Algorithmique des mots.

Adrien Friggeri

16 mai 2007

1 Un problème classique : la recherche de motifs

Problème Etant donné un alphabet Σ et $P, T \in \Sigma^*$, trouver toutes les occurences de P (motif) comme facteur T (texte).

Il existe un certain nombre de variantes :

- rechercher l'existence d'une occurence
- utilisation de joker (expression régulières)
- recherche de sous séquence plutôt que de facteur (programmation dynamique)
- ...

Dans ce cadre, on considère l'opération de base suivante : test d'égalité entre lettres $O(1)$. Les entrées sont sous formes de deux tableaux $T[1 \dots n]$, $P[1 \dots m]$.

2 Algorithme naïf

Algorithme 1 Algorithme naïf de recherche de motif

Entrées: $T[1 \dots n]$
Entrées: $P[1 \dots m]$
  $n :=$ longueur(T)
  $m :=$ longueur(P)
  Pour s de 0 à $n - m$ **Faire**
    Si $P[1 \dots m] = S[s + 1 \dots s + m]$ **Alors**
      "P apparaît dans T avec un décalage s"
    FinSi
  FinPour

Complexité : $O(m(n - m))$.

3 Algorithme de Rabin-Karp

3.1 Principe

Lire les lettres commes des chiffres et les mots comme des nombres en bas $|\Sigma|$ et faire des tests d'égalité entre les entiers.

ex. Si $|\Sigma| = 8$, $\Sigma = \{0, 1, 2, \ldots, 7\}$: lire le mot 3252 comme $\overline{3252}^8$.

On note donc ici p l'entier correspondant à $P[1 \ldots m]$ et $\forall 0 \leq s \leq n-m$, t_s celui correspondant à $T[s+1 \ldots s+m]$. Alors $P[1 \ldots m] = T[s+1 \ldots s+m] ssi p = t_s$.

3.2 Calcul de p et des t_s

. $\Sigma = \{0, 1, \ldots, |\Sigma| - 1\}$.

- $p =_{Horner} P[m] + |\Sigma|(p[m-1] + |\Sigma|(\cdots + |\Sigma| p[1]) \ldots)$
  complexité : $O(m)$ opérations + et $\times$
- t_0 idem que p avec T.
  complexité : $O(m)$ opérations + et $\times$
- $t_{s+1} = (t_s - |\Sigma|^{m-1} T[s+1])|\Sigma| + T[s+m+1]$.
  complexité : $O(1)$ opérations + et $\times$ (pré calcul de $|\Sigma|^{m-1}$)

3.3 Problème : taille des entiers multipliés trop grandes

Problème Il peut arriver que les entiers multipliés soient trop grands, auquel cas la multiplication n'est plus en $O(1)$.

On note $l = \log_2 p \simeq m \log_2 |\Sigma|$ la longueur de p. Mais même avec un bon algorithme de multiplication comme la FFT (transformée de fourier rapide) on obtient une complexité en $O(l \log_2 l) = O(m \log_2 |\Sigma| (m \log_2 |\Sigma|))$

Remède Travailler modulo q avec q bien choisi, par exemple qui tient sur un mot mémoire. Dans ce cas, la formule devient :

$$t_{s+1} = |\Sigma|(t_s - hT[s+1] + T[s+m+1] \mod q$$

avec $h = |\Sigma|^{m-1} \mod q$.

Inconvénients

$$p \equiv t_s \mod q \not\Rightarrow p \equiv t_s$$
$$p \not\equiv t_s \Rightarrow p \not\equiv t_s \mod q$$

Dans ce cas là, quand $p \equiv t_s \mod q$, on fait le test $P[1 \ldots m] = T[s+1 \ldots s+m]$. La complexité dans le pire des cas reste la même que pour l'algorithme naïf : $O(m(n-m))$ (quand $\forall s, p \equiv t_s \mod q, P = a^m, T = a^n$).

3.4 Technique style reste chinois

On utilise une méthode similaire à celle ci dessus en utilisant plusieurs nombres premiers entre eux.

On note M un majorant des entiers manipulés (par exemple $M = |\Sigma|^m$). Comme $p \equiv t_s \Leftrightarrow t_s \equiv p \mod M$, on prend $p_1, \ldots, p_k$ nombres premiers (entre eux) tels que $p_1 \times \cdots \times p_k \geq M$. Alors $t_s \equiv p \mod p_1 \times \cdots \times p_k \Leftrightarrow \forall i, t_s = p \mod p_i$.

Dans ce cas, $k \simeq \log M \simeq m$, donc on effecture m calculs en parallèle. Il n'y a donc pas de gain évident par rapport à l'algorithme naïf pour la complexité dans le pire des cas, mais parfois de très bons résultats en pratique. Par ailleurs, c'est un principe qui peut s'appliquer à la recherche pour d'autres structures que les mots.