ALGORITHMES DE RECHERCHE DE MOTIFS (SUR LES MOTS)

INTRODUCTION

Problématique

La problématique est la suivante : on a une chaine de caractère C : $0110011000101010110\ldots$ et un motif $M = 010$. On recherche les occurrences de M dans la chaine C :

$$C = 011001100\mathbf{0101010}110\ldots$$

Notations

On appellera Σ dans la suite un alphabet fini (ensemble fini de caractères).
On appellera Σ^k l'ensemble des mots de longueur k sur Σ.
On appellera Σ^* :

$$\Sigma^* = \bigcup_{k \geq 0} \Sigma^k.$$

Soit $w \in \Sigma^*$, on appellera $len(w)$ la longueur de w. Soit $i \leq \mathrm{len}(w)$, alors $w[i]$ sera le i-ième caractère de w. $w[i : j] = w[i]w[i+1]\ldots w[j-1]$.

1. ALGORITHME NAÏF

1.1. Présentation

On se donne une chaîne C et un motif M.
On teste l'égalité de M avec chaque sous-mot $C[i : i + \mathrm{len}(M)]$.

> Définition 1.1.0.1. —
> Lorsque $M = C[i : i + \mathrm{len}(M)]$ on dit que M apparaît avec décalage i dans la chaîne C.

```
function rechercheMotifNaif(C,M)
        n = len(C)
        m = len(M)
        S = []
        for i from 0 to n - m do
                if M[0:m] = C[i:i + m] then
                        S = S + [i]
        return S
```

1.2. Coût de l'algorithme naïf

On cherche à compter le nombre de comparaisons de caractères, $C(n)$.

Chaque test $M = C[i : i+m]$ a un coût au plus de m comparaisons. De plus, il y a $n - m + 1$ tests d'égalité donc :

$$C(n) = \mathrm{O}(m(n-m+1)).$$

De plus la borne est atteinte pour : $M = a^m$ avec $a \in \Sigma$ et $C = a^n$ avec $n \geq m$.

2. ALGORITHME DE KARP-RABIN

2.1. L'algorithme

R. Karp, M. Rabin : 1987.

C'est un algorithme qui se base sur le naïf mais enrichi par une technique de *hachage*.

Dans la suite, $\Sigma = \{0, 1, \ldots, 9\}$. On peut donc représenter des mots par des entiers. Le mot $M \in \Sigma^*$ peut être représenté par l'entier :

$$m = \sum_{k=0}^{\mathrm{len}(M)} M[k] 10^{\mathrm{len}(M)-k}.$$

Coût. — Le coût de la représentation est un $\mathrm{O}(m)$ [1§].

Exemple. — Par exemple, si C = '23451' et $\mathrm{len}(M) = 4$. On a $C[0:4] = C[0]C[1]C[2]C[3]$ = '2345' et sa représentation est $c_0 = 2345$. On veut obtenir l'entier c_1 représentant $C[1:5]$ à partir de c_0 et $C[4]$: $c_1 = 10(c_0 - 10^3 \times 2) + 1 = 10(c_0 - 10^3 C[0]) + C[4]$.

Ainsi, on peut calculer 10^{m-1} en $\mathrm{O}(\log m)$ opérations (mais $\mathrm{O}(m)$ suffit). On obtient c_{i+1} à partir de c_i par :

$$c_{i+1} = d \times (c_0 - d^{m-1} \times C[i]) + C[i+m],$$

où $d = |\Sigma|$. Donc on obtient c_{i+1} à partir de c_i en un nombre constant d'opérations.

1§. Voir : schéma de Horner.

Conclusion. — L'ensemble des $c_0, c_1, \dots, c_{n-m+1}$ peut être obtenu en $\mathrm{O}(m+n-m+1) = \mathrm{O}(n)$.

2.2. Représentation « hachée »

Soit $q \in \mathbf{N}$ assez grand. Il s'agit de calculer

$$M, C_0, C_1, \dots \mod q.$$

On pose

$$\begin{aligned}
\tilde{M} &= M \mod q \\
\tilde{C}_0 &= C_0 \mod q \\
\tilde{C}_{i+1} &= d^{(2\S)}(\tilde{C}_i - d^{l-1}C[i]) + C[i+l] \mod q.
\end{aligned}$$

Remarque. — Il faut $\Theta(l_m)$ pour construire $\tilde{M}, \tilde{C}_0$. Finalement, il faut $\mathrm{O}(l_c - l_m + 1)$ pour construire l'ensemble des $\tilde{c}_2, \tilde{c}_2, \dots$.

> Proposition 2.2.0.1. —
> Si $\tilde{C}_i \neq \tilde{M}$ alors $C_i \neq M$.

On peut utiliser cette propriété comme filtre pour ne pas effectuer tous les tests.

```
function KarpRabin(C, M, d, q)
        lc = len(C)
        lm = len(M)
        h = d**(lm-1) % q
        m, c = 0, 0
        S = []
        for i from 0 to lm - 1 do
                m = (d*m + M[i]) % q
                c = (d*c + C[i]) % q
        for i from 0 to lc - lm do
                if m = c then
                        if M[0:lm] = C[i:i+lm] then
                                S = S + [i,]
                if i < lc - lm then
                        c = (d*(c-C[i]*h) + C[i+lm]) % q
        return S
```

2§. Ici $d = 10$ car on est en base décimale.

2.3. Étude de l'efficacité

Supposons que x est le nombre d'apparitions de M dans C. Pour toutes les occurrences de M dans C on doit faire $x * l_m$ comparaisons.

De plus, le nombre de faux-positifs est borné par $\mathrm{O}(l_c/q)$.

Ainsi, le nombre d'opérations est $\mathrm{O}(l_c) + \mathrm{O}(l_m(x + l_m/q))$. Si on prend $q \geq l_m$ alors le nombre d'opérations est en $\mathrm{O}(l_c)$.