Algorithmes sur les mots (séquences)

Introduction

Algorithmes sur les mots (textes, séquences, chaines de caractères)

Nombreuses applications :
- bases de données bibliographiques
- bioinformatique (séquences de biomolécules)
- sécurité informatique

Livres :
- M.Crochemore, C.Hancart, T.Lecroq, Algorithmique du texte, Vuibert 2001
http://igm.univ-mlv.fr/~mac/CHL/CHL.html

Recherche d'un motif dans un texte

T[1..*n*] texte
P[1..*m*] motif
Tâche : localiser toutes les occurrences de P dans T (variantes : vérifier si P apparait dans T, compte le nombre d'occurrences)

Algorithme naïf : O($n \cdot m$)

T=aaaaaa…aa=a^n
P=aa…ab=$a^{m-1}b$

Recherche d'un motif dans un texte

T = a b a b a a a b a b a a b a …
P = a b a a b

Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …

P = a b a a b

Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …

P = a b a a b

Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …

P = a b a a b

Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …

P = a b a a b

Recherche d'un motif dans un texte

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b

Recherche d'un motif dans un texte

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b

Recherche d'un motif dans un texte

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b

Recherche d'un motif dans un texte

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b

Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b
a b a a b

Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b
a b a a b

Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b
a b a a b

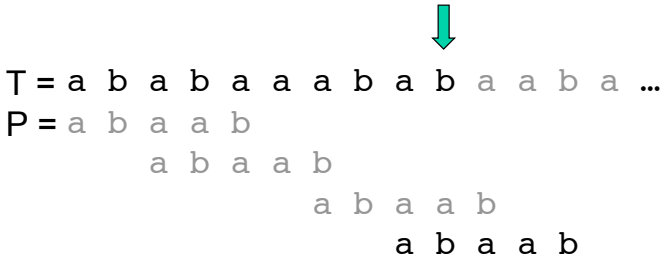
Recherche d'un motif dans un texte

UMLV ©

T = a b a b a a a b a b a a b a …
P = a b a a b
a b a a b
a b a a b

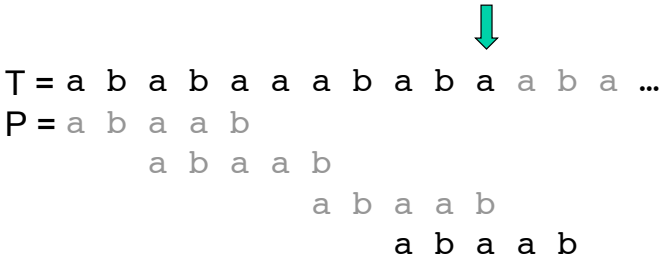
Recherche d'un motif dans un texte

UMLV ©



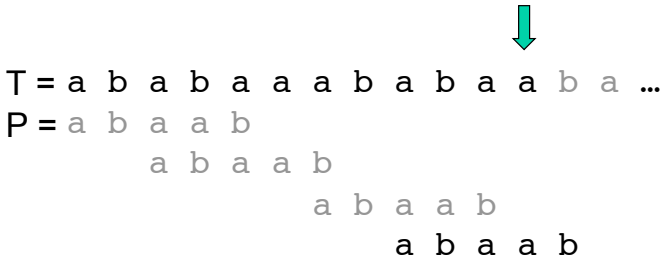
Recherche d'un motif dans un texte

UMLV ©



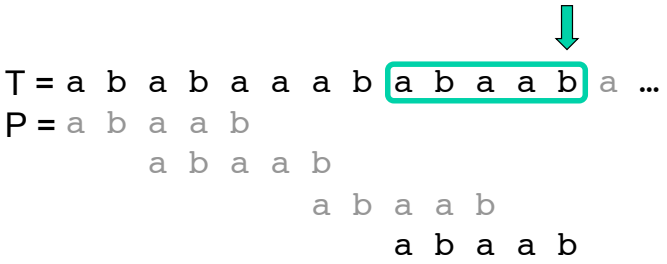
Recherche d'un motif dans un texte

UMLV ©



Recherche d'un motif dans un texte

UMLV ©



Recherche d'un motif dans un texte

```
T = a b a b a a a b a b a a b a …
P = a b a a b
      a b a a b
          a b a a b
              a b a a b
                  a b a a b
```


Recherche d'un motif dans un texte

```
T = a b a b a a a b a b a a b a …
P = a b a a b
      a b a a b
          a b a a b
              a b a a b
                  a b a a b
```


Recherche d'un motif dans un texte

Quelques observations :
- à chaque pas on progresse d'une lettre dans le texte
- l'état de la recherche est défini par les positions dans le texte et dans le motif
- le décalage du motif est défini en fonction de la *position d'échec* dans le motif et la lettre du texte qui a provoqué l'échec

```
T = … * * * * * * * * …
        = = = ≠
P =     * * * * *
```


Recherche d'un motif dans un texte

Quelques observations :
- à chaque pas on progresse d'une lettre dans le texte
- l'état de la recherche est défini par les positions dans le texte et dans le motif
- le décalage du motif est défini en fonction de la *position d'échec* dans le motif et la lettre du texte qui a provoqué l'échec

```
T = … * * * * * * * * …
        = = = ≠
P =     * * * * *
```

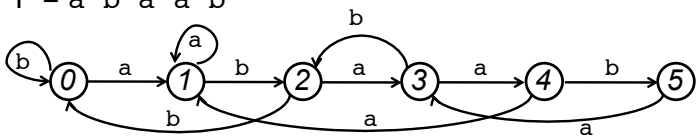
⇒ Automate fini !

Recherche d'un motif dans un texte

P ⇒ automate fini
état ≡ préfixe de P (position) $q \equiv P_q = P[1..q]$

$$\sigma(q,a) = \begin{cases} q+1, \text{ si } a=P[q+1] \\ max\{\ i \mid P_i \text{ est un suffixe de } P_q a\ \} \text{ sinon} \end{cases}$$

P = a b a a b



Invariant: P_q est le préfixe maximal de P qui est un suffixe de la partie lue de T

Recherche d'un motif dans un texte

Algorithme qui en résulte :
1. *Prétraitement* : calculer l'automate O($m \cdot |A|$)
2. *Recherche* : faire marcher l'automate sur le texte O(n)

But : obtenir un algorithme ne dépendant pas de la taille d'alphabet

Algorithme de Knuth-Morris-Pratt

Algorithme de Knuth-Morris-Pratt

```
T = … a b a b * * * * …
          = = = ≠
P =       b a b a b
```

Quels sont les décalages possibles du motif ?

Fonction d'échec (*de suppléance*) :
$f(q) = max\{\ k \mid k<q$ et $P_k = P[1..k]$ est un suffixe de $P_q\ \}$

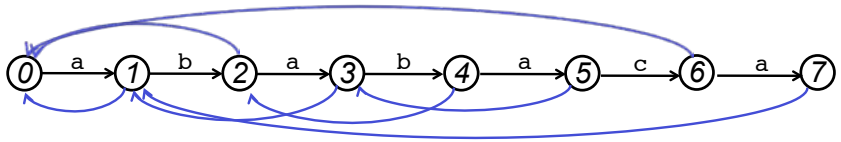
P = a b a b a c a

q	0	1	2	3	4	5	6	7
$f(q)$	-1	0	0	1	2	3	0	1

P = a b a b a c a

q	0	1	2	3	4	5	6	7
$f(q)$	-1	0	0	1	2	3	0	1

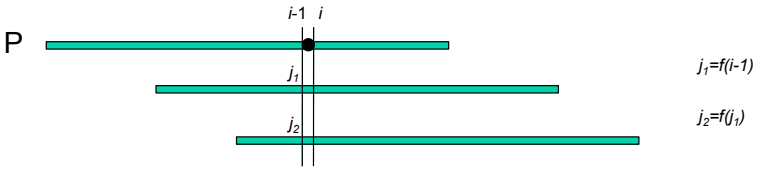
Automate de Knuth-Morris-Pratt



Une fois la fonction *f* calculée …

```
KMP(T[1..n],f)
        j=0    /* pointeur dans P */
        for i=1 to n do
                while j≥0 and P[j+1]≠T[i] do
                        j=f(j) endwhile
                j=j+1
                if j==m then
                        output(occurrence de P à la position (i-m))
                        j=f(j) endif
        endfor
```


Comment calculer la fonction d'échec



Décaler le motif par rapport à lui-même jusqu'à ce que $P[j_q+1]=P[i]$

Calcul de la fonction d'échec

```
FE(P[1..m])
        f[0]=-1
        f[1]=0
        k=0
        for j=2 to m do
                while k≥0 and P[k+1]≠P[j] do
                        k=f(k) endwhile
                k=k+1
                f(j)=k
        endfor
```

Version optimisée (KMP vs MP)

```
T = … a b a * * * * * …
        = = ≠
P =   a b a b a c a
          a b a b a c a    ⇐ décalage inutile
```

$h(3)=0$ h fonction d'échec optimisée

q	0	1	2	3	4	5	6	7
$f(q)$	-1	0	0	1	2	3	0	1
$h(q)$	-1	0	0	0	0	3	0	1

Calcul de la fonction d'échec ***optimisée*** h

FE(P[1..m])
    $h[0]=-1$
    $h[1]=0$
    $k=0$
    for $j=2$ **to** m **do**
        while $k\geq 0$ **and** $P[k+1]\neq P[j]$ **do**
            $k=h(k)$ **endwhile**
        $k=k+1$
        ~~$f(j)=k$~~ → **if** $P[j]\neq P[k]$ **then** $h(j)=h(k)$ **else** $h(j)=k$
    endfor

Différence avec l'approche automate : on peut « piétiner » sur une position de texte

au plus $\log_\varphi m$ décalages sur une position, où $\varphi=(1+\sqrt{5})/2$ nombre d'or

Complexité amortie

$O(n)$ pour la recherche (KMP)
$O(m)$ pour le pré-traitement (FE)

Historique : Morris-Pratt (1969), Knuth (1970)

Algorithme de Aho-Corasick

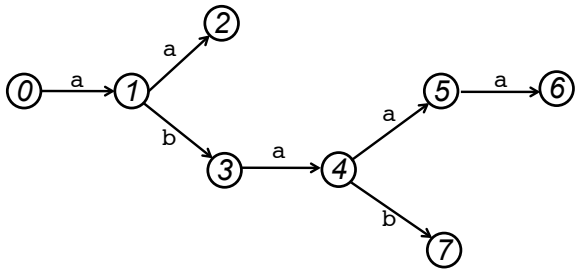
Algorithme de Aho-Corasick

Les idées de KMP peuvent se généraliser pour la *recherche simultanée de plusieurs motifs* ⇒Algorithme de Aho-Corasick (1974)

Algorithme de Aho-Corasick

S={aa, abaaa, abab}

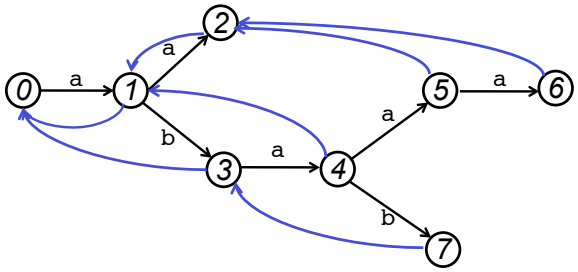
arbre digital (trie)



Algorithme de Aho-Corasick

S={aa, abaaa, abab}

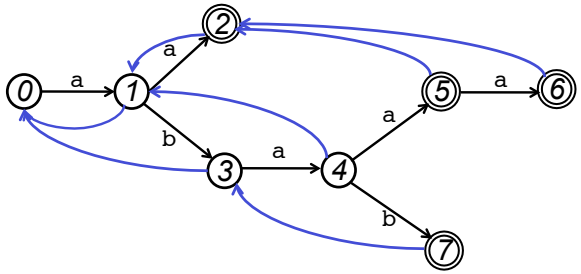
arbre digital (trie) + fonction d'échec



Algorithme de Aho-Corasick

S={aa, abaaa, abab}

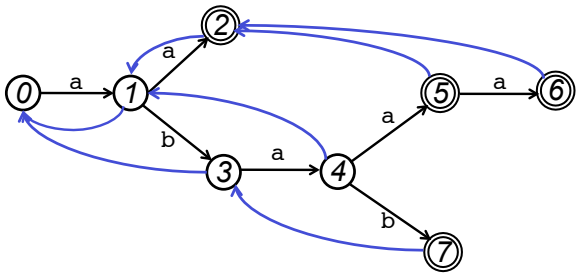
arbre digital (trie) + fonction d'échec + états finaux



Algorithme de Aho-Corasick

S={aa, abaaa, abab}

arbre digital (trie) + fonction d'échec + états finaux =
Automate de Aho-Corasick (AC)



L'automate de AC peut être construit en temps O(*m*) où *m* est la taille **totale** des mots de S

Algorithme de Karp-Rabin (1987)

Algorithme de Karp-Rabin

- soit A={0,1,2,3,4,5,6,7,8,9}
- chaque mot P[1..*m*] peut être encodé par un nombre:
  $p=P[1]\cdot 10^{m-1}+P[2]\cdot 10^{m-2}+\ldots+P[m-1]\cdot 10+P[m]$
- p peut être calculé en temps O(*m*) avec le schéma de Horner :
  $p=P[m]+10\cdot(P[m-1]+10\cdot(P[m-2]+\ldots+10\cdot(P[2]+10\cdot P[1])\ldots))$
- idée générale :
  - comparer p successivement avec les codages t_i de T[*i*..*i*+*m*-1], pour i=1..*n*-*m*+1
  - le passage de t_i *à* t_{i+1} se fait en temps constant :
    $t_{i+1}=10\cdot(t_i-10^{m-1}\cdot T[i])+T[i+m]$ (en supposant que 10^{m-1} est pré-calculé)

Algorithme de Karp-Rabin

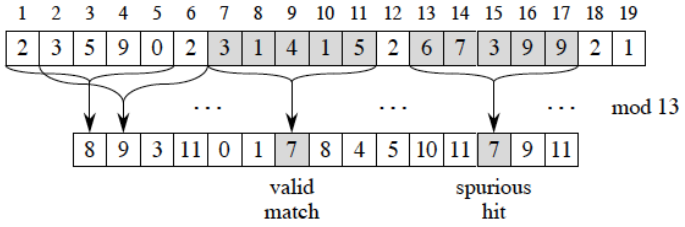
- soit A={0,1,2,3,4,5,6,7,8,9}
- chaque mot P[1..*m*] peut être encodé par un nombre:
  $p=P[1]\cdot 10^{m-1}+P[2]\cdot 10^{m-2}+\ldots+P[m-1]\cdot 10+P[m]$
- p peut être calculé en temps O(*m*) avec le schéma de Horner :
  $p=P[m]+10\cdot(P[m-1]+10\cdot(P[m-2]+\ldots+10\cdot(P[2]+10\cdot P[1])\ldots))$
- idée générale :
  - comparer p successivement avec les codages t_i de T[*i*..*i*+*m*-1], pour i=1..*n*-*m*+1
  - le passage de t_i *à* t_{i+1} se fait en temps constant :
    $t_{i+1}=10\cdot(t_i-10^{m-1}\cdot T[i])+T[i+m]$ (en supposant que 10^{m-1} est pré-calculé)

- *Problème* : les nombres manipulés peuvent être très grands
- *Parade* : manipuler les nombres modulo *q*

Algorithme de Karp-Rabin

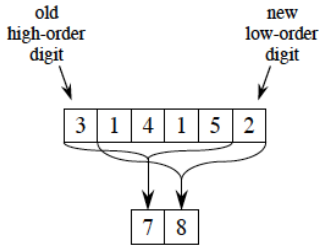
• on a alors
$t_{i+1}=10\cdot(t_i - h\cdot T[i])+T[i+m]$ modulo q, où $h=10^{m-1}$ modulo q

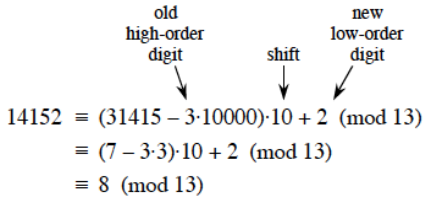
• *attention, on peut avoir des « faux positifs »* !



Algorithme de Karp-Rabin

• passage de t_i *à* t_{i+1} (illustration) :





• dès qu'on trouve une fenêtre candidate (t_i=p), on vérifie en comparant P et T[i..i+m-1] caractère par caractère

Algorithme de Karp-Rabin

• l'algorithme nécessite un pré-traitement en O(*m*) et marche en temps O(*mn*) dans le pire des cas

• en moyenne il marche en temps $O(n + m\cdot(v+n/q))$ où v est le nombre d'occurrences du motif

• très facile à implémenter, plus efficace que l'algorithme naïf