Chapitre 11 : Algorithmique du texte

Table des matières

1 Recherche dans un texte

1.1 Introduction

1.1.1 Problème

Étant donné un texte (un fichier / une chaîne de caractères) t, et un motif (une chaîne de caractères) x, on veut trouver toutes les occurrences de x dans t (*i.e* on veut les positions (indices)).

1.1.2 Algorithme naïf

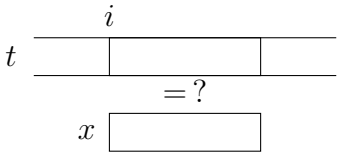
On teste toutes les positions dans t pour déterminer si ce sont des positions d'occurrences de x :

```
let recherche_naive (x : string) (t : string) : int list =
    let l = ref []in
    let n = String.length t
    and m = String.length x in

    for i = 0 to n - m do
        let j = ref 0 in
        while !j < m && x.[!j] = t.[i + !j] do
            incr j
        done;

        if !j = m then l := i::!l
    done;
    !l
```

Complexité : dans le pire cas, la boucle `while` s'arrête au dernier indice dans x (exemple : $t = a^n = \underbrace{a \cdots a}_{n \text{ fois } a}$, et $x = a^{m-1}b$) : $\mathcal{O}((n-m+1)m) = \mathcal{O}(nm)$ si m est petit devant n.

1.1.3 Remarque

Cet algorithme fait partie d'une famille d'algorithmes, dits de *fenêtre glissante* : on fait glisser une fenêtre sur le texte en notant toutes les positions auxquelles la fenêtre contient le motif.



Le côté naïf de cet algorithme vient du fait que l'on fait systématiquement glisser la fenêtre d'un rang, quel que soit son contenu. En analysant les raisons de l'échec d'une comparaison, on peut espérer décaler la fenêtre de plusieurs rangs.

1.2 Algorithme de Boyer-Moore

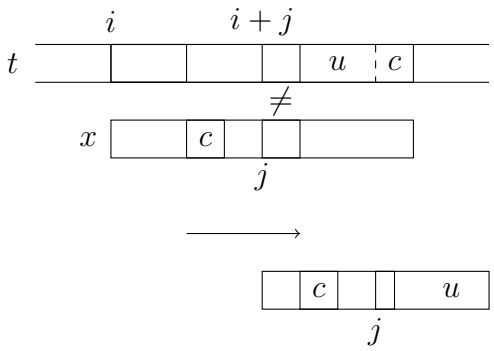
1.2.1 Introduction

L'algorithme de Boyer-Moore est un algorithme de fenêtre glissante, mais la comparaison entre le contenu de la fenêtre et le motif se fait de la droite vers la gauche, afin de repérer en cas d'échec le caractère le plus à droite qui ne correspond pas au motif. À partir de cette position qui entraîne l'échec de la comparaison, on peut calculer un décalage pour la fenêtre.

En pratique le décalage est précalculé pour toutes les positions du motif et il existe de nombreuses variantes suivant la manière dont le décalage est calculé.

1.2.2 Algorithmique de Boyer-Moore-Horspool

Dans cette variante, si la comparaison de $x_0 \dots x_{m-1}$ et $t_i \dots t_{i+m-1}$ (la fenêtre) échoue à l'indice j, *i.e* $x_{j+1} \dots x_{m-1} = t_{i+j+1} \dots t_{i+m-1}$ et $x_j \neq t_{i+j}$, on cherche à aligner t_{i+m-1} avec son occurrence la plus à droite dans x (sauf x_{m-1})



Exemple : $x = aababab$ et $t = aabbbababacaabbaba$

$aababab$
×
$aababab$
×
$aababab$
×

Algorithme : pour toute lettre a, on note $d(a)$ le décalage à effectuer pour aligner cette lettre avec don occurrence la plus à droite dans x (sauf la dernière lettre) en cas d'échec d'une comparaison avec une fenêtre dont la dernière lettre est a.

$$d(a) = \begin{cases} |u| & \text{si } u \text{ est les plus petit suffixe non vide de } x \text{ tq } au \text{ est suffixe de } x \\ |x| & \text{si } u \text{ n'existe pas, } i.e \text{ si } x \text{ ne contient pas } a, \text{ ou alors seulement en dernière position} \end{cases}$$

Pseudo-code :

```
i ← 0
Tant que i ⩽ n − m :
    j ← m − 1
    Tant que j ⩾ 0 et x_j = t_{i+j} :
```

$j \leftarrow j - 1$

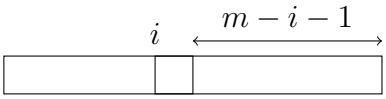
Si $j = -1$:
i est la position d'une occurrence de x
$i \leftarrow i + 1$

Sinon :
$i \leftarrow i + d(t_{i+m-1})$

Précalcul de d :

Pour toute lettre a, $d(a) \leftarrow m$

Pour i de 0 à $m - 2$:
$d(x_i) \leftarrow m - 1 - i$



Complexité :

– Précalcul : $\mathcal{O}(|A| + m)$ où A est l'alphabet, *i.e* l'ensemble des symboles possibles dans un texte.

– Algorithme : dans le pire cas, on décale toujours d'un rang (exemple : $t = a^n$ et $x = ba^{m-1}$) : même complexité que l'algorithme naïf.

En pratique, c'est plus efficace : $\mathcal{O}(n)$ en moyenne (admis).

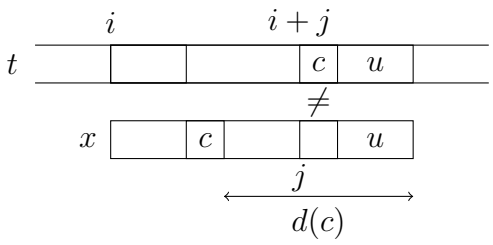
1.2.3 Algorithme de Boyer-Moore (simplifié)

L'algorithme de Boyer-Moore-Horspool ne tient pas compte de ce qu'il se passe lors de la lecture de la fenêtre, mais seulement de son dernier caractère.

Par exemple : si $x = aababab$ et la fenêtre vaut $aabcbab$, le décalage calculé vaut 2 alors que l'absence de c dans x permet un décalage de 4 rangs.

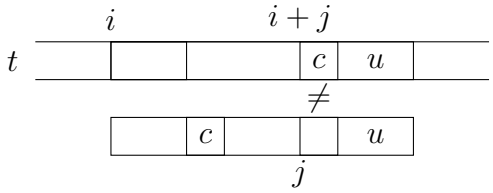
L'idée de Boyer et Moore est d'aligner plutôt de caractère qui provoque l'échec de la comparaison avec son occurrence la plus à droite dans x (sauf la dernière lettre).

Plus précisément, si $x_j \neq t_{i+j}$ et $x_{j+1} \dots t_{i+j+1} \dots t_{i+m-1}$, alors on décale la fenêtre de $d(t_{i+j}) - (m - 1 - j)$



$$\text{décalage} = d(c) - u = d(c) - (m - 1 - j)$$

Attention : cela ne fait pas toujours progresser la recherche :



On obtient un décalage négatif ! Dans ce cas, on décale seulement d'un rang par sécurité.
Algorithme :

```
i ← 0
Tant que i ⩽ n − m :
    j ← m − 1
    Tant que j ⩾ 0 et x_j = t_{i+j} :
        j ← j + 1
    Si j = −1 :
        Occurrence de x à la position i
        i ← i + 1
    Sinon :
        i ← i + max(1, d(t_{i+j}) − (n − 1 − j))
```

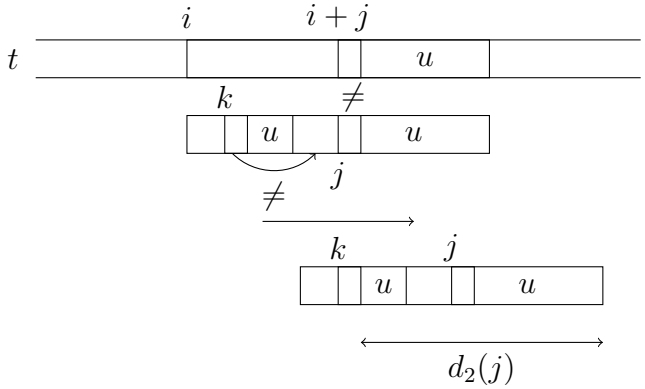
Complexité : dans le pire cas, $\mathcal{O}(nm)$ avec le même exemple qu'en 1.2.2 (page 3).

1.2.4 Algorithme de Boyer-Moore

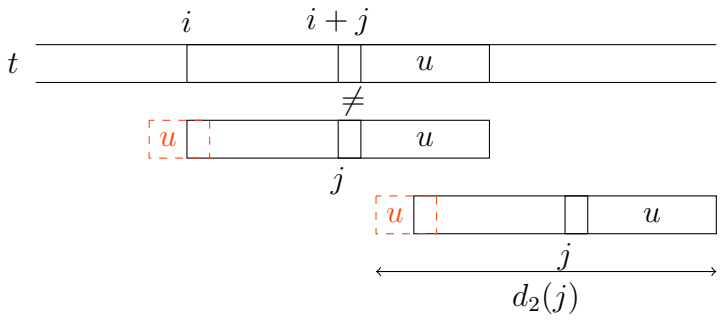
La version complète de l'algorithme de Boyer-Moore utilise une seconde fonction de décalage.

Celle de 1.2.3 (page 4) correspond à la règle du *mauvais caractère* : si un caractère de la fenêtre fait échouer la comparaison avec le motif, on essaie de l'aligner avec sa dernière occurrence dans le motif (sauf la dernière lettre).

Il y a aussi la règle du *bon suffixe* : lorsqu'un caractère fait échouer la comparaison, on a réussi à lire un suffixe de x dans la fenêtre. On peut essayer d'aligner ce suffixe dans le texte avec son occurrence la plus à droite dans x à condition qu'elle soit précédée d'un caractère différent.



$d_2(j)$ = longueur du plus court suffixe de x qui a $x_{j+1} \dots x_{m-1}$ comme suffixe et préfixe et qui n'est pas précédé dans x de x_j.
Si un tel suffixe n'existe pas, on peut chercher le plus long suffixe de u qui est préfixe de x et l'aligner avec le u de la fenêtre.



$d_2(j)$ = longueur de plus court mot w qui a u comme préfixe et x comme suffixe.
Remarque : $|w| \leqslant |u| + |x|$.
Remarque : $\forall j,\ d_2(j) \geqslant 1 + m - j - 1 = m - j$

L'algorithme de Boyer-Moore utilise le décalage maximal entre ceux calculés à partir de d et d_2.
Remarque : si on trouve une occurrence de x ($u = x$), alors on tombe dans le deuxième cas du calcul de d_2 : on cherche le plus long suffixe de x qui est aussi préfixe de x, que l'on appelle le *bord* de x.
Algorithme :

$i \leftarrow 0$
Tant que $i \leqslant n + m$:
    $j \leftarrow m - 1$
    Tant que $j \geqslant 0$ et $x_j = t_{i+j}$:
        $j \leftarrow j - 1$
    Si $j = -1$:
        Occurrence de x à la position i
        $i \leftarrow i + d_2(j) - m$
    Sinon :
        $i \leftarrow i + \max(d(t_{i+j}) - (m - j - 1),\ d_2(j) - (m - j - 1))$