

# Cours n° 2 : Preprocessing

Christophe Gonzales



HUGO3 — Algorithmique

## *Preprocessing*

- ▶ Problème  $P$  à résoudre
- ▶ Idée : commencer par résoudre un problème  $Q$
- ▶ Utiliser  $Q$  pour résoudre plus rapidement/facilement  $P$

## Illustrations du principe :

- ▶ Recherche de motifs (chaînes de caractères)
- ▶ Tris linéaires de tableaux
- ▶ Compression de données (Huffman)

## Problème

- ▶ Chaîne  $T =$ 

b	a	c	b	a	b	a	b	a	a	b	a	b	a	c	a	a
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
- ▶ Chaîne  $P =$ 

a	b	a	b	a	c	a
---	---	---	---	---	---	---
- ▶ Problème : Rechercher la 1ère occurrence de  $P$  dans  $T$ .

## ▶ Exemple :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
b	a	c	b	a	b	a	b	a	a	b	a	b	a	c	a	a
									a	b	a	b	a	c	a	

Réponse : index 9

# Algorithme naïf (1/2)

$T =$ 

b	a	c	b	a	b	a	b	a	a	b	a	b	a	c	a	a
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$P =$ 

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

 égalité observée

 inégalité observée

```
1 fonction recherche(chaine T, chaine P) :  
2   taille_T ← longueur(T)  
3   taille_P ← longueur(P)  
4   i ← 0  
5   tant que i + taille_P < taille_T faire  
6     j ← 0  
7     tant que j < taille_P et T[i+j] = P[j] faire  
8       j ← j + 1  
9     fait  
10    si j = taille_P alors  
11      retourner i  
12    finsi  
13    i ← i + 1  
14  fait  
15  retourner -1
```

Complexité :

# Exécution de l'algorithme de Knuth-Morris-Pratt

$T =$ 

b	a	c	b	a	b	a	b	a	a	b	a	b	a	c	a	a
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$P =$ 

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

a	b	a	b	a	c	a
---	---	---	---	---	---	---

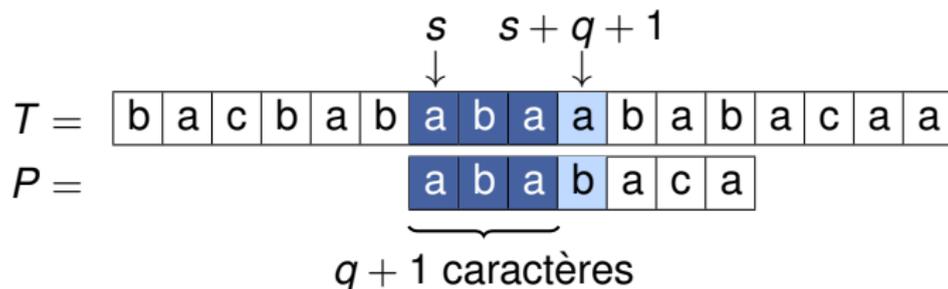
a
---

 égalité observée

a
---

 inégalité observée

# Idee de l'algorithme de Knuth-Morris-Pratt



- ▶ **Notation :**  $T[s..s + q]$  = caractères d'indices  $s$  à  $s + q$  (inclus) de  $T$

## Meilleur prochain décalage

- ▶ Supposons que  $P[0..q] = T[s..s + q]$  mais  $P[q + 1] \neq T[s + q + 1]$
- ▶ Meilleur prochain décalage = le plus petit  $s' > s$  tel que :
  - ▶  $s' + k' = s + q$
  - ▶  $P[0..k'] = T[s'..s' + k']$
- ▶ ici :  $s' = s + 2$  et  $k' = 0$

## *Meilleur prochain décalage*

- ▶ Supposons que  $P[0..q] = T[s..s + q]$  mais  
 $P[q + 1] \neq T[s + q + 1]$
- ▶ Meilleur prochain décalage = le plus petit  $s' > s$  tel que :
  - 1  $s' + k' = s + q$
  - 2  $P[0..k'] = T[s'..s' + k']$
- ▶ Décalage de  $s''$  tel que  $s < s'' < s'$  :

Soit  $k''$  t.q.  $s'' + k'' = s + q$

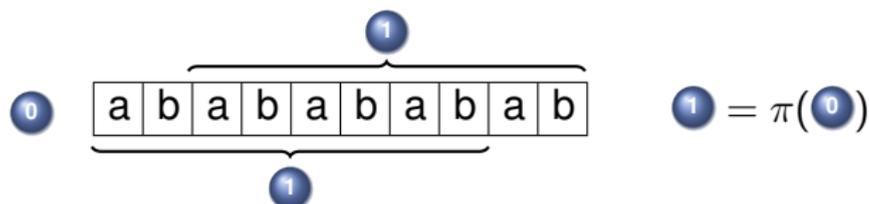
Si  $P[0..k''] = T[s''..s'' + k'']$  alors  $s'$  n'est pas le plus petit  
 $\implies$  il existe  $k < k''$  t.q.  $P[k] \neq T[s'' + k]$   
 $\implies s'$  ne peut pas être une solution
- ▶ Décalage de  $s''$  tel que  $s' < s''$  :

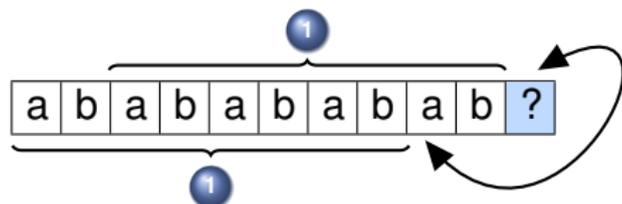
Facile de trouver des exemples où on manque la solution recherchée



# Propriétés du prochain décalage (2/3)

- ▶  $S$  : chaîne de caractères
- ▶  $\pi(S)$  : plus grand suffixe de  $S$ , différent de  $S$ , qui est aussi préfixe de  $S$





## Calcul itératif de $\pi$

- ▶  $S$  = chaîne de caractères
- ▶  $\textcircled{1} = \pi(S)$  = plus grand suffixe qui est préfixe
- ▶  $S' = S \cup$  nouveau caractère ?
- ▶ Si ? = le caractère suivant de  $\textcircled{1}$  alors  $\pi(S') = \pi(S) \cup ?$
- ▶ Sinon  $\textcircled{2} = \pi(\textcircled{1})$
- ▶ Si ? = le caractère suivant de  $\textcircled{2}$  alors  $\pi(S') = \pi(\textcircled{1}) \cup ?$
- ▶ Sinon  $\textcircled{3} = \pi(\textcircled{2})$ , etc.

# Détermination du plus grand suffixe $P[r..q]$

- ▶  $P$  : chaîne de caractères,  $P_k = P[0..k]$
- ▶  $L[k]$  = longueur de  $\pi(P_k)$

```
1 fonction longueur_prefixe_KMP (chaîne P) :
2   m ← longueur(P)
3   L ← tableau de m entiers
4   L[0] ← -1 # 1er caractère : pas de  $\pi$ 
5   i ← -1    # la longueur du  $\pi$  de  $P[0..j-1]$ 
6
7   # ici, on calcule le  $\pi$  de  $P[0..j]$ 
8   pour j variant de 1 à m faire
9     tant que i ≥ 0 et P[j] ≠ P[i+1] faire
10      i ← L[i]
11     fait
12     si P[j] = P[i+1] alors
13       i ← i + 1 # on rajoute le caractère j à  $\pi$ 
14     finsi
15     L[j] ← i
16   fait
17
18   retourner L
```

Complexité =

# Algorithme de Knuth-Morris-Pratt

```
1  fonction recherche_KMP (chaîne T, chaîne P) :
2    n ← longueur(T)
3    m ← longueur(P)
4    L ← longueur_prefixe_KMP(P)
5    q ← -1
6
7    pour i variant de 0 à n-1 faire
8      tant que q ≥ 0 et P[q+1] ≠ T[i] faire
9        q ← L[q]
10     fait
11     si P[q+1] = T[i] alors
12       q ← q+1
13     finsi
14     si q = m alors
15       retourner "chaîne trouvée à l'indice i - m"
16     finsi
17   fait
18
19   retourner "chaîne non trouvée"
```

Complexité =

- ▶ Algorithme naïf :  $O(m \times n)$
- ▶ Algorithme de Knuth-Morris-Pratt :  $O(m + n)$

⇒ Préprocessing accélère significativement les calculs

# Tri linéaire de tableaux

- ▶ But : trier les éléments d'un tableau
- ▶ Très souvent utile en programmation
- ▶ Algorithme efficace dans le cas général : cours 3

## Cas particulier :

- ▶ Tableau  $T$  contenant  $n$  entiers entre 0 et  $k$
- ▶  $k$  très inférieur à  $n$

## Preprocess :

- ▶ Créer un tableau  $C$  de  $(k + 1)$  cellules
- ▶ Placer dans  $C[i]$ ,  $0 \leq i \leq k$  le nombre d'occurrences de  $i$  dans  $T$

## Exploitation du preprocess :

- ▶ Parcourir  $C$  pour remplir le tableau  $T$  trié

# Algorithme de tri linéaire de tableaux

```
1  fonction counting_sort (tableau T, k):
2    créer tableau C de taille (k+1)
3    pour i variant de 0 à k faire
4      C[i] ← 0 # initialisation des comptages
5    fait
6
7    # calcul des nombres d'occurrence
8    n ← longueur(T)
9    pour i variant de 0 à n-1 faire
10     C[T[i]] ← C[T[i]] + 1 # occurrence de T[i]
11   fait
12
13   # remplissage du tableau T trié
14   j ← 0
15   r ← 0
16   pour i variant de 0 à k faire
17     r ← r + C[i]
18     tant que j < r faire
19       T[j] ← i
20       j ← j+1
21   fait
22   fait
```

Complexité :  $O(n)$