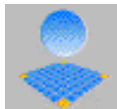


# Algorithmes de recherche de motifs

Eric Rivals

LIRMM - Méthodes Algorithmiques pour la Bioinfo

[www.lirmm.fr/~rivals](http://www.lirmm.fr/~rivals)



## Recherche d'un motif

Soit  $\Sigma$  un **alphabet**. Étant donné un **texte**  $T$  de longueur  $n$  et un **motif**  $M$  de longueur  $m$  tous deux sur  $\Sigma$ , trouver toutes les positions de début/fin des occurrences de  $M$  dans  $T$ . En général,  $m \ll n$ .

### Exemple :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T :	C	T	G	T	G	T	G	T	A	C	A	T	G	T	G
		T	G	T	G										
				T	G	T	G								
												T	G	T	G

Solutions :  $\{2, 4, 12\}$  ou  $\{5, 7, 15\}$

# Intérêts

- **Pratiques** :
  - recherche de mots dans des documents
  - analyse lexicale
  - recherche motifs avec erreurs
  - comparaison de séquences biologiques
- **Théoriques** : pédagogique, conception et analyse des algorithmes, mesure des performances, combinatoire des mots
- **En biologie** : recherche de motifs et signaux de fonctions connues ou putatives, ex : TATA box, motifs PROSITE, motifs de régulation des gènes, de sites de clivage de l'ADN, etc.
- Importance de la méthode : Temps de recherche avec *GCG* d'un motif de 30 bp dans Genbank de 4 h, et  $< 1$  min avec d'autres logiciels.

# Notations

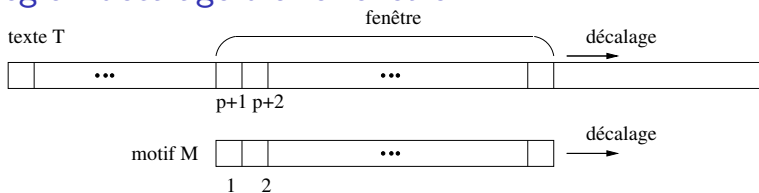
Soit  $M$  un **mot** de longueur  $m$  sur  $\Sigma$ .  $M$  est aussi appelé texte ou chaîne de caractères (en anglais **string**). Les lettres de  $M$  sont indicées de 1 à  $m$ . Pour tout entier  $i$  tel que  $0 < i \leq m$  on note  $M[i]$  la  $i$ ème lettre de  $M$ .

**Définition** : un **facteur** de  $M$  est une suite contiguë de lettres de  $M$ . Pour deux entiers  $i, j$  tels que  $0 < i \leq j \leq m$ , on note  $M[i..j]$  le facteur allant des positions  $i$  à  $j$  dans  $M$ .

Un mot  $u$  est facteur de  $M$  s'il existe deux entiers  $i, j$  tels que  $0 < i \leq j \leq m$  et  $M[i..j] = u$ .

**Définitions** : un **préfixe** de  $M$  est un facteur qui débute à la position 1 ; un **suffixe** est un facteur qui finit à la position  $m$ . Un préfixe/suffixe est **propre** s'il est de longueur strictement inférieure à  $m$ .

## Stratégie : décalage d'une fenêtre



mettre fenêtre au début du texte

**tant que** fenêtre dans texte **faire**

*tentative* : comparer fenêtre et motif

**si** fenêtre égale motif **alors**

imprimer position courante

**fsi**

*décalage* : décaler la fenêtre vers la droite

et mémoriser des informations

**fin**

# Tentative et décalage

- **Tentative** : suite de comparaison de caractères de  $T$  et  $M$ , jusqu'à ce qu'une inégalité ou la fin du motif survienne négative ou positive
- **Décalage** : de longueur  $d$ , avec  $d \geq 1$   
condition de **validité** du décalage : ne rater aucune occurrence qui débute à une position intermédiaire  
correction de l'algorithme

# Algorithme naïf

Décalage de 1 position, pas de prétraitement, ni de mémorisation

---

**Algorithme 1** : Algorithme naïf

---

```
1  $p := 1$ ;  
2 tant que  $(p \leq n - m)$  faire  
3    $c := 1$ ;  
4   tant que  $(c \leq m)$  et  $(M[c] = T[p + c - 1])$  faire  
5      $c := c + 1$ ;  
6   si  $(c = m + 1)$  alors  
7     imprimer  $p$ ;  
8    $p := p + 1$ ;
```

---

## Exemples de décalages

### Tentative échouée

		$p$					
$T$	...	$a$	$a$	$b$	$a$	$b$	...
						X	
$M$		$a$	$a$	$b$	$a$	$c$	
		1	2	3	4	5	
						$c$	

On a :  $T[p + 1..p + 4] = M[1..4]$  et  
 $T[p + 5] \neq M[5]$

Décalage vers la droite : mise en correspondance d'un préfixe de  $M$  avec un suffixe de  $T[p + 1..p + 4]$ , c.a.d. de  $M[1..c - 1]$ .

### Décalage de 1 position vers la droite :

		$p$					
$T$	...	$a$	$a$	$b$	$a$	$b$	...
			X				
$M$		$a$	$a$	$b$	$a$	$c$	
		1	2	3	4	5	
						$c$	

mise en correspondance de  $M[1..3]$  avec  
 $M[2..4]$  : mismatch

## Décalages suite

### Décalage de 2 positions vers la droite :

$T$  ...  $a$   $a$   $b$   $a$   $b$  ...

$p$

X

$M$        $a$   $a$   $b$   $a$   $c$

1 2 3 4 5

$c$

mise en correspondance de  $M[1..2]$   
avec  $M[3..4]$  : mismatch

### Décalage de 3 positions vers la droite :

$T$  ...  $a$   $a$   $b$   $a$   $b$  ...

$p$

| X

$M$        $a$   $a$   $b$   $a$   $c$

1 2 3 4 5

$c$

mise en correspondance de  $M[1]$   
avec  $M[4]$  ; match ou mismatch en  
position  $p + 1$  ?

Les décalages de 1 ou 2 positions provoquent un *mismatch prévisible*,  
mais pas celui de 3 positions. Donc 3 est le *décalage minimal*.

# Recherche d'un ensemble de motifs

## « Set Pattern Matching »

## Recherche de plusieurs motifs

- **Recherche d'un ensemble de motifs** : Soient  $\mathcal{M} = \{M_1, \dots, M_z\}$  un ensemble de  $z$  motifs de longueurs respectives  $m_1, \dots, m_z$ ,  $m := \sum_{i=1}^z m_i$ , et  $T$  un texte de longueur  $n$ . Trouver toutes les occurrences de tous les motifs dans  $T$ .
- Algorithme d'**Aho-Corasick** : Utilisation d'un arbre des motifs, généralisation de l'algorithme Knuth-Morris-Pratt.  
**Hypothèse** : aucun motif n'est préfixe d'un autre.
- **Arbre des motifs** : arbre enraciné dont les arcs sont étiquetés par un caractère et tel que : (i) tous les arcs sortants frères sont étiquetés par des caractères différents, (ii) tout motif  $M_i$  est associé à un noeud  $v$  tel que les étiquêtes des arcs sur le chemin de la racine à  $v$  épellent  $M_i$  (iii) chaque feuille est associée à un motif. La feuille associée au motif  $M_i$  est étiquetée par  $i$ .

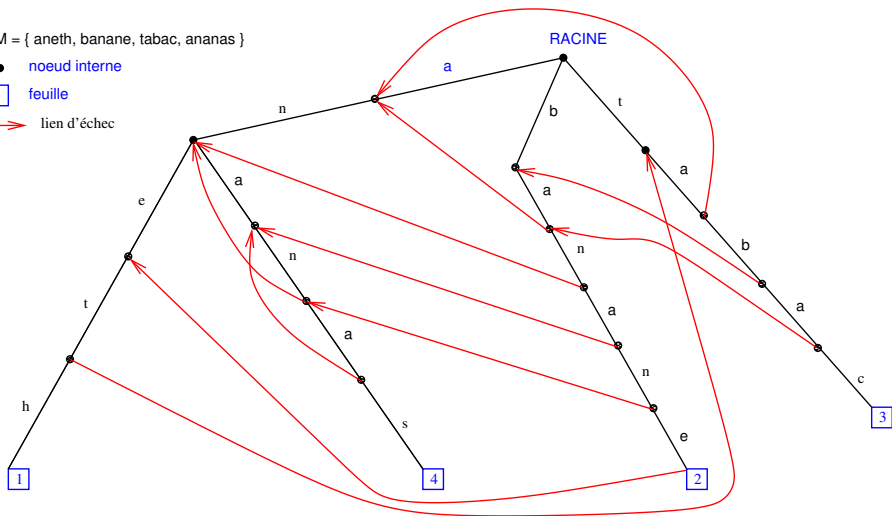


$M = \{ \text{aneth, banane, tabac, ananas} \}$

● noeud interne

□ feuille

→ lien d'échec



## Interface avec l'arbre

- Stockage des motifs dans un tableau de chaîne de caractères :

**Motifs**[1..z]

- Structure d'une variable de type noeud :

noeud{

**Fils**[ $\Sigma$ ] : liens vers les fils du noeud indicés par la lettre de l'arc

**Mot** : indice d'un motif dans le tableau **Motifs**

**Lg** : longueur du mot représenté sur le chemin de  $R$  au noeud

**Feuille** : booléen indiquant si ce noeud est une feuille

**LS** : lien suffixe / lien d'échec

}

- Noeud racine : **R**
- Notation d'accès aux variables d'un noeud : **noeud.variable**  
Ex : Soit **nc** un noeud, **nc.Feuille**.
- Procédure : Soit  $a \in \Sigma$ . **nc.TestFils**[a] renvoie 1 si **nc.Fils**[a] existe/pointe sur un autre noeud et 0 sinon.

## Recherche des motifs dans un texte

---

### Algorithme 2 : Algorithme naïf

---

**Variable** :  $p$  : position de début de fenêtre,  $nc$  noeud courant;  
 $i$  indice de parcours de la fenêtre à partir de  $p$ ;

```
1  $p := 1$ ;  
2 tant que ( $p \leq n$ ) faire  
3    $nc := R$  ;  $i := 0$ ;  
4   tant que ( $p + i \leq n$ ) et ( $nc.TestFils[T[p + i]]$ ) faire  
5      $i := i + 1$ ;  
6      $nc := nc.Fils[T[p + i]]$ ;  
7   si ( $nc.Feuille$ ) alors  
8     imprimer occurrence de Motifs[ $nc.Mot$ ] à la position  $p$  ds  $T$ ;  
9    $p := p + 1$ ;
```

---

# Construction de l'arbre des motifs

- Construction incrémentale par insertion d'un motif par étape. L'étape  $i$  avec  $i > 1$  insère le motif  $M_i$  dans l'arbre  $A_{i-1}$  pour construire l'arbre  $A_i$ . D'où une série d'arbres  $A_0 := R, A_1, \dots, A_i, \dots, A_z := A$ .
- **Initialisation** : pour obtenir  $A_1$ , créer dans  $A_0$  une unique branche ayant autant d'étiqûetes que de caractères de  $M_1$ .
- **Étape  $i$**  : descendre dans  $A_{i-1}$  à partir de  $R$  pour trouver le plus long préfixe entre  $M_i$  et une branche de  $A_{i-1}$ . Soit  $j$  la taille du préfixe et  $v$  le noeud d'arrivé. Si  $j < |M_i|$  alors ajouter une branche partant de  $v$  qui épellent les  $(m_i - j)$  derniers caractères de  $M_i$  et étiqueter la nouvelle feuille par  $i$ . Si  $j = |M_i|$  alors étiqueter  $v$  par  $i$ .

## Liens suffixes

**Notation** : Soit  $v$  un noeud de  $A$ . Notons  $C(v)$  la chaîne représentée par  $v$ , i.e. la suite des labels sur le chemin de  $R$  à  $v$ .

Soit  $S(v)$  le plus long suffixe propre de  $C(v)$  pour lequel il existe  $i \in [1, z]$  tel que  $S(v)$  soit un préfixe de  $M_i$ .  $S(v)$  peut être le mot vide, et donc il existe toujours.

**Lemme** : il existe un noeud  $w$  de  $A$  qui représente  $S(v)$ , i.e., tq  $C(w) = S(v)$ . Si  $S(v)$  est le mot vide alors  $w$  vaut  $R$ .

**Définition** : le lien suffixe de  $v$  pointe sur  $w$ .

# Recherche d'Aho Corasick

---

**Algorithme 3** : Algorithme Aho Corasick

---

**Entrée** : texte  $T$  de longueur  $n$ , l'ens. de motifs  $\mathcal{M}$  et l'arbre des motifs  $\mathcal{A}$

**Variables** :  $i$  : indice de position courante dans  $T$ ,  $nc$  noeud courant;

```

1  $i := 1$ ;
2  $nc := \text{Racine}(\mathcal{A})$ ;
3 tant que ( $p \leq n$ ) faire
4     tant que ( $nc.\text{TestFils}[T[i]]$ ) faire
5         si ( $nc.\text{Fils}[T[i]]$ ) est une feuille alors
6             imprimer fin d'occurrence de  $\text{Motifs}[nc.\text{Fils}[T[i]].\text{Mot}]$  à la
              position  $i$  ds  $T$ ;
7          $i := i + 1$ ;
8          $nc := nc.\text{Fils}[T[i]]$ ;
9      $nc := nc.\text{LS}$ ;
```

---

## Calcul d'un lien suffixe

Soient  $v$  un noeud différent de  $R$ ,  $v'$  son père,  $x$  l'étiquette du lien  $(v', v)$ .

**Lemme** :  $S(v)$  est soit un suffixe de  $S(v')$  suivi par  $x$ , soit le mot vide.

---

**Algorithme 4** : Calcul du lien suffixe de  $v$

---

**Variable** :  $w$  : noeud courant;

```

1  $w := v'.LS;$ 
2 tant que ( $w \neq R$ ) et ( $\text{non } w.TestFils[x]$ ) faire
3    $w := w.LS;$ 
4 si ( $w.TestFils[x]$ ) alors  $v.LS := w.Fils[x];$ 
5 sinon  $v.LS := R;$ 

```

---

Pour tous les noeuds, itérer l'algorithme en effectuant un parcours en profondeur d'abord. Complexité en  $O(m)$ .