Algorithme de Boyer-Moore (ENS 2002)\(^1\)

On s’intéresse au problème de la recherche de la première occurrence d’un motif \(x = x[0..m - 1]\) dans un mot \(y = y[0..n - 1]\). Pour cela, on peut appliquer l’algorithme naïf suivant qui déplace le motif de gauche à droite le long du mot et qui, pour chaque position du motif, teste s’il coïncide avec le facteur correspondant de \(y\) en comparant les lettres du motif et du facteur de droite à gauche.

\[
\begin{align*}
1 & \text{ soit } \text{Recherche\_ naïve}(x, y) = \\
2 & \text{ soit } j \leftarrow m - 1 \text{ dans} \\
3 & \text{ tant que } j < n \text{ faire} \\
4 & \quad i \leftarrow m - 1 \\
5 & \quad \text{ tant que } i \geq 0 \text{ et } x[i] = y[j - m + 1 + i] \text{ faire } i \leftarrow i - 1 \\
6 & \quad \text{ si } i < 0 \text{ alors renvoyer } j - m + 1 (* x = y[j - m + 1..j] *) \\
7 & \quad j \leftarrow j + 1 \\
8 & \text{ échouer } (* \text{ le motif n’apparaît pas } *)
\end{align*}
\]

1. **Heuristique du mauvais caractère**

   Donner une méthode de calcul, à partir du motif \(x\), d’un tableau \(mc\) qui associe un entier à toute lettre \(\ell\) de manière que l’on puisse remplacer la ligne 7 de l’algorithme naïf par l’instruction suivante:

   \[
   j \leftarrow j + \max(1, i - mc[y[j - m + 1 + i]])
   \]

2. **Heuristique du bon suffixe**

   Donner la définition d’un tableau \(bs[0..m - 1]\), calculable à partir de \(x\), tel que l’on puisse remplacer avantageusement la ligne 7 de l’algorithme naïf par

   \[
   j \leftarrow j + \max(bs[i], i - mc[y[j - m + 1 + i]])
   \]

3. **Donner un algorithme en \(O(m)\) qui renvoie la table du bon suffixe \(bs\). On pourra commencer par calculer la table des suffixes \(s[0..m - 1]\) définie par \(s[i] = \) la longueur du plus long suffixe commun à \(x\) et \(x[0..i]\).**

**Corrigé**

1. Dans l’algorithme naïf, l’instruction \(j \leftarrow j + 1\) de la ligne 7 effectue un décalage d’une unité du motif vers la droite quand la lettre \(a = x[i]\) diffère de \(b = y[j - m + 1 + i]\) (le mauvais caractère). Or si l’occurrence la plus à droite de la lettre \(b\) dans \(x\) se trouve à gauche de \(a = x[i]\); \(b = x[i']\) avec \(i' < i\), on peut effectuer un décalage de \(x\) de \(i - i'\) caractères sans risquer de perdre une occurrence de \(x\) dans \(y\). Dans la figure suivante, la flèche symbolise ce décalage.

\[
\begin{array}{c}
\text{y} \\
\text{b} \\
\text{u}
\end{array}
\begin{array}{c}
\text{x} \\
\text{a} \\
\text{u}
\end{array}
\begin{array}{c}
\text{x} \\
\text{b} \\
\neq b
\end{array}
\]

---

Si la lettre $b$ n’apparaît pas dans le motif, un décalage autorisé est $i + 1$:

$$\begin{array}{c}
\text{y} \\
\hline
b & u \\
\text{x} \\
\hline
a & u \\
\text{x} \\
\hline
\not{= b}
\end{array}$$

Finalement, on pourra remplacer la ligne 7 par celle indiquée à condition de définir le tableau $mc$ par

$$mc[\ell] = \begin{cases} 
\max\{i \mid x[i] = \ell\} & \text{si } \ell \text{ apparaît dans } x \\
-1 & \text{sinon.}
\end{cases}$$

Le code suivant construit $mc$.

1. **pour** chaque lettre $\ell$ **faire** $mc[\ell] \leftarrow -1$

2. **pour** $i \leftarrow 0$ à $m - 1$ **faire** $mc[x[i]] \leftarrow i$

2. Reprenons la situation de la question précédente dans laquelle $u = x[i + 1 \ldots m - 1] = y[j - m + 2 + i \ldots j]$ et $a = x[i] \not= b = y[j - m + 1 + i]$. Soit $d$ une valeur du décalage à droite du motif pour laquelle il y a coïncidence du motif et du facteur de $y$ correspondant. Si $d \leq i$, alors $u = x[i + 1 \ldots m - 1] = x[i + 1 - d \ldots m - 1 - d]$ et $a = x[i] \not= x[i - d]$:

$$\begin{array}{c}
\text{y} \\
\hline
b & u \\
\text{x} \\
\hline
a & u \\
\text{x} \\
\hline
\end{array}$$

Si $i < d < m$, alors le préfixe $v = x[0 \ldots m - 1 - d]$ de $x$ est aussi suffixe de $x$:

$$\begin{array}{c}
\text{y} \\
\hline
b & u \\
\text{x} \\
\hline
a & u \\
\text{x} \\
\hline
\end{array}$$

On définit donc $bs[i]$ comme le plus petit des $i$-décalages où on appelle $i$-décalage un entier $d$ tel que

- ou bien $1 \leq d \leq i$, $x[i + 1 \ldots m - 1] = x[i + 1 - d \ldots m - 1 - d]$ et $x[i] \not= x[i - d]$;
- ou bien $i < d \leq m$ et $x[d \ldots m - 1] = x[0 \ldots m - 1 - d]$.

3. On donne d’abord un algorithme de calcul de la table des suffixes $s$.

$s[m - 1] = m$ puis on calcule $s[i]$, pour $i = m - 2, m - 3, \ldots, 0$ en introduisant deux variables $f$ et $g$ auxquelles on impose, pour $i = m - 2, m - 3, \ldots, 0$ mais pas pour $i = m - 1$, l’invariant suivant:

$$\begin{array}{c}
i \leq f \leq m - 2 \\
g = f - s[f] \\
g \text{ est minimum.}
\end{array}$$

Noter que, au départ, $f$ n’est pas définie et on donne à $g$ la valeur $m - 1$ afin que la condition $i > g$ soit fausse dans le corps de la boucle.

$$\begin{array}{c}
g & i & f \\
\hline
u & u & u
\end{array}$$
soit \( Table\_des\_suffixes(x) = \)
\[
g \leftarrow m - 1
\]
\[
s[m - 1] \leftarrow m
\]
pour \( i \leftarrow m - 2 \) décroissant à 0 faire
\[
\text{si } i > g \text{ et } s[i + m - 1 - f] < i - g \text{ alors}
\]
\[
s[i] \leftarrow s[i + m - 1 - f]
\]
 sinon
\[
g \leftarrow \min(g, i)
\]
\[
f \leftarrow i
\]
tant que \( g \geq 0 \) et \( x[g] = x[g + m - 1 - f] \) faire \( g \leftarrow g - 1 \)
\[
s[i] \leftarrow f - g
\]
renvoyer \( s \)

La complexité est bien \( \mathcal{O}(m) \) car \( g \) est décément dans le corps de la boucle de la ligne 10.

On propose l’algorithme suivant pour calculer la table du bon suffixe:

soit \( Table\_du\_bon\_suffixe(x, s) = \)
\[
i \leftarrow 0
\]
pour \( d = 1 \) à \( m \) faire
\[
\text{si } d = m \text{ ou } s[m - 1 - d] = m - d \text{ alors}
\]
tant que \( i < d \) faire
\[
bs[i] \leftarrow d
\]
\[
i \leftarrow i + 1
\]
pour \( d \leftarrow m - 1 \) décroissant à 1 faire
\[
bs[m - 1 - s[m - 1 - d]] \leftarrow d
\]
renvoyer \( bs \)

Dans une première étape, lignes 2 à 7, on donne à \( bs[i] \) la valeur du plus petit \( i \)-décalage \( > i \). On remarque pour cela qu’un entier \( d > i \) est un tel décalage ssi il vérifie la condition suivante:

\[
d = m \text{ ou } (d < m \text{ et } s[m - 1 - d] = m - d)
\]

Soit donc \( 1 \leq d_1 < \cdots < d_p = m \) l’ensemble des \( d \) vérifiant cette condition; alors, si \( d_{k-1} \leq i < d_k \) (on convient \( d_0 = 0 \)), le plus petit \( i \)-décalage \( > i \) est \( d_k \).

Il reste, dans une deuxième étape, lignes 8 et 9, à mettre à jour \( bs[i] \) dans le cas où il existe un \( i \)-décalage \( d \leq i \). Dans ce cas, \( i = m - 1 - s[j] \) où \( j = m - 1 - d \). A toute valeur de \( d \) correspond donc un unique \( i \) et il suffit de parcourir les valeurs possibles de \( d \) en ordre décroissant pour être assuré que chaque \( bs[i] \) se verra affecté du \( d \) minimum.
En fait $d = i + 1$ pour les valeurs de $d$ telles que $s[j] = j + 1$ et, dans ce cas, l’affectation $bs[i] \leftarrow d$ n’a pas d’effet car $bs[i]$ a déjà la valeur $i + 1$ d’après la première étape.

L’algorithme est en $O(m)$ car $i$ augmente dans le corps de la boucle des lignes 5 à 7.

On peut montrer que l’algorithme de Boyer-Moore est $O(n)$. 