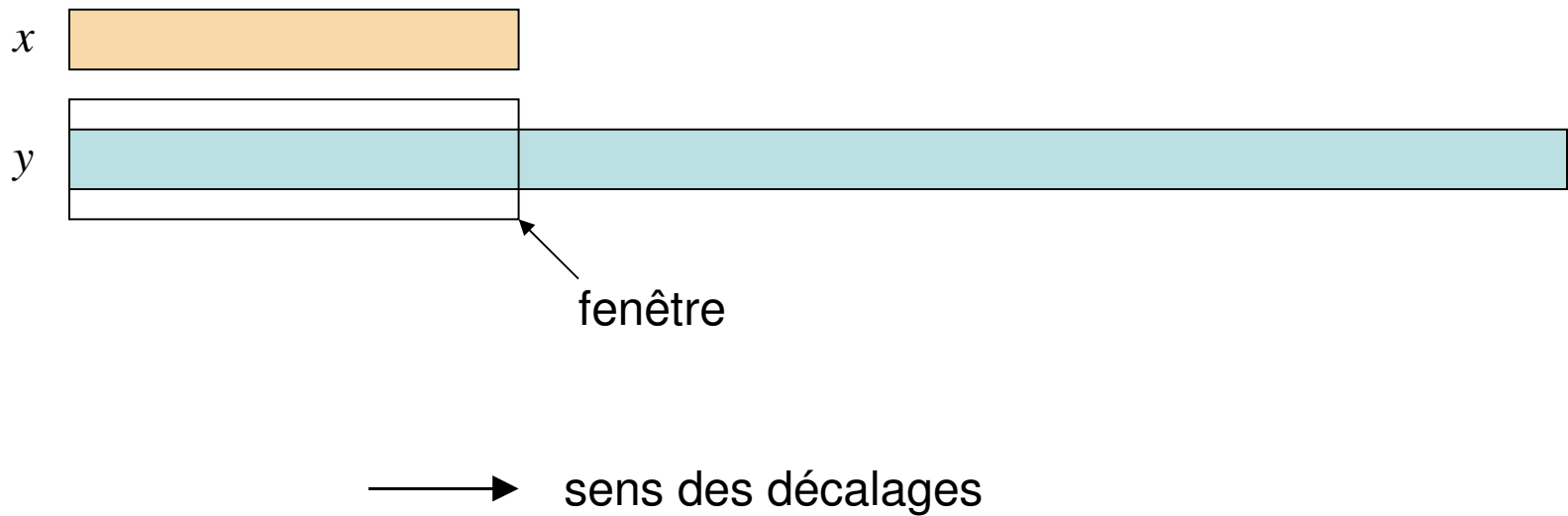


Algorithme de Boyer-Moore

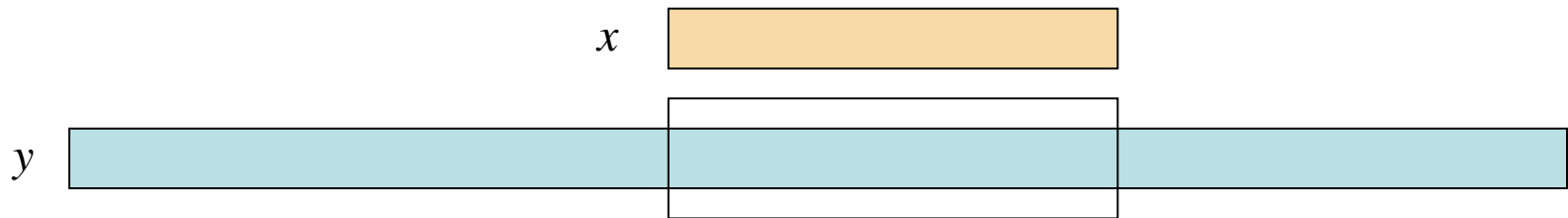
Phase de recherche

- Problème : localiser toutes les occurrences d'un mot x de longueur m dans un texte y de longueur n
- Méthode : fenêtre glissante

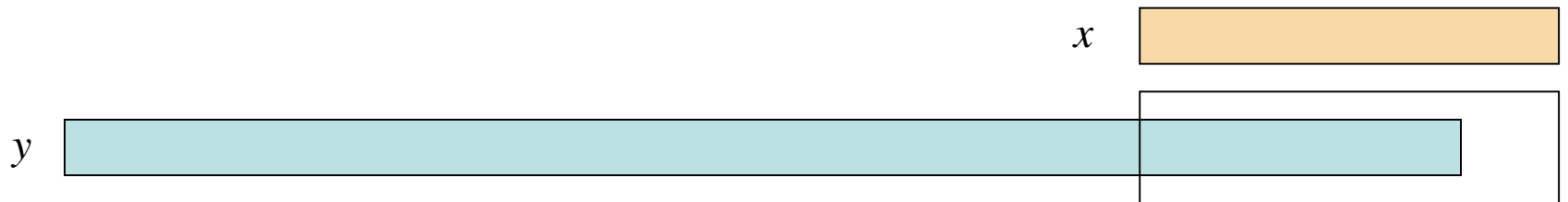
Fenêtre glissante



Fenêtre glissante



Fenêtre glissante



Algorithme de recherche de mot

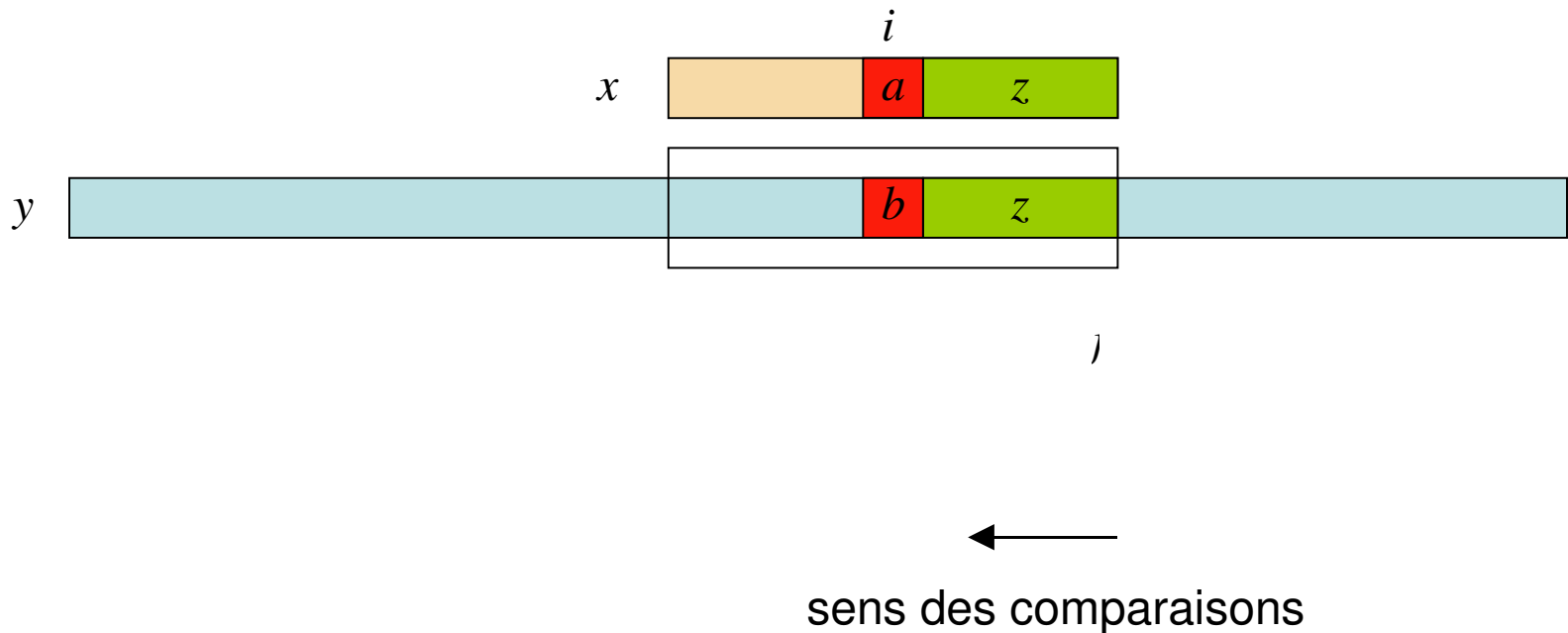
Succession de

- tentatives (comparaison du mot et du contenu de la fenêtre)
- décalages (de la fenêtre vers la droite)

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

Fenêtre glissante



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

Alors le décalage consiste à choisir le maximum entre aligner

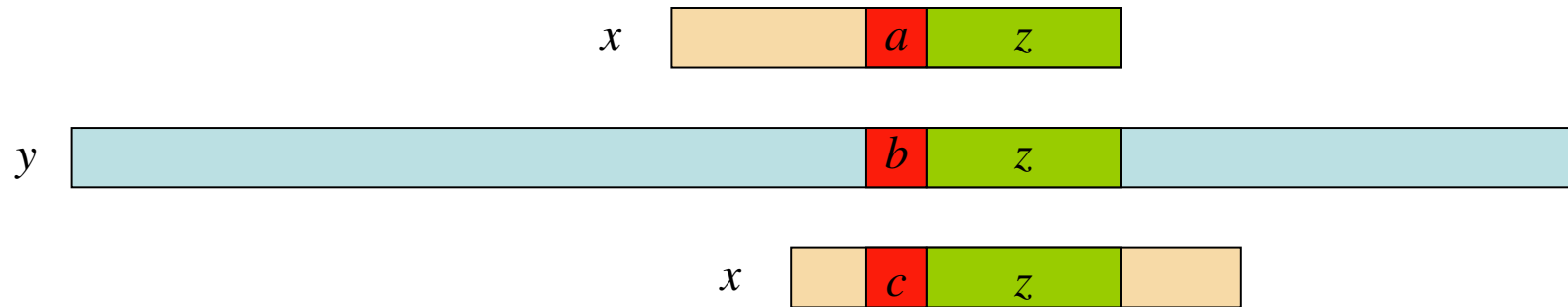
- en face de $y[j-m+i+1..j]$ le facteur cz de $x[0..m-2]$ le plus à droite ou s'il n'existe pas, à aligner le plus long préfixe de x suffixe de z
- en face de $y[j-m+i+1]$ l'occurrence de b la plus à droite dans x

Décalage

Alors le décalage consiste à choisir le maximum entre aligner

- en face de $y[j-m+i+1..j]$ le facteur cz de $x[0..m-2]$ 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 $y[j-m+i+1]$ l'occurrence de b la plus à droite dans x
 - dépend de A

3 types de décalages



- pas de contrainte sur c (éventuellement $c = a$) :
décalage de faible suffixe
- $c \neq a$: **décalage du bon suffixe**
- $c = b$: **décalage du meilleur facteur**

Décalage

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

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

Décalage

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

la condition d'occurrence Co

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

Décalage

Alors la fonction de meilleur facteur est définie
comme suit, pour $0 \leq i \leq m-1$, et $b \in A$

$$\textit{meil-fact}(i,b) = \min \{ d \mid Cs(i,d) \text{ et } Co(b,i,d) \text{ sont satisfaites } \}$$

algo BM(x, m, y, n)

$j \leftarrow m-1$

tantque $j < n$ faire

$i \leftarrow m-1$

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

$i \leftarrow i-1$

si $i < 0$ alors

signaler une occurrence de x

$j \leftarrow j + \text{pér}(x)$

sinon

$j \leftarrow j + \text{meil-fact}(i, y[j-m+1+i])$

Décalage

On définit la condition *Co-aff*,
pour $0 \leq i \leq m-1$ et $1 \leq d \leq m$ par

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

Décalage

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

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

algo BM-FAIBLE(x, m, y, n)

$j \leftarrow m-1$

tantque $j < n$ faire

$i \leftarrow m-1$

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

$i \leftarrow i-1$

si $i < 0$ alors

signaler une occurrence de x

$j \leftarrow j + \text{pér}(x)$

sinon

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

Décalage

On définit la table

$$\textit{dern-occ} : A \rightarrow \{ 1, 2, \dots, m \}$$

de la façon suivante

$$\textit{dern-occ}[a] = \min \{ \{ m-1-k \mid 0 \leq k \leq m-2 \text{ et } x[k] = a \} \cup \{ m \} \}$$

pour $a \in A$

algo BM-FAIBLE-AVEC-DERN-OCC(x, m, y, n)

$j \leftarrow m-1$

tantque $j < n$ **faire**

$i \leftarrow m-1$

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

$i \leftarrow i-1$

si $i < 0$ **alors**

signaler une occurrence de x

$j \leftarrow j + \text{pér}(x)$

sinon

$j \leftarrow j + \max\{\text{bon-suff}[i],$
 $\text{dern-occ}[y[j-m+1+i]]-m+1+i\}$

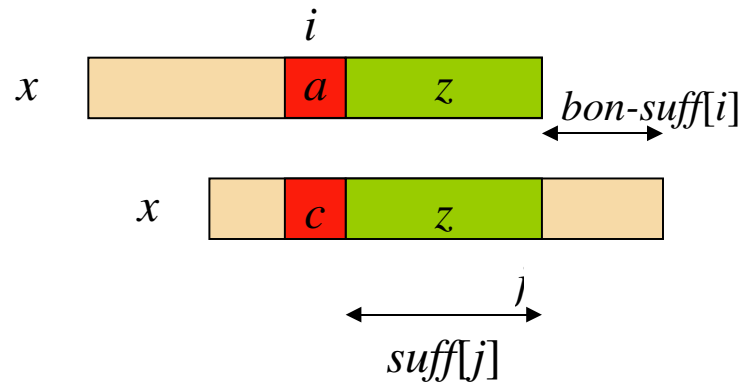
Décalage

Remarque

$$pér(x) = \text{meil-fact}(0,a) = \text{bon-suff}[0]$$

$$\forall a \in A$$

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

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

$f \leftarrow i$

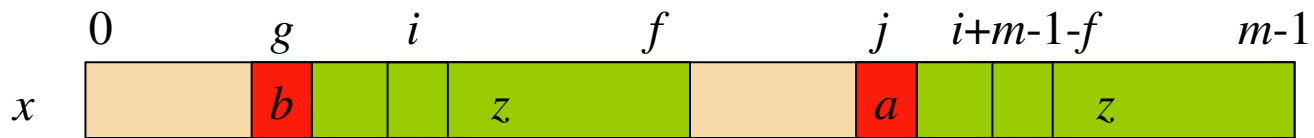
tantque $g \geq 0$ **et** $x[g] = x[g+m+1-f]$ **faire**

$g \leftarrow g-1$

$suff[i] \leftarrow f-g$

retourner $suff$

Invariants de l'algorithme SUFFIXES(x, m)



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

Complexité de l'algorithme SUFFIXES(x, m)

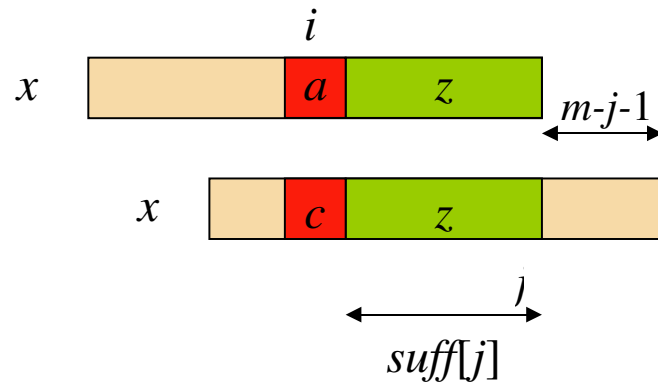
Proposition 1

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

Preuve (idée) :

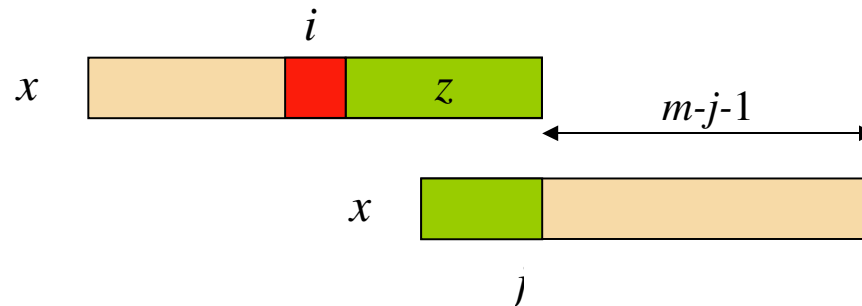
- Chaque lettre comparée positivement n'est pas recomparée : m comparaisons.
- Chaque lettre comparée négativement implique une décrémentation de i : $m-1$ comparaisons.
- Au total : $2m-1$ comparaisons.

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



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

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



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

algo BON-SUFFIXES($x, m, suff$)

$i \leftarrow 0$

pour $j \leftarrow m-2$ à -1 **faire**

si $j = -1$ **ou** $suff[j] = j+1$ **alors**

tantque $i < m-1-j$ **faire**

$bon-suff[i] \leftarrow m-1-j$

$i \leftarrow i+1$

pour $j \leftarrow 0$ à $m-2$ **faire**

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

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) :

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

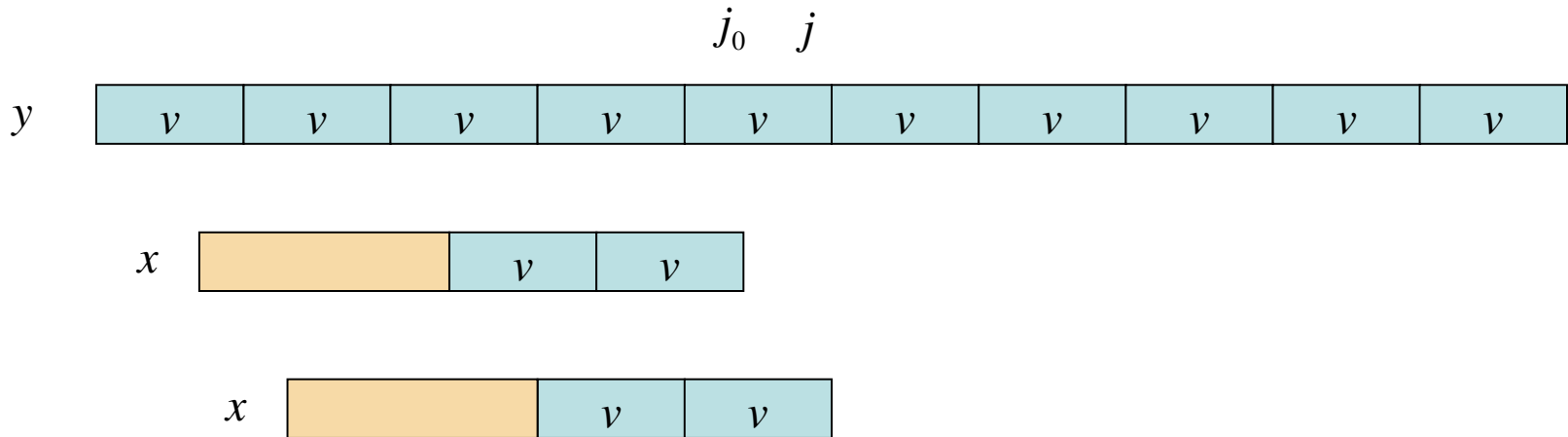
Lemme 3

Lemme 3

Soient x un mot, y un texte, v un mot primitif et k un entier tels que $v^2 \preceq_{suff} x$, $y = v^k$ et $k \geq 2$.

Durant l'exécution de l'algorithme $\text{BM-FAIBLE}(x, m, y, n)$, s'il existe une tentative T_0 à une position j_0 sur y qui n'est pas de la forme $\ell|v|$ ($\ell \in \mathbb{N}$), celle-ci est suivie, immédiatement ou non, d'une tentative à la position $j = \min\{ h \mid h = \ell|v|, h > j_0, \ell \in \mathbb{N} \}$.

Lemme 3 : figure



Lemme 3 : preuve

Preuve

v est primitif donc au plus $|v|$ comparaisons peuvent être effectuées lors de T_0 .

Soit $a = x[i]$ la lettre du mot qui provoque la comparaison négative ($b = y[j_0 - m + 1 + i] \neq a$).

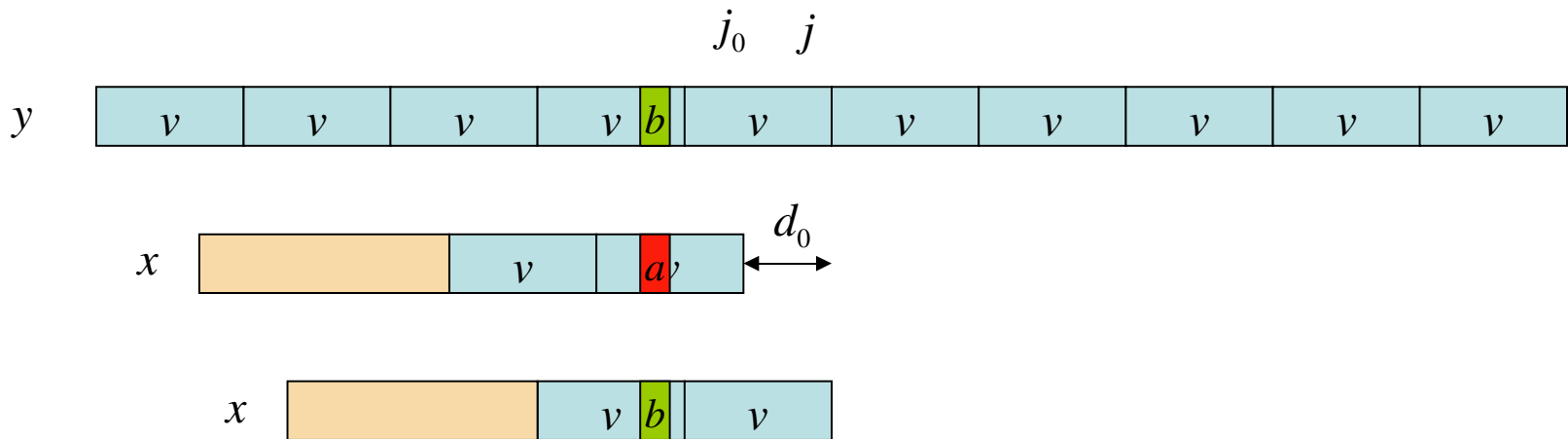
Soit $d_0 = j_0 - j$.

$Cs(i, d_0)$ est satisfaite : $d_0 \leq i$ et $x[i - d_0 + 1 .. m - d_0 - 1] \preceq_{suff} x$

$Co-aff(i, d_0)$ est satisfaite : $b = x[i - d_0] \neq x[i] = a$.

donc $bon-suff[i] \leq d_0$.

Lemme 3 : figure



Lemme 3 : preuve

Si $\text{bon-suff}[i] < d_0$ alors une tentative T_1 suit la tentative T_0 sur laquelle on peut appliquer le même raisonnement.

Une suite finie de telles tentatives mène à une tentative à la position j . □

Analyse de la phase de recherche

Soit T une tentative à la position j sur y avec
 $bz \preceq_{suff} y[0..j]$, $az \preceq_{suff} x$, $a \neq b$, $z = wv^k$, $w \preceq_{suff} v$,
 $aw \preceq_{suff} x$, $k \geq 2$ et v primitif.

Lemme 4

Lemme 4

Il n'y a pas eu de tentatives aux positions $j - \ell|\nu|$ avec $1 \leq \ell \leq k-1$ avant la tentative T .

Lemme 4 : preuve

Preuve

Par l'absurde.

Soit T_0 une tentative à une position $j_0 = j - \ell_0 |v|$ avec $1 \leq \ell_0 \leq k-1$:

$$bwv^{k-\ell_0} \preceq_{suff} y[0..j_0]$$

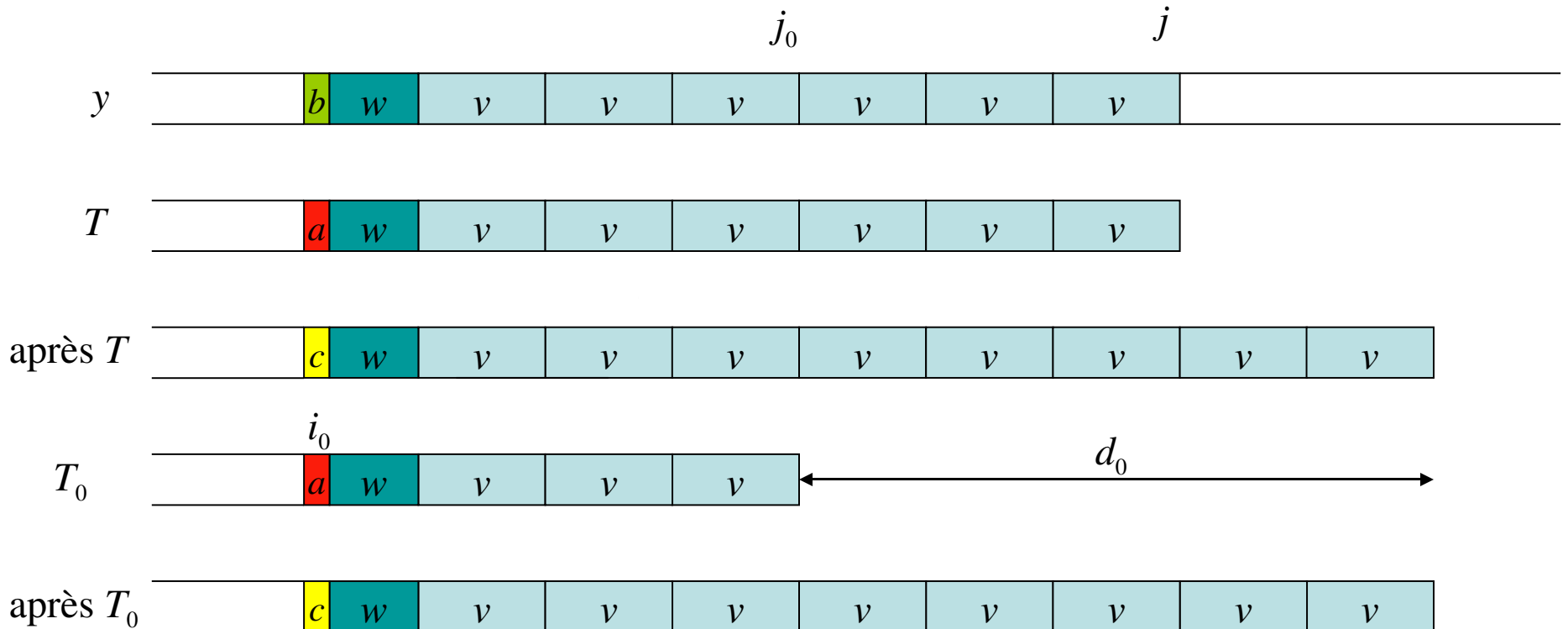
$$awv^{k-\ell_0} \preceq_{suff} x$$

$$i_0 = m - |w| - (k - \ell_0) |v|$$

on alors

$$d_0 = \text{bon-suff}[i_0] > \ell_0 |v|.$$

Lemme 4 : figure



Lemme 4 : preuve

Tout décalage plus petit et non multiple de $|v|$ contredit le fait que v est primitif.

Tout décalage plus petit et multiple de $|v|$ alignerait une lettre de a de x en face de la lettre b de y donc $\text{bon-suff}[i_0] > \ell_0 |v|$ ce qui signifie que la tentative T à la position j n'a pas pu avoir lieu d'où la contradiction.



Lemme 5

Lemme 5

Avant la tentative T il n'y a pas eu de tentative aux positions ℓ telles que $j-|z|+|v| \leq \ell \leq j-|v|$.

Preuve

Lemme 4 : pas de tentative aux positions $j-\ell|v|$ avec $1 \leq \ell \leq k-1$.

Lemme 3 : toute tentative à une position comprise entre $j-|z|+|v|$ et $j-|v|$ est suivie (immédiatement ou non) par une tentative à une position $j-\ell|v|$ avec $1 \leq \ell \leq k-1$. \square

Corollaire 6

Corollaire 6

Avant la tentative T au plus $3|v|-3$ lettres du facteur z de y ont été comparées à des lettres de x .

Preuve

Lemme 5 : les tentatives précédant T et pendant lesquelles des lettres de z ont pu être comparées n'ont pu intervenir qu'à des positions dans les intervalles :

$[j-|z|+1, j-|z|+|v|-1]$ et

$[j-|v|+1, j-1]$.

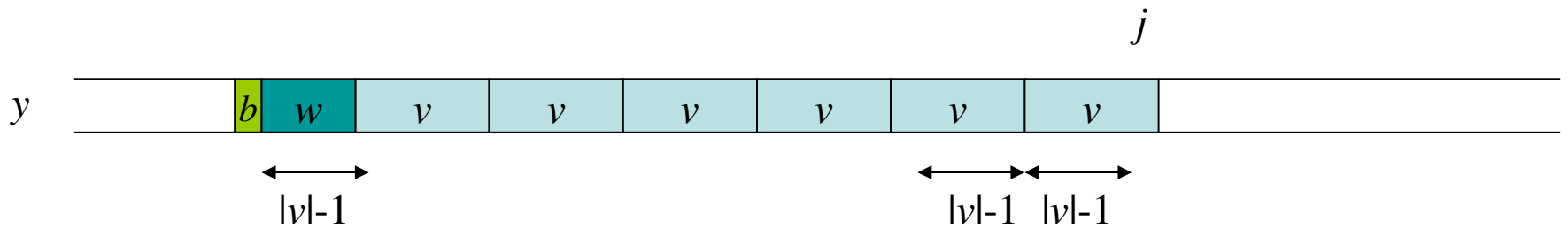
Corollaire 6 : preuve

Comme v est primitif, ces tentatives effectuent au plus $|v|$ comparaisons.

D'où $3(|v|-1)$ lettres de z déjà comparées.



Corollaire 6 : figure



Théorème 7

Théorème 7

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

Théorème 7 : preuve

Preuve

Pour une tentative T à la position j , on note t le nombre d'occurrences de lettres de y comparées pour la première fois au cours de cette tentative et d la longueur du décalage qui suit.

Nous allons borner le nombre de comparaisons effectuées lors de la tentative T par $3d+t$.

Théorème 7 : preuve

Soit $z = lcs(x, y[0..j])$.

Si $|z| \leq 3d$, le nombre de comparaisons effectuées lors de la tentative T est au plus $|z|+1$ et la lettre $y[j]$ n'avait pas été comparée avant T ($t \geq 1$).

Donc, dans ce cas, $|z|+1 \leq 3d+t$.

Théorème 7 : preuve

Si $|z| > 3d$, cela implique $z = wv^k$, $bz \preceq_{suff} y[0..j]$,
 $az \preceq_{suff} x$, $a \neq b$, $k \leq 2$, $aw \preceq_{suff} v$ et $d \geq |v|$.

Corollaire 6 : au plus $3|v| - 3$ lettres de z ont déjà
été comparées

donc $t \geq |z| - 3|v| + 3 \geq |z| - 3d + 3$

d'où $|z| + 1 \leq 3d + |z| - 3d + 3 = |z| + 3 \leq 3d + t$.



Algorithme de Galil

Pour les mots périodiques on applique une technique dite de « mémorisation de préfixe » [Galil 1979] : complexité en $O(n)$.

algo GALIL(x, m, y, n)

$\ell \leftarrow 0$

$j \leftarrow m-1$

tantque $j < n$ **faire**

$i \leftarrow m-1$

tantque $i \geq \ell$ **et** $x[i] = y[j-m+1+i]$ **faire**

$i \leftarrow i-1$

si $i < \ell$ **alors**

signaler une occurrence de x

$\ell \leftarrow m - \text{pér}(x)$

$j \leftarrow j + \text{pér}(x)$

sinon

$\ell \leftarrow 0$

$j \leftarrow j + \max\{\text{bon-suff}[i], \text{dern-occ}[y[j-m+1+i]]\}$

Algorithme de Smyth

[Smyth 2003] : la technique de « mémorisation de préfixe » peut s'appliquer à chaque fois que la longueur du décalage est supérieure à i : preuve simple en $4n$.

```

algo SMYTH( $x, m, y, n$ )
   $\ell \leftarrow 0$ 
   $j \leftarrow m-1$ 
  tantque  $j < n$  faire
     $i \leftarrow m-1$ 
    tantque  $i \geq \ell$  et  $x[i] = y[j-m+1+i]$  faire
       $i \leftarrow i-1$ 
    si  $i < \ell$  alors
      signaler une occurrence de  $x$ 
       $\ell \leftarrow m - \text{pér}(x)$ 
       $j \leftarrow j + \text{pér}(x)$ 
    sinon
      si  $\text{bon-suff}[i] > i$  et  $\text{bon-suff}[i] > \text{dern-occ}[y[j-m+1+i]]$  alors
         $\ell \leftarrow m - \text{bon-suff}[i]$ 
      sinon
         $\ell \leftarrow 0$ 
       $j \leftarrow j + \max\{\text{bon-suff}[i], \text{dern-occ}[y[j-m+1+i]]\}$ 

```

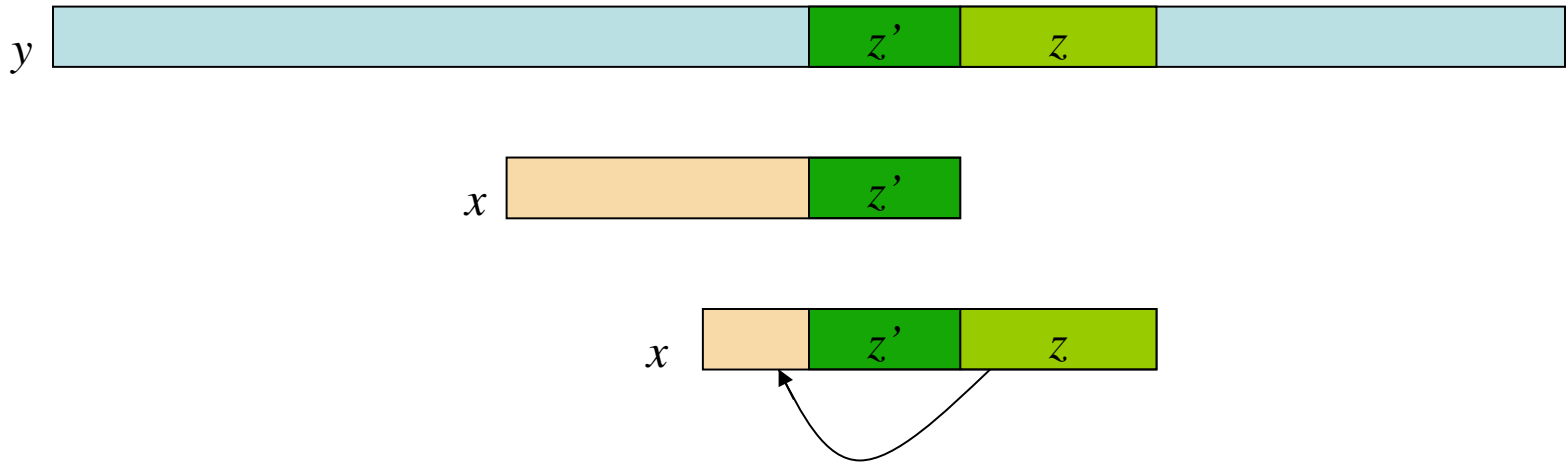
Algorithme TURBO-BM

Mémorisation du facteur du texte reconnu lors de la tentative précédente [CCGJLPR 1994]

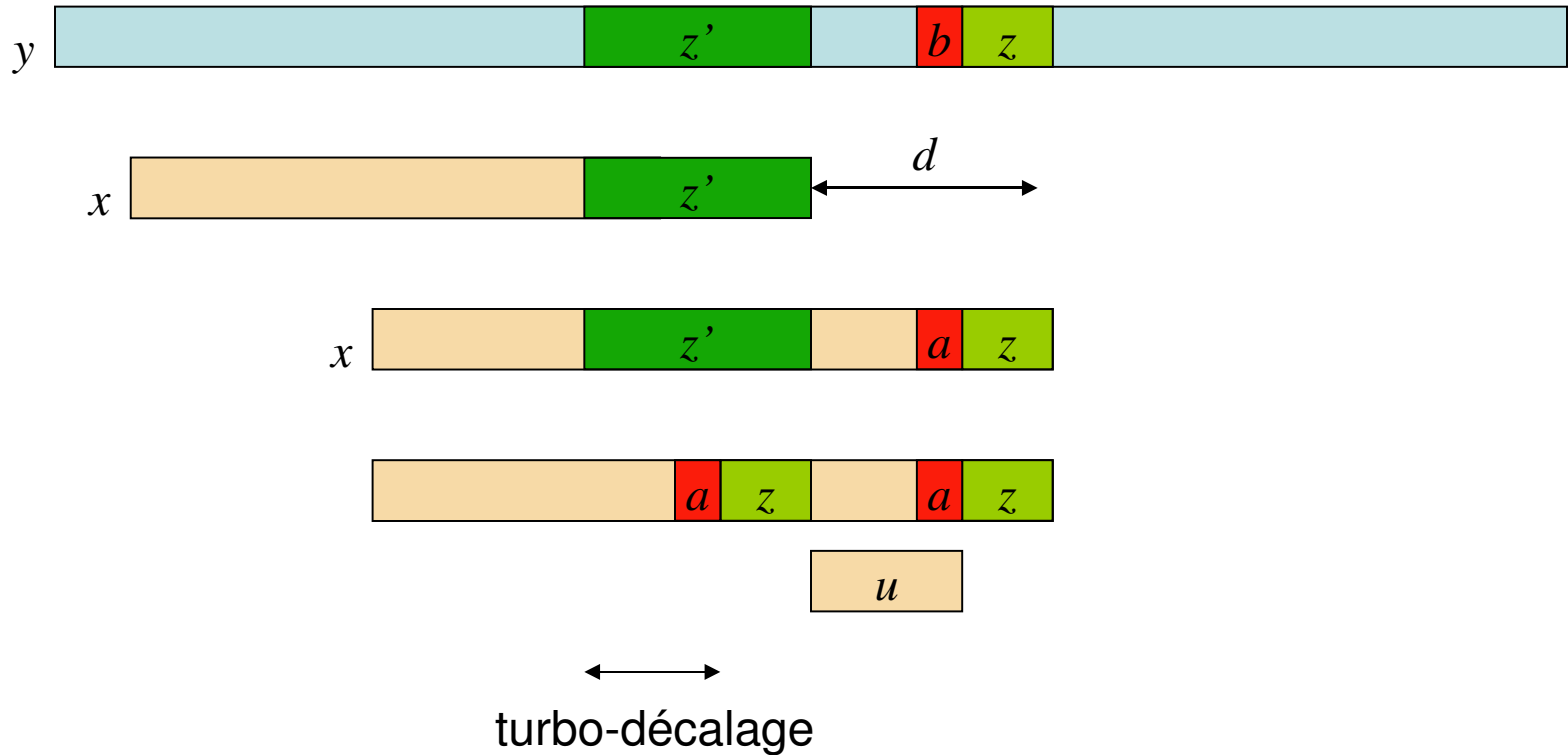
Deux avantages :

- saut ;
- turbo-décalage.

Saut



Turbo décalage : si $|z| < |z'|$



Turbo décalage : si $|z| < |z'|$

$$u = x[m-d..i]$$

$$z'uz \preceq_{suff} x \text{ et } z' \preceq_{suff} x \text{ donc } z' \preceq_{suff} z'uz$$

donc z' est un bord de $z'uz$

donc $d = |uz|$ est une période de $z'uz$

les lettres a et b de y sont à distance d

donc ne peuvent pas être dans une occurrence du
suffixe $z'uz$ de x

donc le décalage doit être supérieur à $|z'| - |z|$.

```

algo TURBO-BM( $x, m, y, n$ )
   $d \leftarrow 0$ 
   $mémoire \leftarrow 0$ 
   $j \leftarrow m-1$ 
  tantque  $j < n$  faire
     $i \leftarrow m-1$ 
    tantque  $i \geq 0$  et  $x[i] = y[j-m+1+i]$  faire
      si  $i = m-d$  alors
         $i \leftarrow i - mémoire - 1$            // saut
      sinon
         $i \leftarrow i - 1$ 
    si  $i < 0$  alors
      signaler une occurrence de  $x$ 
       $d \leftarrow pér(x)$ 
       $mémoire \leftarrow m - d$ 
    sinon
       $turbo \leftarrow mémoire - m + 1 + i$ 
      si  $turbo \leq bon-suff[i]$  alors
         $d \leftarrow bon-suff[i]$ 
         $mémoire \leftarrow \min\{m-d, m-i\}$ 
      sinon
         $d \leftarrow \max\{turbo, m-i-1\}$ 
         $mémoire \leftarrow 0$ 
     $j \leftarrow j + d$ 

```

Algorithme TURBO-BM

Complexité : au pire $2n$ comparaisons entre lettres du mot et lettres du texte.

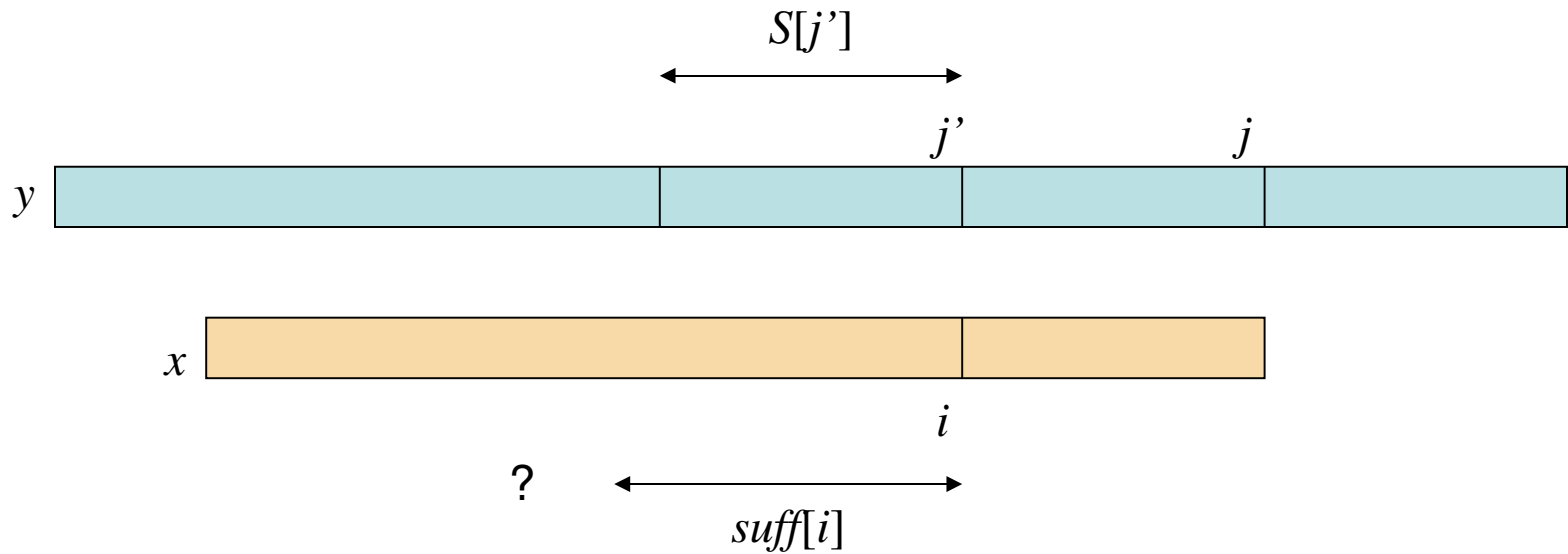
Algorithme Apostolico-Giancarlo

Mémorisation des facteurs du texte plus long
suffixe du mot aux positions droites des fenêtres :

$$S[j'] = k \Leftrightarrow \begin{cases} k < m \text{ et } x[m-k..m-1] = y[j'-k+1..j'] \text{ et } x[m-k-1] \neq y[j'-k] \\ \text{ou} \\ k = m \text{ et } x = y[j'-m+1..j'] \end{cases}$$

$$\textit{suff}[i] = \textit{lcs}(x[0..i], x)$$

Algorithme Apostolico-Giancarlo



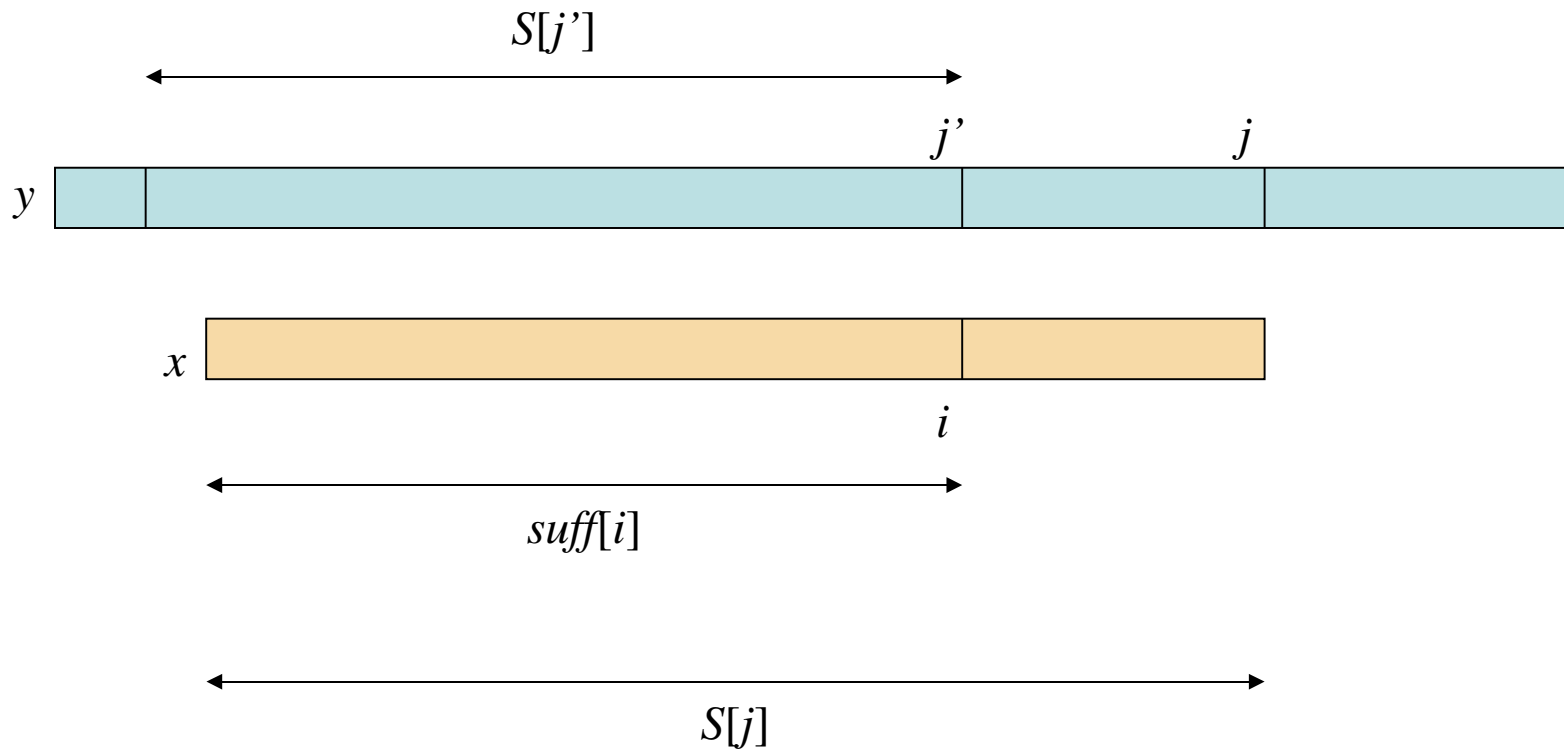
Algorithme Apostolico-Giancarlo

4 cas :

cas 1 :

- $suff[i] \leq S[j']$ et $suff[i] = i+1 : S[j] = m$ et $d = pér(x)$

Algorithme Apostolico-Giancarlo



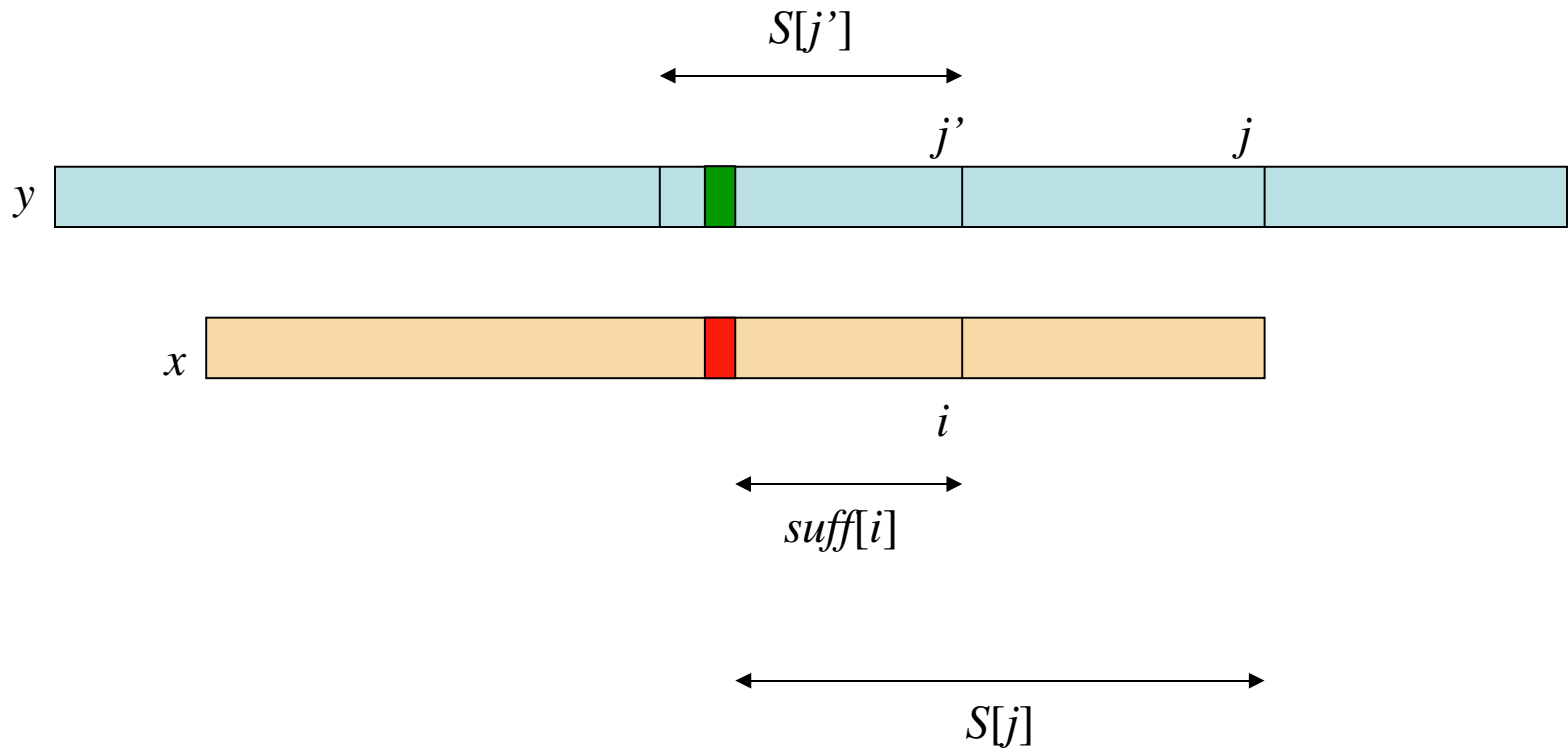
Algorithme Apostolico-Giancarlo

4 cas :

cas 2 :

- $suff[i] < S[j']$ et $suff[i] \leq i$: $S[j] = m-1-i+suff[i]$
et $d = bon-suff[i-suff[i]]$

Algorithme Apostolico-Giancarlo



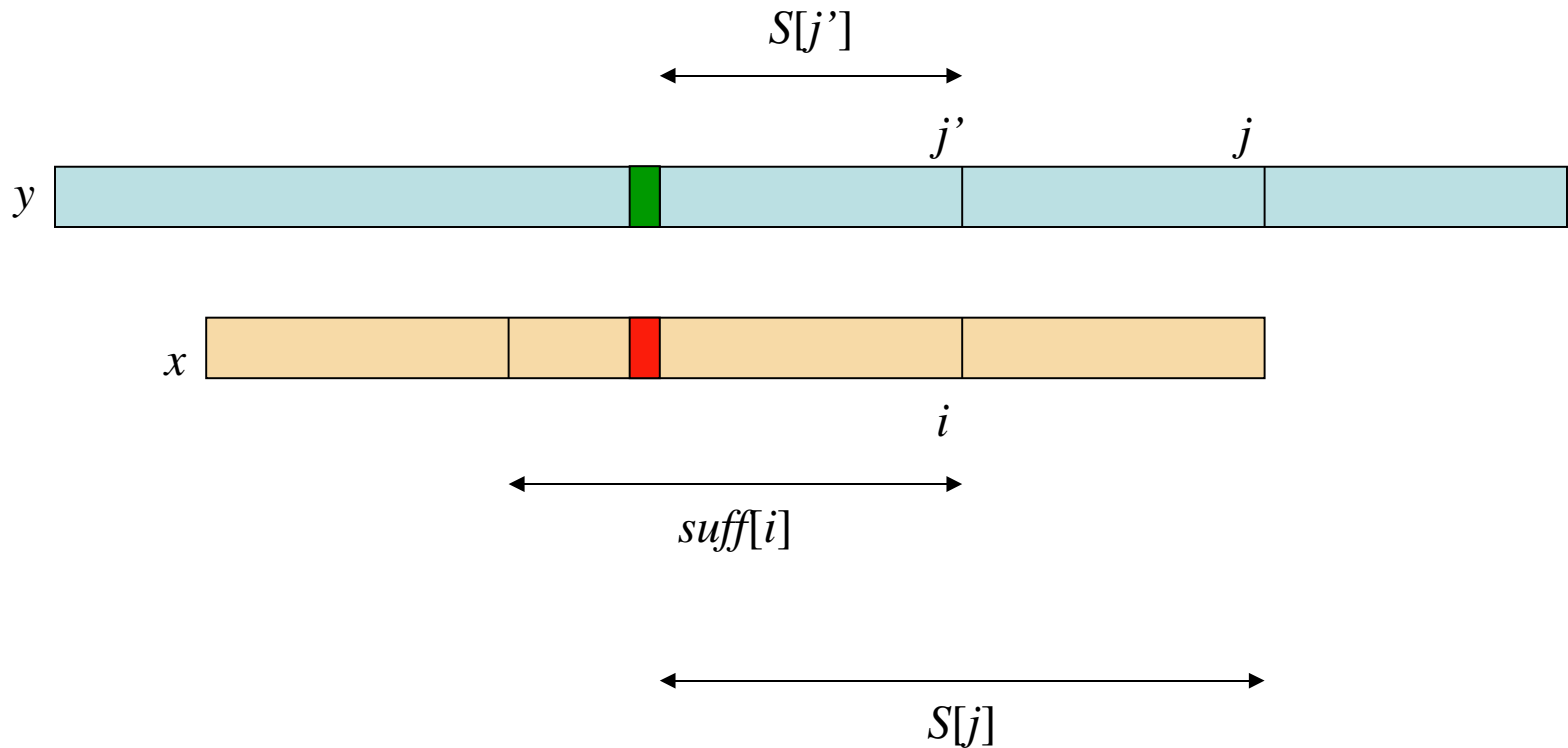
Algorithme Apostolico-Giancarlo

4 cas :

cas 3 :

- $suff[i] > S[j']$ et $suff[i] \leq i$: $S[j] = m-1-i+S[j']$
et $d = bon-suff[i-S[j']]$

Algorithme Apostolico-Giancarlo



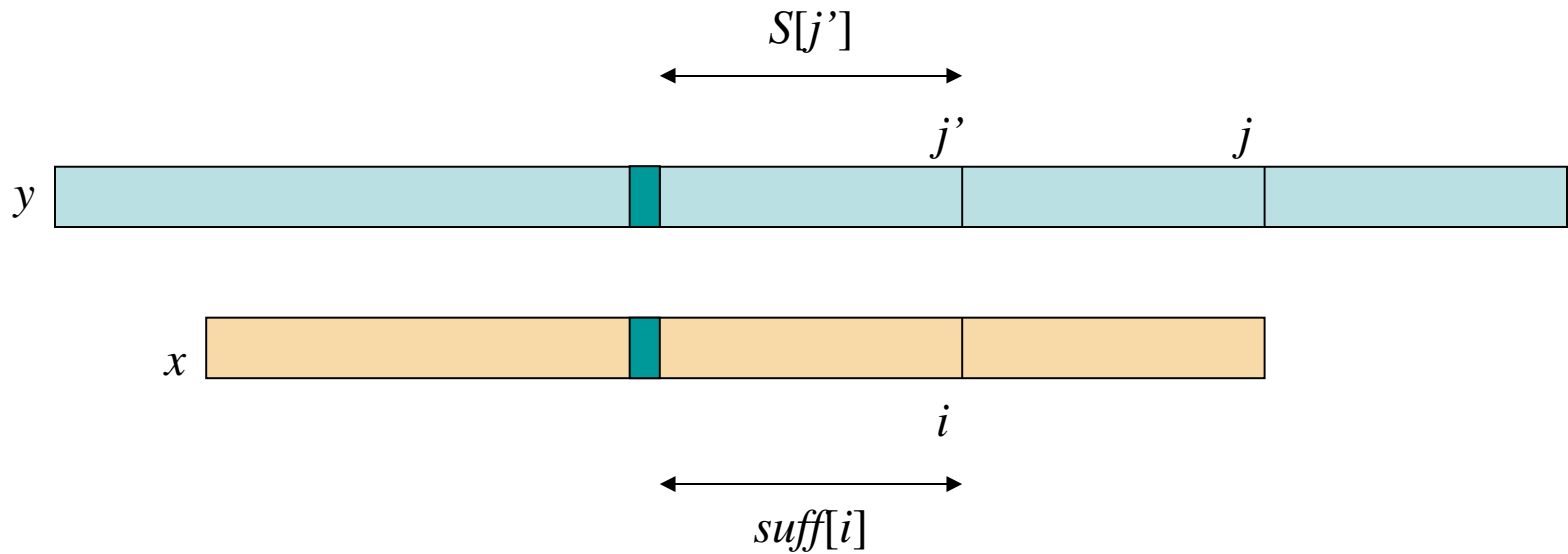
Algorithme Apostolico-Giancarlo

4 cas :

cas 4 :

- $suff[i] = S[j']$: saut

Algorithme Apostolico-Giancarlo



Algorithme Apostolico-Giancarlo

Nombres de comparaisons dans le pire des cas :

- $2n$ [Apostolico & Giancarlo 1986]
- $3n/2$ [Crochemore & Lecroq 1997]