

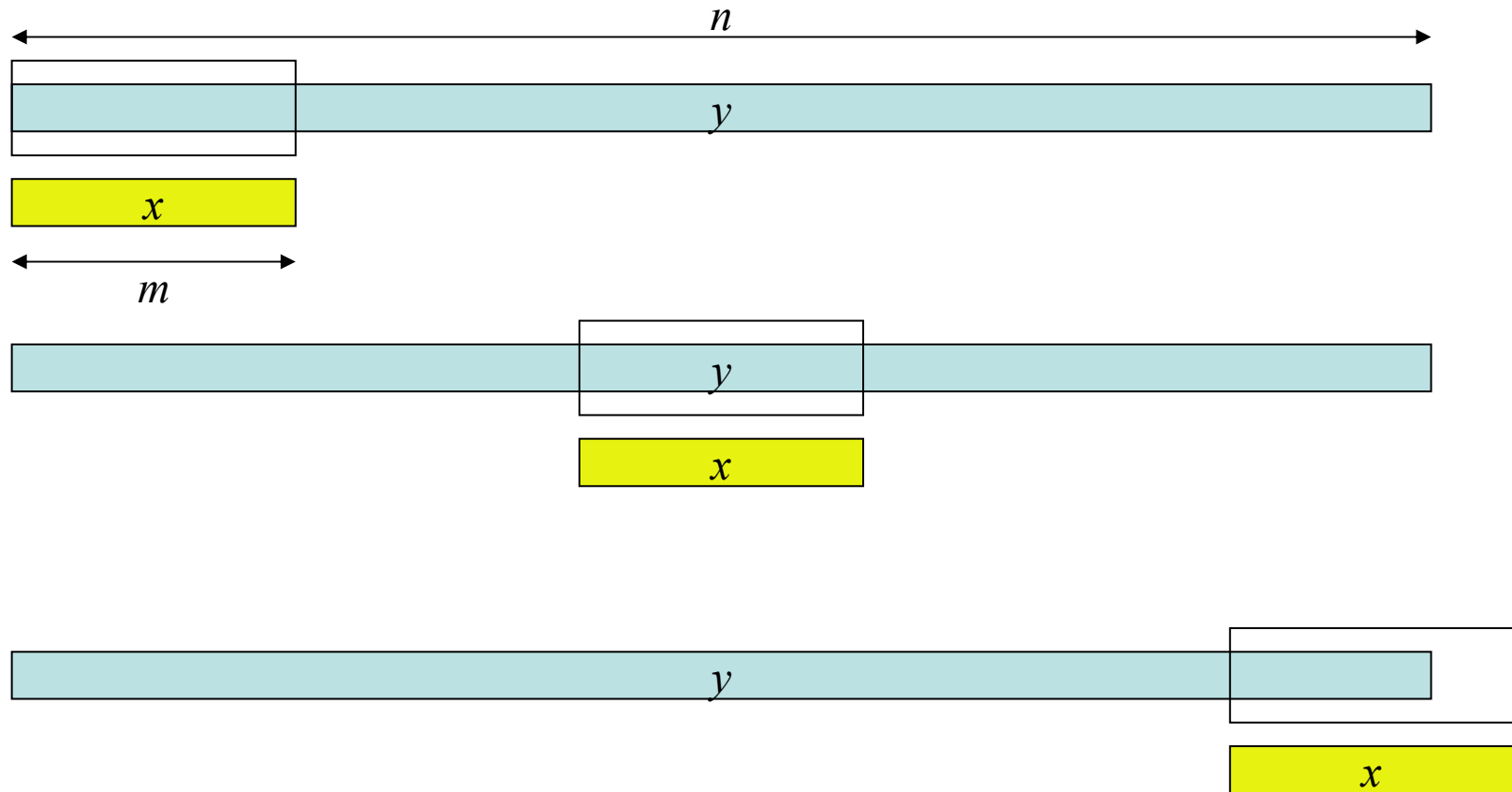
Recherche d'un mot

Le problème de la recherche d'un mot x de longueur m dans un texte y de longueur n consiste à signaler toutes les occurrences de x dans y .

Ce problème admet deux variantes :

1. le mot x est connu à l'avance et peut subir un prétraitement. En général, les solutions à cette variante ont une phase de prétraitement en $O(m)$ et une phase de recherche en $O(n)$.
2. le mot y est connu à l'avance et peut subir un prétraitement. En général, les solutions à cette variante ont une phase de prétraitement en $O(n)$ et une phase de recherche en $O(m)$.

Fenêtre glissante



Un algorithme de recherche exacte de mot est
une succession de

- tentatives (traitements consistant à comparer le contenu de la fenêtre et le mot) ;
- décalages (de la fenêtre vers la droite).

Un décalage de longueur $d \geq 1$ après une tentative à la position gauche j est dit valide si on est assuré qu'il n'y a pas d'occurrence du mot commençant aux positions $j+1, j+2, \dots, j+d-1$ sur y .

Algorithme naïf

Il est caractériser par des décalages de longueur exactement 1.

algo LOCALISER-NAIVEMENT(x, m, y, n)

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

si $x = y[j..j+m-1]$ **alors**

signaler une occurrence de x

Algorithme naïf

Complexité

- temps : $O(mn)$;
- espace : $O(1)$ en plus de x et y .

Proposition 1

Si $\text{card } A > 1$ et que la distribution des lettres de l'alphabet est uniforme et indépendante, le nombre moyen de comparaisons de lettres effectuées par l'algorithme LOCALISER-NAIVEMENT(x, m, y, n) est $\Theta(n-m)$.

Preuve

Soit $c = \text{card } A$.

Le nombre de comparaisons pour déterminer si deux mots u et v de même longueur m sont identiques est

$$1 + 1/c + 1/c^2 + \cdots + 1/c^{m-1}$$

indépendamment de la permutation sur les positions suivant laquelle les comparaisons entre les lettres sont effectuées.

Si $c \geq 2$ cette quantité est inférieure à

$$1/(1-1/c) \geq 2.$$

Il s'ensuit que le nombre moyen de comparaisons effectuées par l'algorithme LOCALISER-NAIVEMENT(x, m, y, n) est $2(n-m+1)$ puisque cet algorithme effectue exactement $n-m+1$ tentatives. □

Recherche avec l'automate reconnaissant A^*x

L'automate reconnaissant A^*x est l'automate $D(\{x\}) = (A, Q, q_0, T, F)$ où :

- A est l'alphabet ;
- $Q = \text{Préf}(x)$;
- $q_0 = \varepsilon$;
- $T = \{x\}$;
- $F = \{ (u, a, ua) \mid u \in Q, a \in A, ua \in Q, ua \preceq_{\text{préf}} x \} \cup \{ (u, a, \text{Bord}(ua)) \mid u \in Q, a \in A, ua \not\preceq_{\text{préf}} x \}$

```

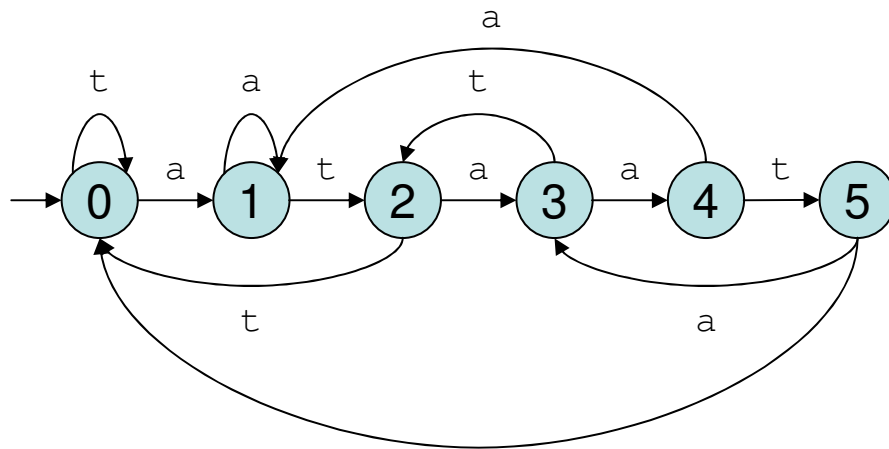
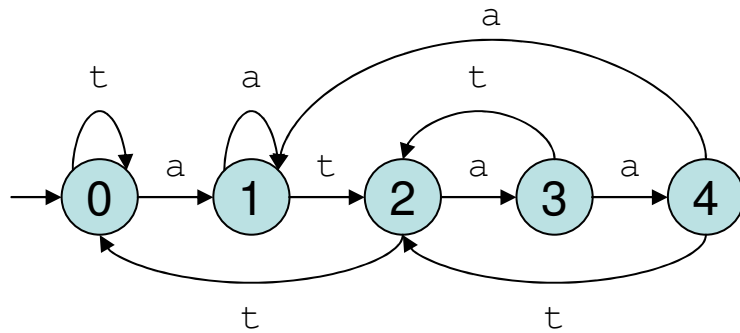
algo ALU-COMPLET( $x, m$ )
   $q_0 \leftarrow$  NOUVEL-ETAT()
   $Q \leftarrow \{q_0\}$ 
  pour chaque lettre  $b \in A$  faire
     $F \leftarrow F \cup \{ (q_0, b, q_0) \}$ 
   $t \leftarrow q_0$ 
  pour  $i \leftarrow 0$  à  $m-1$  faire
     $p \leftarrow$  NOUVEL-ETAT()
     $Q \leftarrow Q \cup \{p\}$ 
     $r \leftarrow$  CIBLE( $t, x[i]$ )
     $F \leftarrow F - \{ (t, x[i], r) \}$ 
     $F \leftarrow F \cup \{ (t, x[i], p) \}$ 
    pour chaque  $(r, a, q) \in F$  faire
       $F \leftarrow F \cup \{ (p, a, q) \}$ 
     $t \leftarrow p$ 
   $T \leftarrow \{p\}$ 

```

```
algo CIBLE( $q, a$ )  
  si il existe  $(q, a, p) \in F$  alors  
    retourner  $p$   
sinon  
  retourner NIL
```

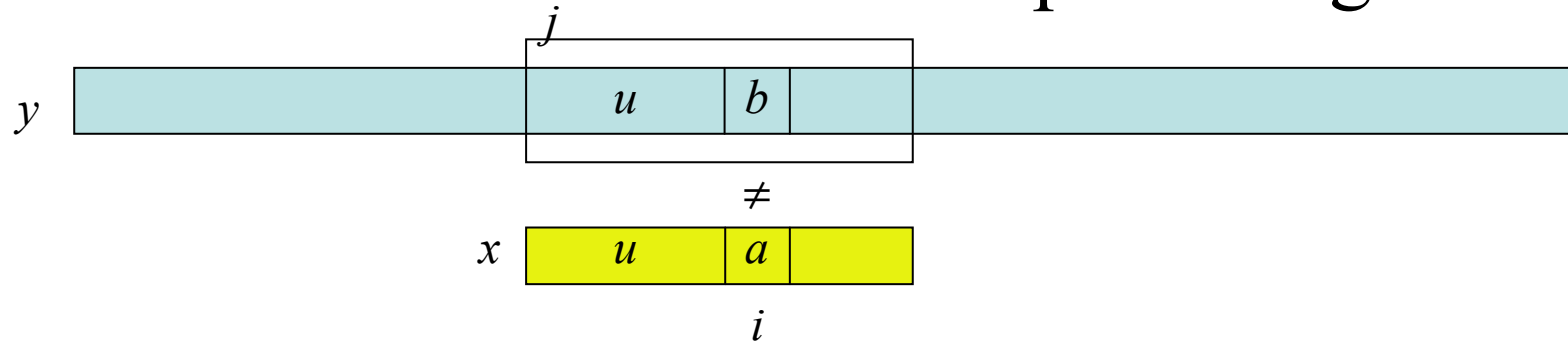
Exemple

$D(\{ataa\})$



L'algorithme de Morris & Pratt (1970)

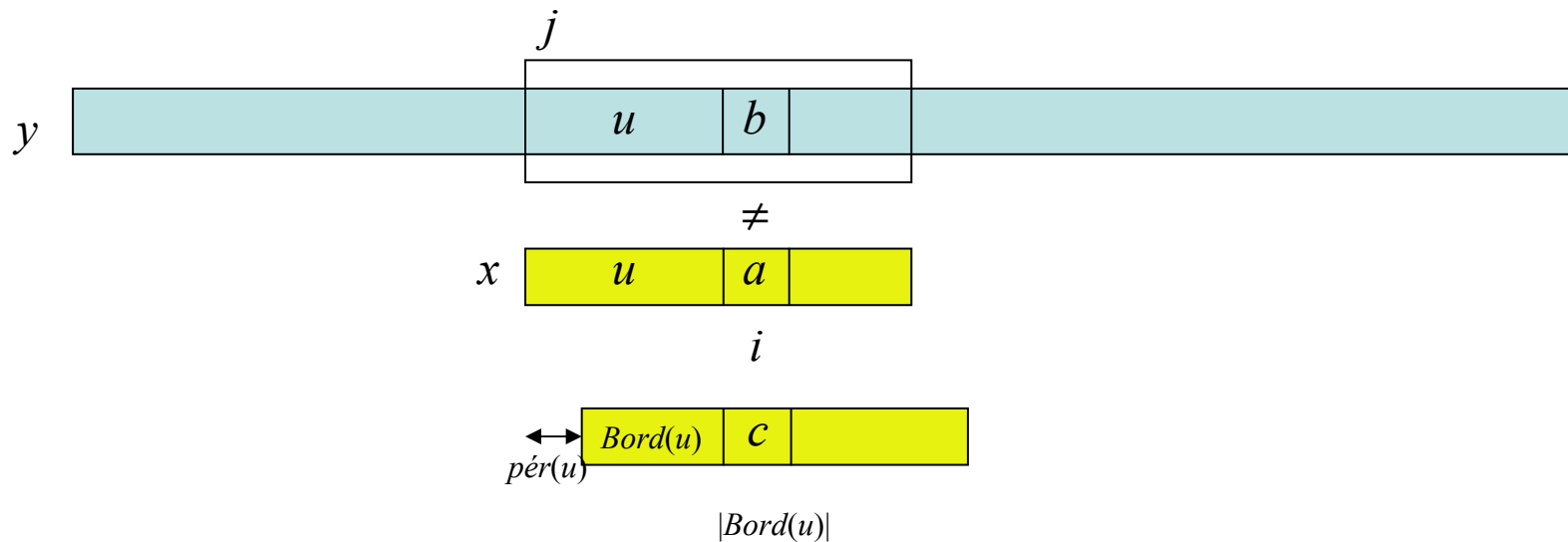
Considérons une tentative à la position gauche j :



On a reconnu un préfixe u de x dans y et on a une inégalité entre une lettre $x[i] = a$ dans le mot et une lettre $y[i+j] = b$ dans le texte.

L'algorithme de Morris & Pratt (1970)

Un décalage valide consiste à décaler la fenêtre de la période de $u = x[0..i-1]$:



et de reprendre les comparaisons entre la lettre qui suit $Bord(u)$ dans x soit $x[|Bord(u)|]$ et la lettre $y[i+j] = b$ dans y .

Soit la table *bon-préf* à $m+1$ éléments définie comme suit, pour $0 \leq i \leq m$

$$\mathit{bon\text{-}préf}[i] = \begin{cases} -1 & \text{si } i = 0, \\ |\mathit{Bord}(x[0..i-1])| & \text{sinon.} \end{cases}$$

algo LOCALISER-SELON-PRÉFIXE1(x, m, y, n)

$i \leftarrow 0$

pour $j \leftarrow 0$ à $n-1$ **faire**

si $i = m$ **alors**

$i \leftarrow \text{bon-préf}[i]$

tantque $i \leftarrow 0$ et $x[i] \neq y[j]$ **faire**

$i \leftarrow \text{bon-préf}[i]$

$i \leftarrow i+1$

si $i = m$ **alors**

 signaler une occurrence de x

Complexité

Théorème 2

L'algorithme LOCALISER-SELON-PRÉFIXE1(x, m, y, n) effectue au plus $2n-1$ comparaisons.

Preuve

Il suffit de considérer la quantité $2j-i$. Cette quantité croît d'au moins une unité après chaque comparaison :

- i et j sont incrémentés de 1 après chaque comparaison positive ;
- j reste inchangé après une comparaison négative alors que i décroît d'au moins une unité.

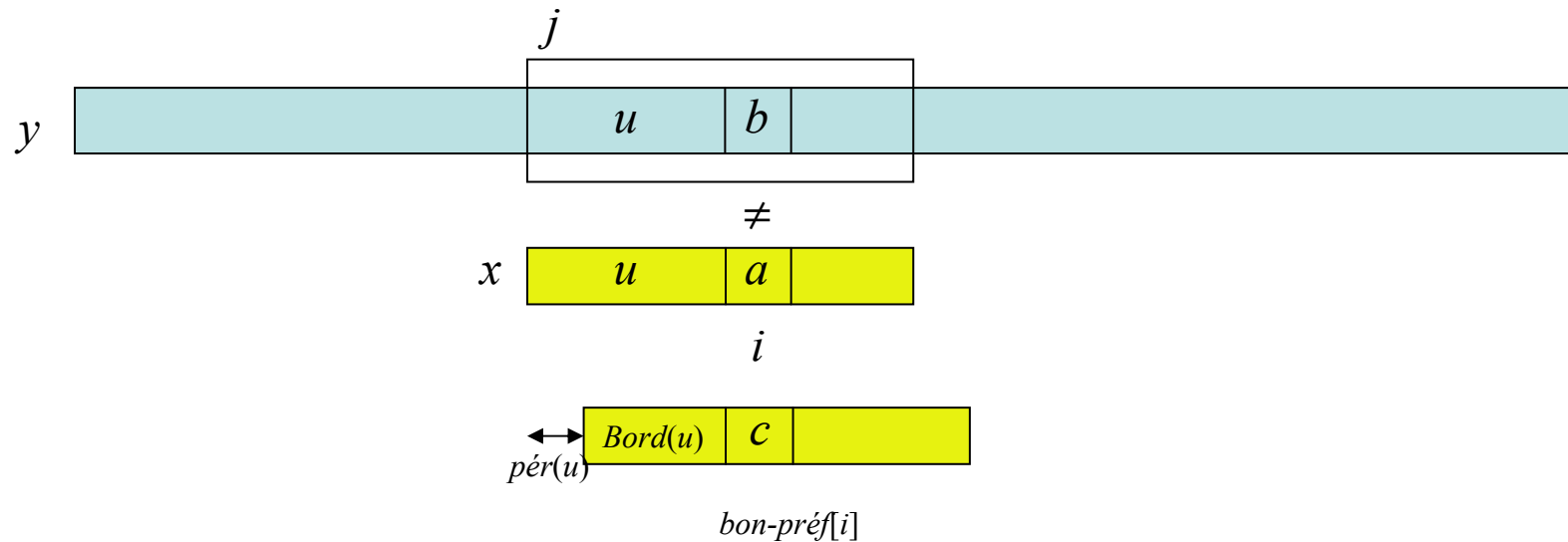
Valeur initiale de $2j-i = 2 \times 0 - 0 = 0$.

Valeur finale maximale de $2j-i = 2(n-1)-0 = 2n-2$.

Donc au plus $2n-1$ comparaisons sont effectués. \square

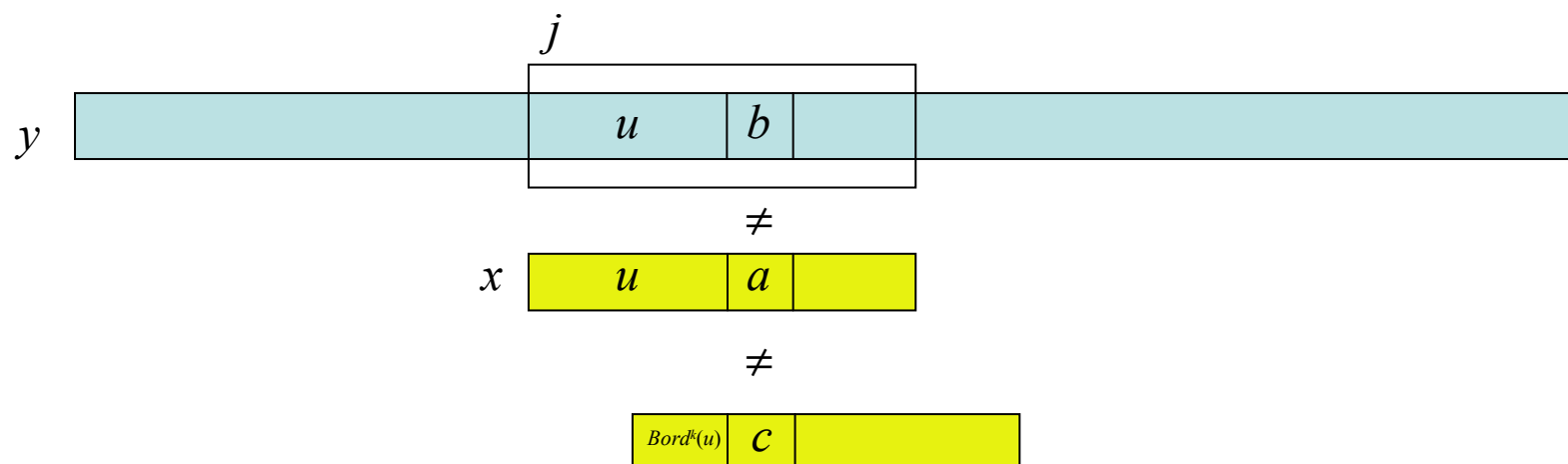
L'algorithme de Knuth, Morris & Pratt (1977)

Dans la situation générale



si $c = a$ alors le résultat de la comparaison entre $x[\text{bon-préf}[i]]$ et $y[i+j]$ est connu à l'avance : il est négatif et la comparaison peut être évitée.

L'algorithme de Knuth, Morris & Pratt (1977)



$$k = \min \{ \ell \mid x[|Bord^\ell(u)|] \neq a \}$$

Pour cela on définit la table *meil-préf* à m éléments de la manière suivante :

$$meil\text{-}préf[i] = \begin{cases} -1 & \text{si } i = 0 \\ |Bord(x[0..i-1])| & \text{si } x[|Bord(x[0..i-1])|] \neq x[i] \\ meil\text{-}préf[|Bord(x[0..i-1])|] & \text{sinon} \end{cases}$$

algo LOCALISER-SELON-PRÉFIXE2(x, m, y, n)

$i \leftarrow 0$

pour $j \leftarrow 0$ à $n-1$ **faire**

si $i = m$ **alors**

$i \leftarrow \text{meil-préf}[i]$

tantque $i \leftarrow 0$ et $x[i] \neq y[j]$ **faire**

$i \leftarrow \text{meil-préf}[i]$

$i \leftarrow i+1$

si $i = m$ **alors**

signaler une occurrence de x

Complexité

Théorème 3

L'algorithme LOCALISER-SELON-PRÉFIXE2(x, m, y, n) effectue au plus $2n-1$ comparaisons.

Preuve

Idem théorème 2.

Calcul des tables *bon-préf* et *meil-préf*

algo BON-PRÉFIXE(x, m)

$bon\text{-}préf[0] \leftarrow -1$

$i \leftarrow 0$

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

$bon\text{-}préf[j] \leftarrow i$

tantque $i \geq 0$ et $x[i] \neq x[j]$ **faire**

$i \leftarrow bon\text{-}préf[i]$

$i \leftarrow i+1$

$bon\text{-}préf[m] \leftarrow i$

retourner $bon\text{-}préf$

Calcul des tables *bon-préf* et *meil-préf*

```
algo MEILLEUR-PRÉFIXE( $x, m$ )  
   $meil\text{-}préf[0] \leftarrow -1$   
   $i \leftarrow 0$   
  pour  $j \leftarrow 1$  à  $m-1$  faire  
    si  $x[i] = x[j]$  alors  
       $meil\text{-}préf[j] \leftarrow meil\text{-}préf[i]$   
    sinon  
       $meil\text{-}préf[j] \leftarrow i$   
    faire  
       $i \leftarrow meil\text{-}préf[i]$   
    tantque  $i \geq 0$  et  $x[i] \neq x[j]$   
   $i \leftarrow i+1$   
   $meil\text{-}préf[m] \leftarrow i$   
  retourner  $meil\text{-}préf$ 
```

Complexité

Théorème 4

Les algorithmes **BON-PRÉFIXE**(x, m) et **MEILLEUR-PRÉFIXE**(x, m) effectuent au plus $2m-3$ comparaisons.

Preuve

Idem théorème 2.

Valeur initiale de $2j-i = 2$.

Valeur finale maximale de $2j-i = 2(m-1)-0 = 2m-2$.

Donc au plus $2m-3$ comparaisons sont effectués. \square