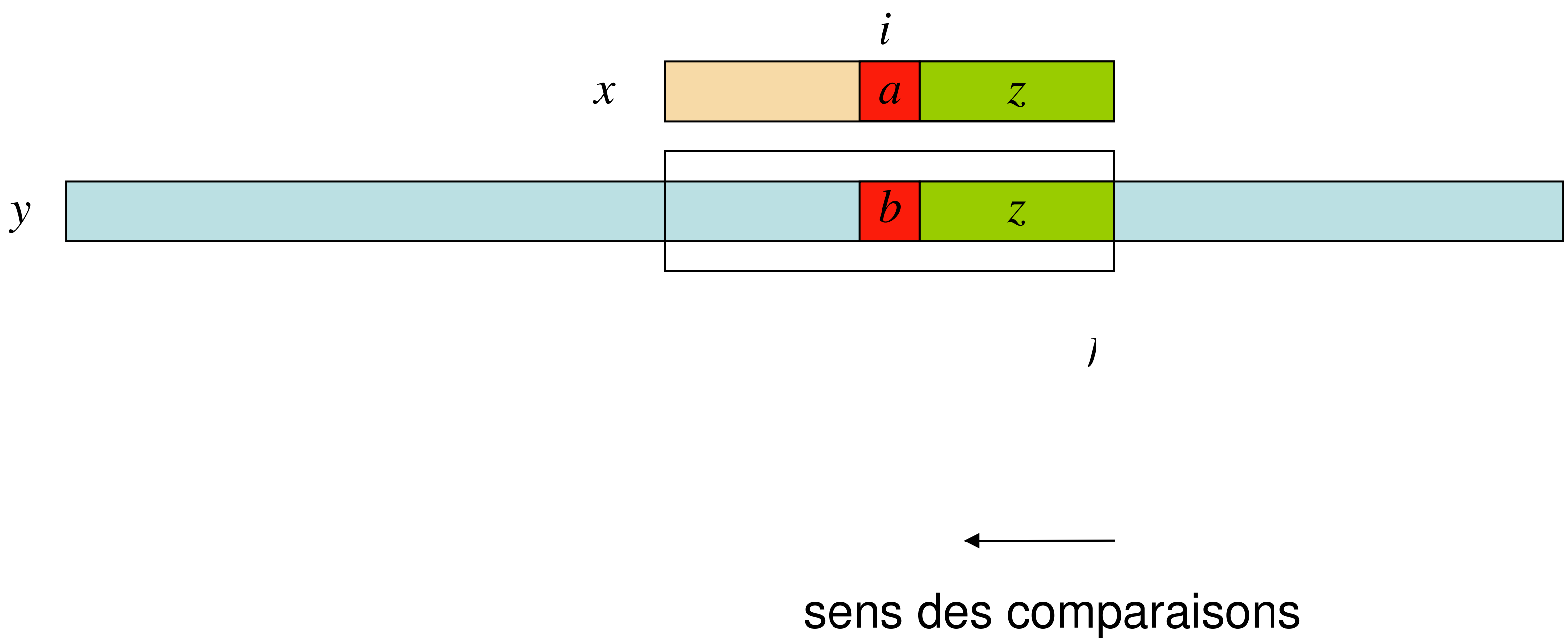
Algorithme de Boyer-Moore

Algorithme de Boyer-Moore

Tentative : compare les lettres du mot et les lettres de la fenêtre, de la droite vers la gauche, en commençant par la lettre la plus à droite

Situation générale



Le suffixe $z = x[i+1..m-1]$ est égal au facteur $z = y[j-m+i+2..j]$ et la lettre $a = x[i]$ est différente de la lettre $b = y[j-m+i+1]$

Décalage du bon suffixe

Alors le décalage consiste à choisir le maximum entre aligner

- en face de $y[j\text{-}m+i+2..j] = x[i+1..m\text{-}1] = z$ le facteur z de $x[0..m\text{-}2]$ précédé par une lettre différente de $a = x[i]$ le plus à droite ou s'il n'existe pas, à aligner le plus long préfixe de x suffixe de z

- en face de $y[j\text{-}m+i+1] = b$ l'occurrence de b la plus à droite dans x

Décalage du bon suffixe

Alors le décalage consiste à choisir le maximum entre aligner

- en face de $y[j\text{-}m+i+2..j] = x[i+1..m\text{-}1] = z$ le facteur z de $x[0..m\text{-}2]$ précédé par une lettre différente de $a = x[i]$ le plus à droite ou s'il n'existe pas, à aligner le plus long préfixe de x suffixe de z

➢ ne dépend que de x

- en face de $y[j\text{-}m+i+1] = b$ l'occurrence de b la plus à droite dans x

➢ dépend de A

Décalage du bon suffixe

On définit deux conditions, pour $0 \leq i \leq m$-1, $1 \leq d \leq m$ et $b \in A$

la condition de suffixe *Cs*

- $$Cs(i,d) = \begin{cases} 0 < d \leq i+1 \text{ et } x[i\text{-}d+1..m\text{-}d\text{-}1] \preccurlyeq_{suff} x \\ \text{ou} \\ i+1 < d \text{ et } x[0..m\text{-}d\text{-}1] \preccurlyeq_{suff} x \end{cases}$$

Décalage du bon suffixe

On définit la condition *Co*,

pour $0 \leq i \leq m\text{-}1$ et $1 \leq d \leq m$ par

$$Co(i,d) = \begin{cases} 0 < d \leq i \text{ et } x[i\text{-}d] \neq x[i] \\ \text{ou} \\ i < d \end{cases}$$

Décalage du bon suffixe

Alors la table du bon suffixe est définie comme suit, pour $0 \leq i \leq m\text{-}1$

$$bon\text{-}suff[i] = \min \{ \; d \mid Cs(i,d) \text{ et } Co(i,d) \text{ sont satisfaites} \; \}$$

Décalage de la dernière occurrence

On définit la table

$dern\text{-}occ : A \rightarrow \{ 1,2,\ldots,m \}$

de la façon suivante

$$dern\text{-}occ[a] = \min \{ \{ k \mid 1 \leq k \leq m\text{-}1 \text{ et } x[m\text{-}1\text{-}k] = a \} \cup \{m\} \}$$

pour $a \in A$

algo BM (x,m,y,n)

$j \leftarrow m$-1

tantque $j < n$ faire

$i \leftarrow m$-1

tantque $i \geq 0$ et $x[i] = y[j\text{-}m\text{+}1\text{+}i]$ faire

$i \leftarrow i$-1

si $i < 0$ alors

signaler une occurrence de x

$j \leftarrow j + pér(x)$

sinon

$j \leftarrow j + \max\{bon\text{-}suff[i],$

$dern\text{-}occ[y[j\text{-}m\text{+}1\text{+}i]]\text{-}m\text{+}1\text{+}i\}$

Décalage

Remarque

$$p\acute{e}r(x) = bon\text{-}suff[0]$$

$$\forall\ a \in\ A$$

Phase de prétraitement : calcul de la table *dern-occ*

algo DERN-OCC(x,m)

  pour $a \in A$ faire

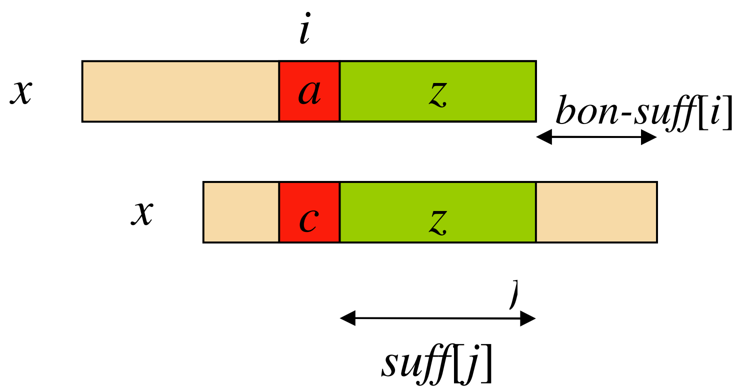
    $dern\text{-}occ[a] \leftarrow m$

  pour $i \leftarrow 0$ à m-2 faire

    $dern\text{-}occ[x[i]] \leftarrow m\text{-}1\text{-}i$

  retourner $dern\text{-}occ$

Phase de prétraitement : calcul de la table *bon-suff*



Calcul de la table *bon-suff*

On définit la table *suff* par :

$suff[i] = |lsc(x, x[0..i])|$

où $lsc(x,y)$ = longueur du plus long suffixe commun à x et y

algo SUFFIXES(x,m)

$g \leftarrow m$-1

$suff$[m-1] $\leftarrow m$

pour $i \leftarrow m$-2 à 0 faire

si $i > g$ et $suff[i+m-1-f] \neq i-g$ alors

$suff[i] \leftarrow \min\{suff[i+m-1-f], i-g\}$

sinon

$g \leftarrow \min\{i,g\}$

$f \leftarrow i$

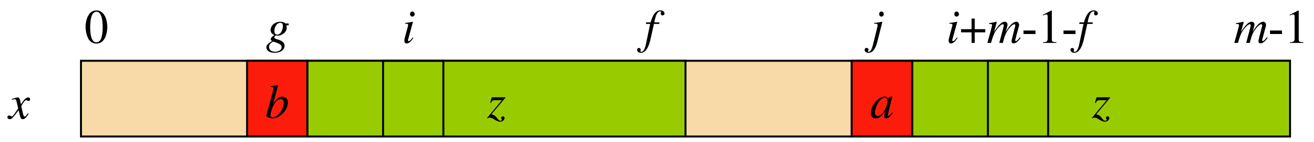
tantque $g \geq 0$ et $x[g] = x[g+m+1-f]$ faire

$g \leftarrow g-1$

$suff[i] \leftarrow f-g$

retourner $suff$

Invariants de l'algorithme SUFFIXES(x,m)



$$z = lsc(x, x[0..f])$$

Complexité de l'algorithme SUFFIXES(x,m)

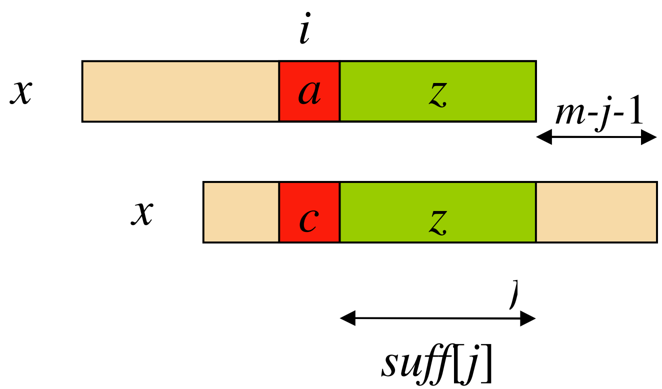
Proposition 1

L'algorithme SUFFIXES(x,m) calcule les valeurs de la table *suff* en temps et espace $O(m)$.

Preuve (idée) :

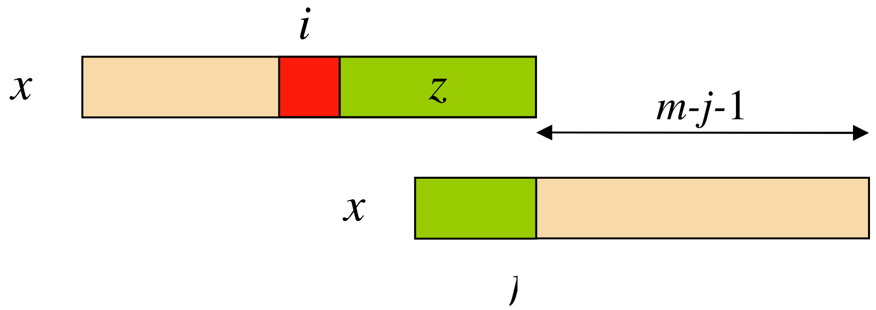
- Chaque lettre comparée positivement n'est pas recomparée : m comparaisons.
- Chaque lettre comparée négativement implique une décrémentation de i : m-1 comparaisons.
- Au total : 2m-1 comparaisons.

Phase de prétraitement : calcul de la table *bon-suff*



$bon\text{-}suff[i] = m\text{-}j\text{-}1$
avec $i = m\text{-}1\text{-}suff[j]$

Phase de prétraitement : calcul de la table *bon-suff*



$$bon\text{-}suff[i] = m\text{-}j\text{-}1$$

```
algo BON-SUFFIXES(x,m,suff)
   i ← 0
   pour j ← m-2 à -1 faire
      si j = -1 ou suff[j] = j+1 alors
         tantque i < m-1-j faire
            bon-suff[i] ← m-1-j
            i ← i+1
   pour j ← 0 à m-2 faire
      bon-suff[m-1-suff[j]] ← m-1-j
   retourner bon-suff
```

Complexité de l'algorithme BON-SUFFIXE(x,m,$suff$)

Proposition 2

L'algorithme BON-SUFFIXE(x,m,$suff$) calcule les valeurs de la table *bon-suff* en temps et espace $O(m)$.

Preuve (idée) :

- j varie de m-2 à -1 dans la première boucle **pour** ;
- i varie de 0 à m-1-min$\{j\}$=m dans la première boucle **pour** ;
- j varie de 0 à m-2 dans la deuxième boucle **pour**.

Complexité de l'algorithme de Boyer-Moore

Théorème 3 [Cole 1994]

Lors de la localisation d'un mot x de longueur m tel que $p\acute{e}r(x) > m/3$ dans un texte y de longueur n, l'algorithme BM(x,m,y,n) effectue moins de $4n$ comparaisons entre des lettres de x et des lettres de y.

Complexité de l'algorithme de Boyer-Moore

Théorème 4 [Cole 1994]

Lors de la localisation d'un mot x de longueur m tel que $pér(x) > m/2$ dans un texte y de longueur n, l'algorithme BM(x,m,y,n) effectue moins de $3n$ comparaisons entre des lettres de x et des lettres de y.

Améliorations

- Algorithme Turbo-BM : au pire $2n$ comparaisons entre des lettres du mot et des lettres du texte
- Algorithme Apostolico-Giancarlo : au pire $3n/2$ comparaisons entre des lettres du mot et des lettres du texte