ALGORITHME DE RABIN-KARP

Cormen 3ème éd p 910.

Pb Il s'agit encore du problème de la recherche de toutes les occurences d'un motif $x = x_1 \text{ — } x_m$ dans un texte $t_1 \text{ — } t_n = t$.
On renseigne ces occurences par leur indice de début.

Idée L'idée est de voir nos lettres comme des chiffres.
Ainsi au lieu de comparer des mots on compare des nombres.

Codage / Décodage

Soit $\Sigma = [0..b[$ où $b \in \mathbb{N}^*$

Puisqu'on a coutume de noter $\hookleftarrow_b$ le codage d'un entier en chiffres selon la base b, on notera ici $>\!\!\multimap_b$ le décodage d'un mot de Σ (une suite de chiffres) selon la base b.

$$>\omega_1 \omega_2 \text{ — } \omega_n <_b := \omega_1 \times b^{n-1} + \omega_2 \times b^{n-2} \dots + \omega_{n-1} \times b + \omega_n$$

← poids fort → poids faible

$$= \sum_{i=1}^{n} \omega_i \, b^{n-i}$$

$$= P(b) \text{ où } P = \sum_{i=1}^{n} \omega_i \, X^{n-i}$$

$$= \text{EVALUER}\,(\omega_n, \omega_{n-1}, \omega_{n-2} \dots \omega_1), b)$$

$$= (((\omega_1 \cdot b) + \omega_2) \times b \dots) \times b + \omega_n$$

] cf Horner

décoder($\overbrace{\omega_1 \text{ — } \omega_n}^{\omega}$, b)

```
n ← |ω|
v ← ω1
pour i allant de 2 à n
    v ← v × b
    v ← v + ωi
retourner v.
```

NB: l'algo ci-contre n'est pas utile tel quel mais permet de mieux comprendre les 1ères boucles de l'algo de Rabin-Karp.

ex $\Sigma = [0,1,2,3]$ $b = 4$ $>123< = 3 + 2\times4 + 1\times16 = 27$ // calcul à la main
$= (((1\times4)+2)\times4)+3 = (6\times4)+3 = 27$ // Horner.

Dans notre problème on va vouloir non seulement calculer $>x<_b$ mais aussi $>w<_b$ pour tout les facteurs w de t de taille m.

Mais du calcul de $>w_i \dots w_{i+m-1}<$ on peut déduire des infos sur $>w_{i+1} \dots w_{i+m}<$ le facteur suivant

ex $b = 10$, $m = 4$

$>1234<_b = 1234$

$>2345<_b = 234 \times 10 + 5$

$= (>1234<_b - 1 \times 10^3) \times 10 + 5$

$= (>1234<_b - 1 \times b^{m-1}) \times b + 5$

Pté Considérons $(m, b) \in \mathbb{N}^*$ fixés, et $\Sigma = [0 .. (b-1)]$.

Si $w \in \Sigma^m$, $g \in \Sigma$ et $d \in \Sigma$ alors

$$>wd<_b = (>gw<_b - g \times b^{m-1}) + d \quad \star$$

Réduction modulo q

L'amélioration par rapport à l'algorithme naïf doit être qu'on puisse comparer d'un coût d'un seul, le contenu d'une fenêtre de taille m et le motif, en comparant les nombres associés. Actuellement on considère la comparaison de nombre comme une opération constante. Or le coût de cette opération dépend de la taille des nombres en réalité et ici nos nombres sont bornés par $b^m - 1$, ce qui n'est pas cst !!!

En fixant un entier premier q, et en réalisant toutes nos opérations modulo q, nos nombres seront bornés par $q-1$, il est alors justifié de considérer constante la comparaison de nombre, ainsi que les addi° et multiplica°.

Mais ⚠ aux faux positifs. En effet

$>x<_b \not\equiv >t_i \dots t_{i+m-1}<_b \ [q] \Rightarrow x \neq t_i \dots t_{i+m-1}$

mais $>x<_b \equiv >t_i \dots t_{i+m-1}<_b \ [q] \not\Rightarrow x = t_i \dots t_{i+m-1}$.

D'où une vérification (V) dans l'algorithme qui suit.

EN MOYENNE

On fait l'hypothèse que la réduction modulo q du décodage en base b agit comme une transformation aléatoire uniforme entre Σ^* et $\mathbb{Z}/q\mathbb{Z}$.

Supposons le motif x fixé, et T un texte aléatoire sur Σ de n lettres, suivant une loi uniforme.

Autrement dit $\forall \omega \in \Sigma^n \quad \mathbb{P}(T=\omega) = 1/b^n$

et même $\forall i \in [1..n], \forall a \in \Sigma \quad \mathbb{P}(T_i = a) = 1/b$.

Puisque les T_i sont unif, $T_i \dots T_{i+m+1}$ est unif dans Σ^{m}.

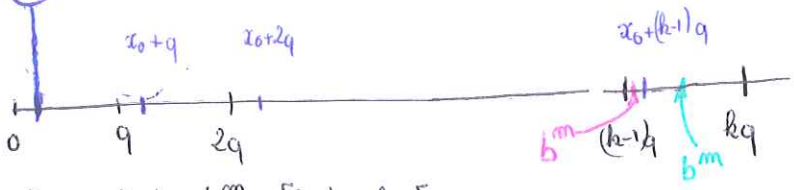
Donc $$\mathbb{P}(\text{>}T_i \dots T_{i+m+1}\text{<}_b \equiv \text{>}x\text{<}_b\ [q]) = \frac{\#\{\omega \in \Sigma^m \mid \text{>}\omega\text{<}_b \equiv \text{>}x\text{<}\ [q]\}}{\#\{\omega \in \Sigma^m\}}$$

$$= \frac{\#\{s \in [0..b^m-1] \mid s \equiv \text{>}x\text{<}_b\ [q]\}}{b^m}$$

$\approx \frac{k \text{ ou } k-1}{b^m}$ où $k-1 = \lfloor b^m/q \rfloor$

$\approx 1/q$

$x_0 \in [0,q[$ tq $x_0 \equiv \text{>}x\text{<}_b\ [q]$.



Fixons k tq $b^m \in [(k-1)q, kq[$

Selon que b^m se situe "sous" $x_0+(k-1)q$ ou au delà de $x_0+(k-1)q$, il y a $k-1$ ou k s entre 0 et b^m-1 tels que $s \equiv \text{>}x\text{<}_b\ [q]$ ie tq $s \in x_0 + \mathbb{Z}$.

On cherche le nombre de fois (nombre moyen) où l'on rentre dans la boucle implicite de **(V)** car elle est en $O(m)$.

Autrement dit on cherche le nombre de tests "$v_t = v_x$" positifs, soit

$$\mathbb{E}\left[\sum_{i=1}^{n-m+1} \mathbb{1}_{(v_t = v_x)}\right] = \sum_{i=1}^{n-m+1} \mathbb{E}\left[\mathbb{1}_{\text{>}T_i \dots T_{i+m-1}\text{<}_b \equiv \text{>}x\text{<}_b\ [q]}\right]$$

$$= \sum_{i=1}^{n-m+1} \mathbb{P}\ (\text{>}T_i \dots T_{i+m-1}\text{<}_b \equiv \text{>}x\text{<}_b\ [q])$$

$$\approx (n-m+1) \times \frac{1}{q}.$$

D'où une complexité moyenne en $O\Big(\underbrace{n-m+1}_{\substack{\text{test négatif ou positif} \\ \text{(coût du test)}}} + \underbrace{\frac{(n-m+1)}{q} \times m}_{\substack{\text{test positif} \\ \text{et donc vérification (V)} \\ \text{en } O(m)}}\Big)$ pour la boucle B_2.

Finalement la complexité moyenne est en

$$O\left(m + (n-m+1)\left(1+\frac{m}{q}\right)\right) = O\left(m + \frac{nm}{q}\right)$$

Algo

Considérons Σ et donc b fixés. Fixons aussi q (év. choisi selon b)

Rabin_Karp (x, t)

$n \leftarrow$ longueur (t)
$m \leftarrow$ ——— (x)

$h \leftarrow b^{m-1} \mod (q)$

$v_x \leftarrow x_1$
$v_t \leftarrow t_1$

Pour i allant de 2 à m — **B_1**
- $v_x \leftarrow v_x \otimes b \mod [q]$
- $v_x \leftarrow v_x \oplus x_i \mod [q]$
- $v_t \leftarrow v_t \otimes b \mod [q]$
- $v_t \leftarrow v_t \oplus t_i \mod [q]$

// $v_x = > x <_b$ et $v_t = > t_1 - t_m <_b$

Pour i allant de 1 à $n-m+1$ — **B_2**
- Si $v_t = v_x$
  - alors si t_i ——— $t_{i+m-1} = x$) (V)
    - alors afficher i
- Si $i < n-m+1$
  - alors $v_t \leftarrow (v_t - t_i \times h) + t_{i+m}$ // applica° de ★

Complexité

AU PIRE

B_1 se fait toujours en $O(m)$

B_2 se fait au pire en $O(\underline{n-m+1} \times \underline{m})$

puisqu'au pire chaque indice est un début possible de motif et l'on doit faire la vérifica° (V) en $O(m)$

ex $x = a^m \quad t = a^n$

La complexité au pire est en $\underline{O((n-m+1) \times m)}$

soit identique à celle de l'algorithme naïf.

esc

$\Sigma = \{0,1,2,3,4,5,6\}$ $\quad b = 7$.

x	17
0	0
1	17
2	34
3	51
4	68
5	85

$x = 116$

$t = 425116$

Au plus un motif de taille 3 vaut

$$>666<_7 = 6\times7^2 + 6\times7^1 + 6\times7^0$$
$$= 6\times49 + 42 + 6$$
$$= 300 - 6 + 42 + 6$$
$$= 342$$

342 ne tient pas sur un octet (qui vaut au + $2^8 - 1 = 127$)

Si on choisit $q = 17$ on a $qb = 119 \leq 127$ donc qb tient sur un octet.

$>116<_7 \equiv 1\times49 + 1\times7 + 6\times1 \equiv 62 \equiv 11\ [17]$ à la main

$\equiv (((1\times7)+1)\times7)+6$ dans l'algo via Horner.

$7;\ 7+1 \equiv 8\ [17];\ 8\times7 = 56 \equiv 5\ [17];\ 5+6 \equiv 11\ [17]$

$>425<_7$: $4\times7 \equiv 28 \equiv 11\ [17]$
$11+2 \equiv 13 \equiv -4\ [17]$
$7\times-4 \equiv -28 \equiv -11 \equiv 6\ [17]$
$6+5 \equiv 11\ [17]$

Donc $>425<_7 \equiv >116<_7\ [17]$
il faut tester mais $4 \neq 1$
donc on n'a pas le motif

$>251<_7$ $\quad 2\times7 \equiv 14 \equiv -3\ [17]$
$-3+5 \equiv +2\ [17]$
$2\times7 \equiv 14 \equiv -3\ [17]$
$-3+1 \equiv -2\ [17] \neq 11$ Donc $>251<_7 \not\equiv >116<_7\ [17]$

$>511<_7$: $5\times7 \equiv 35 \equiv 1\ [17]$
$1+1 \equiv 2\ [17]$
$2\times7 \equiv 14 \equiv -3\ [17]$
$-3+1 \equiv -2\ [17] \neq 11$

Donc $>511<_7 \not\equiv >116<_7\ [17]$

$>116<_7$: $1\times7 \equiv 7\ [17]$
$7+1 \equiv 8\ [17]$
$8\times7 \equiv 56 \equiv 5\ [17]$
$5+6 \equiv 11\ [17]$

$t_4 t_5 t_6 \quad x$

On a $>\widehat{116}<_7 \equiv >\widehat{116}<_7\ [17]$.

On teste et on a bien $t_4 = x_1$, $t_5 = x_2$, $t_6 = x_3$. On a trouvé !