme de Knuth-Morris-Pratt (3)

Calcul-Fonction-Préfixe(P)

$m \leftarrow longueur[P]$
$\pi[1] \leftarrow 0$
$k \leftarrow 0$
pour $q \leftarrow 2$ **à** m
    faire tant que $k > 0$ et $P[k+1] \neq P[q]$
        faire $k \leftarrow \pi[k]$
    si $P[k+1] = P[q]$
        alors $k \leftarrow k+1$
    $\pi[q] \leftarrow k$
retourner π

Complexité. $O(m)$

Algorithme de Boyer-Moore (1)

Le plus efficace lorsque P est long et Σ n'est pas trop grand.

Boyer-Moore(T, P, Σ)

$n \leftarrow longueur[T]$
$m \leftarrow longueur[P]$
$\lambda \leftarrow$ **Calcul-Dernière-Occurence**(P, m, Σ)
$\gamma \leftarrow$ **Calcul-Bon-Suffixe**(P, m)
$s \leftarrow 0$
tant que $s \leq n - m$
    faire $j \leftarrow m$
        tant que $j > 0$ et $P[j] = T[s+j]$
            faire $j \leftarrow j - 1$
        si $j = 0$
            alors imprimer "Le motif apparait au décalage " s
                $s \leftarrow s + \gamma[0]$
            sinon $s \leftarrow s + \max(\gamma[j], j - \lambda[T[s+j]])$

Algorithme de Boyer-Moore (2)

$\lambda[a]$ est la position de l'occurence de a la plus à droite dans le motif P.

Dernière-occurence(P, m, Σ)

pour chaque caractère $a \in \Sigma$
    faire $\lambda[a] \leftarrow 0$
pour $j \leftarrow 1$ à m
    faire $\lambda[P[j]] \leftarrow j$
retourner λ

Compléxité. $O(|\Sigma| + m)$

Algorithme de Boyer-Moore (3)

$\gamma[j]$ est la plus petite quantité par laquelle on peut avancer s en préservant l'appariement du "bon suffixe" $T[s+j+1..s+m]$.

$$\gamma[j] = \min\left(\{m - \pi[m]\} \cup \{l - \pi'[l] : 1 \leq l \leq m \text{ et } j = m - \pi'[l]\}\right)$$

Calcul-bon-suffixe(P, m) $\rightarrow$ *Complexité* $O(m)$

$\pi \leftarrow$ **Calcul-fonction-préfixe**(P)
$P' \leftarrow$ mirroir(P)
$\pi \leftarrow$ **Calcul-fonction-préfixe**(P')
pour $j \leftarrow 0$ **à** m
    $\gamma[j] \leftarrow m - \pi[m]$
pour $l \leftarrow 1$ **à** m
    faire $j \leftarrow m - \pi'[l]$
        si $\gamma[j] > l - \pi'[l]$
            alors $\gamma[j] \leftarrow l - \pi'[m]$
retourner γ