Module HMIN121M 27 septembre 2018

Recherche exacte de motif

Algotihme Boyer-Moore

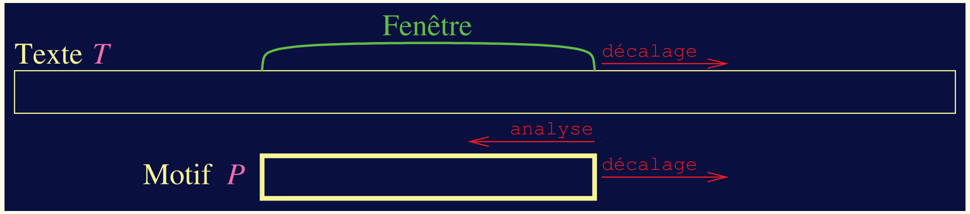
Sèverine Bérard

ISE-M – FDS, Université de Montpellier

Plan

- Introduction
- Règle du mauvais caractère
- Règle du bon suffixe
- Phase de recherche
- Extensions et améliorations

Stratégie de la fenêtre glissante - version BM



Comparaison de la droite vers la gauche

Algorithme 1 : Mécanisme « analyse et décalage » version BM

Données : Deux chaînes T et P de longueurs respectives n et m.

Placer la Fenêtre au début du texte ;

tant que la Fenêtre est sur le texte **faire**

  $u :=$ plus long suffixe commun entre la Fenêtre et le Motif ;

  si $u =$ Motif **alors** le rapporter ;

  Décaler la Fenêtre vers la droite ;

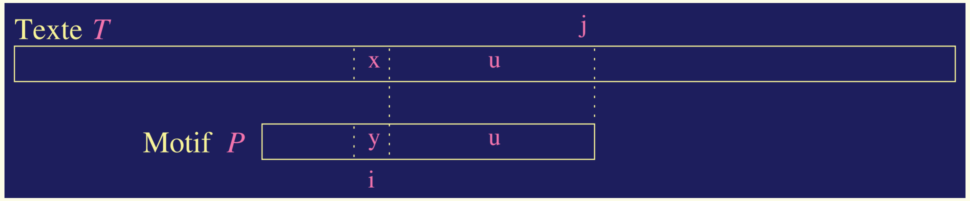
Principes généraux

- Stratégie de la fenêtre glissante, décalage de la fenêtre vers la droite
- Analyse du motif de la droite vers la gauche : on commence par comparer le dernier caractère du motif avec le dernier caractère de la fenêtre
- Plusieurs décalages possibles :
  1. Règle du bon suffixe
  2. Règle du plus long bord
  3. Règle du mauvais caractère

$\Rightarrow$ BM choisit le plus grand !

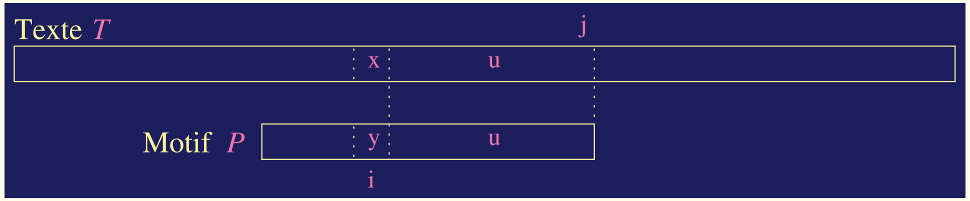
Remarque : les règles 1 et 2 marchent ensembles (*i.e.* quand le « bon suffixe » n'existe pas on choisit le plus long bord)

Situation d'échec



- Situation d'échec : identité des caractères de u mais $x \neq y$
- u est un suffixe de P
- Plus précisément,
  le facteur $u = T[j-m+i+1..j]$ est égal au suffixe $u = P[i+1..m]$
  et la lettre $x = T[j-m+i]$ est différente de la lettre $y = P[i]$

Décalages



- Le décalage consiste à choisir le maximum entre aligner :
  1. en face de $u = T[j-m+i+1..j]$ le facteur zu (z une lettre) de $P[1..m-1]$ le plus à droite ou s'il n'existe pas
  2. à aligner le plus long préfixe de P suffixe de u

     $\Rightarrow$ Règle du bon suffixe
  3. en face de $x = T[j-m+i]$ l'occurrence de x la plus à droite dans P

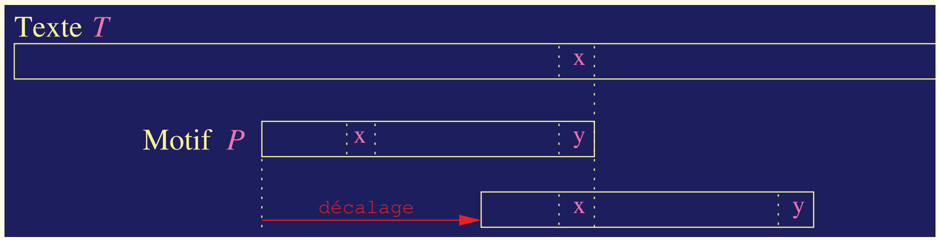
     $\Rightarrow$ Règle du mauvais caractère

Plan

- Introduction
- **Règle du mauvais caractère**
- Règle du bon suffixe
- Phase de recherche
- Extensions et améliorations

Principe de la règle du mauvais caractère

- Supposons que le dernier caractère de P soit y et qu'on tente de l'aligner avec le caractère $x \neq y$ de T
- Si on connaît l'occurrence la plus à droite de x dans P, on peut alors décaler P de manière à aligner ce x avec le x du texte sans rater une occurrence de P dans T



- *Tout décalage plus court conduit à un échec*
- De plus, si x n'apparaît pas dans P, on peut décaler complètement P après l'occurrence de x dans T
  Dans ce cas, certains caractères de T ne sont pas examinés

Calcul et utilisation

Définition 1 (Table R) *Pour chaque caractère x de l'alphabet, soit $R(x)$ la position de l'occurrence la plus à droite de x dans P. $R(x) = 0$ si x n'apparaît pas dans P.*

- Il est facile de pré-traiter P en $O(m)$
- Utilisation dans une situation d'échec : les $m - i$ caractères de P les plus à droite sont identiques au texte mais le caractère suivant $P[i]$ est différent de $T[k]$.

> Règle du mauvais caractère
> P peut-être décalé à droite de $\max(1; i - R(T[k]))$ places

- Autrement dit, **si** l'occurrence la plus à droite de $T[k]$ dans P est à une position $p < i$, **alors** décaler P de manière à ce que la position p soit en dessous de la position k, **sinon** décaler P d'une seule position

Règle du mauvais caractère améliorée

- La règle du mauvais caractère n'a aucun effet si l'occurrence de x est plus à droite que la situation d'échec
- Situation fréquente quand l'alphabet est de petite taille (ADN)

> Règle du mauvais caractère améliorée
> Quand un échec se produit à la position i de P et que le mauvais caractère dans T est x, alors décaler P vers la droite de manière à ce que le x de P le plus proche par la gauche de la position i soit aligné avec le x de T.

- Cette règle améliorée donne des décalages plus grands
- La règle simple utilise seulement $O(\Sigma)$ espace pour la table R et juste une consultation de R par situation d'échec
- On peut implémenter la règle améliorée en $O(m)$ espace
- L'algorithme original de Boyer-Moore utilise la règle simple

Implémentation de la règle améliorée

- Ce pré-traitement doit calculer pour chaque position i de P et chaque caractère x de Σ, la position de la plus proche occurrence de x dans P à gauche de i.
- Approche directe : une table de dimension $m \times |\Sigma|$
  $\Rightarrow$ Consultation rapide mais espace et temps de calcul potentiellement excessif
- Meilleur compromis : examiner P de droite à gauche en conservant pour chaque $x \in \Sigma$ la liste des positions où x apparaît dans P (ces positions seront dans l'ordre décroissant)
- **Exemple :** si $P = abacbabc$, la liste pour le caractère a est $(6, 3, 1)$
- Ces listes sont calculées en temps $O(m)$ et prennent $O(m)$ espace

Utilisation de la règle améliorée

- Pendant la phase de recherche, s'il y a une situation d'échec à la position i de P et que le caractère correspondant dans le texte est x

$\Rightarrow$ Chercher dans la liste de x jusqu'à trouver le premier nombre inférieur à i

  - S'il n'y en en pas, c'est qu'il n'y a pas d'occurrence de x avant i $\Rightarrow$ décaler tout P après la position de x dans T
  - Sinon, le nombre trouvé donne la position désirée de x dans P

- **Remarque :** après un échec à la position i de P, le temps de scanner la liste est au plus $m - i$. Dans le pire des cas cette approche double le temps d'exécution.
  En général c'est moins du double. On peut aussi implémenter une recherche par dichotomie.

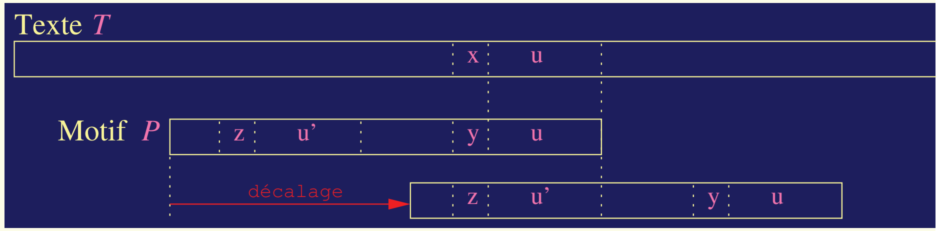
Plan

- Introduction
- Règle du mauvais caractère
- **Règle du bon suffixe**
- Phase de recherche
- Extensions et améliorations

Principe de la règle du bon suffixe

- Supposons un alignement entre P et T tel qu'un facteur u de T soit identique à suffixe u de P mais que les caractères suivants soient différents
- Alors trouver, si elle existe, une copie u' de u la plus à droite dans P telle que
  - u' n'est pas un suffixe de P
  - le caractère à gauche de u' dans P est différent du caractère à gauche de u dans P

  Décaler P de manière à aligner u' avec le u du texte



Principe de la règle du bon suffixe (2)

- Si u' n'existe pas, soit w le plus long préfixe de P suffixe de u. Décaler P de manière à aligner w avec son occurrence dans T, si $w = \epsilon$ décaler P de m places.
  Remarque : $|w| \leqslant |u|$
- Si une occurrence de P est trouvée, décaler P de $period(P)$

> **Théorème**
> L'application de la règle du bon suffixe ne rate jamais une occurrence de P dans T

- L'algorithme original de Boyer-Moore utilise une version plus simple et plus faible de la règle du bon suffixe qui ne spécifie pas que les caractères à gauche de u et u' doivent être différents

Exemple

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T :	p	r	s	t	a	b	s	t	u	b	a	b	v	q	x	r	s	t
										*								
P :			q	c	a	b	d	a	b	d	a	b						
			1	2	3	4	5	6	7	8	9	10						

- Situation d'échec à la position 8 de P et 10 de T
- u=ab et le caractère à gauche de u dans P est d
- u' apparaît donc à la position 3 de P

$\Rightarrow$ P est décalé de 6 positions vers la droite

Exemple

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
T :	p	r	s	t	a	b	s	t	u	b	a	b	v	q	x	r	s	t
										*								
P :									q	c	a	b	d	a	b	d	a	b
									1	2	3	4	5	6	7	8	9	10

- Situation d'échec à la position 8 de P et 10 de T
- u=ab et le caractère à gauche de u dans P est d
- u' apparaît donc à la position 3 de P

⇒ P est décalé de 6 positions vers la droite

- La règle simple aurait décalé P de 3 positions seulement

Implémentation de la règle du bon suffixe

- Règle considérée comme difficile et mystérieuse, même si la version plus faible est facile à comprendre
- La version forte de la règle a été publiée fausse puis corrigée sans explication dans une publication ultérieure. Le code est publié mais sans commentaires. Il manque des références détaillant et prouvant cette partie du pré-traitement de l'algorithme de Boyer-Moore

⇒ Plusieurs approches différentes existent dans la bibliographie. Dans ce cours, on a choisit un *mix* entre [Gusfield, 97] et [Lecroq, 11]

- Deux étapes :
  1. Calcul d'une table de suffixes $suff$
  2. Calcul de la table des décalages D

Calcul d'une table de suffixes $suff$

- Soit P un motif de longueur m
- Pour $1 \leqslant i \leqslant m$, $suff[i]$ est la longueur du plus long suffixe de $P[1..i]$ qui est aussi suffixe de P
- **Exemple :** $P = cabdabdab$, $m = 9$

	1	2	3	4	5	6	7	8	9
P	c	a	b	d	a	b	d	a	b
$suff$	0	0	2	0	0	5	0	0	9

- **Remarque :** Si $P[i] \neq P[m]$, $suff[i] = 0$
- **Remarque :** Que peut-on dire de $P[1..i]$ si $suff[i] = i$ et $i \neq m$?

Algorithme de construction de $suff$

Algorithme 2 : Calcule $suff$

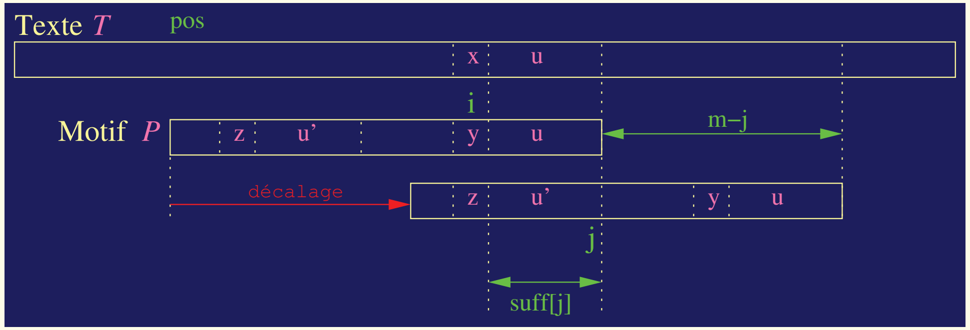
Données : Un mot P de longueur m

```
suff[m] := m ; g := m ;
pour (i de m − 1 à 1) faire
    si (i > g et suff[i + m − f] ≠ i − g) alors
        suff[i] := min(suff[i + m − f], i − g) ;
    sinon
        f := i ; g := min(g, i) ;
        tant que (g > 0 et P[g] = P[g + m − f]) faire
            g := g − 1 ;
        suff[i] := f − g ;
```

- $suff$ est calculé en temps $O(m)$

Calcul de la table D (cas 1)

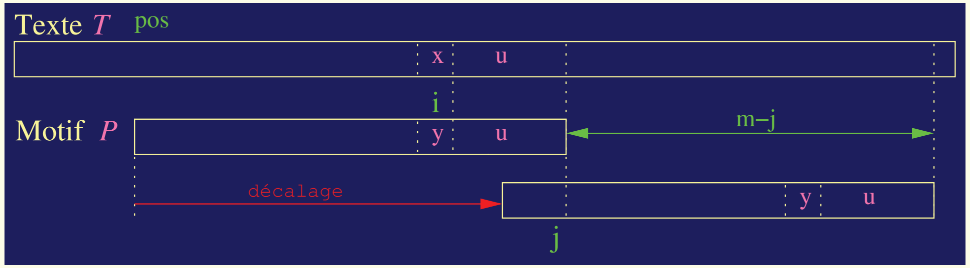
- u' existe dans P



- Situation d'échec à la position i de P
- On cherche la plus grande position j telle que $suff[j] = |u| = m - i$, d'où $i = m - suff[j]$
- À cette position i on doit faire un décalage de $m - j$, d'où $D[m - suff[j]] = m - j$

Calcul de la table D (cas 2)

- u' n'existe pas dans P



- Situation d'échec à la position i de P
- On cherche la plus grande position j $\leqslant |u|$ telle que $P[1..j]$ est suffixe de u

$\Rightarrow$ On cherche donc un bord $P[1..j]$ tel que $j \leqslant |u|$, càd on cherche j tel que $j = suff[j]$ et $j \leqslant |u|$

Remarque : Le cas 2 produit des décalages plus grands que le cas 1

Exemple de la table des décalages

	1	2	3	4	5	6	7	8	9
P	c	a	b	d	a	b	d	a	b
D	9	9	9	3	9	9	6	9	1

- **Remarque :** On a toujours $D[1] = period(P)$

Algorithme de construction de D

Algorithme 3 : Calcule D

Données : Un mot P de longueur m, la table $suff$

```
/* Initialisation                                   */
pour (i de 1 à m) faire D[i] := m ;
/* Cas 2                                            */
i := 1 ;
pour (j de m − 1 à 0) faire
  | si (j = 0 ou suff[j] = j) alors
  |   | tant que (i ⩽ m − j) faire
  |   |   | D[i] := m − j ;
  |   |   | i := i + 1 ;
/* Cas 1                                            */
pour (j de 1 à m − 1) faire  D[m − suff[j]] := m − j ;
```

- D est calculé en temps $O(m)$

Plan

- Introduction
- Règle du mauvais caractère
- Règle du bon suffixe
- **Phase de recherche**
- Extensions et améliorations

Phase de recherche Boyer-Moore

- Utilisation de la règle simple du mauvais caractère (table R)
- Utilisation de la règle forte du bon suffixe (table D)

Algorithme 4 : Algorithme de Boyer-Moore

Données : Deux chaînes T et P de longueurs respectives n et m.

```
pos := 1 ;
tant que pos ⩽ n − m + 1 faire
    i := m ;
    tant que (i > 0 et P[i] = T[pos + i − 1]) faire i := i − 1 ;
    si i = 0 alors
        Écrire("P apparaît à la position ", pos) ;
        pos := pos + D[1] ;
    sinon
        pos := pos + max(D[i], i − R[T[pos + i − 1]]) ;
```

Complexité

- Complexité dans le pire des cas : $O(n \times m)$

→ Nombre minimum de comparaison n/m

→ Nombre maximum de comparaison $n \times m$

- Si le motif n'existe pas dans le texte, en utilisant seulement la règle forte du bon suffixe, la complexité dans le pire des cas est linéaire
- En utilisant seulement la règle du mauvais caractère, la complexité est $O(m \times n)$ mais si on suppose des chaînes générées aléatoirement, la complexité attendue est sous-linéaire
- Sur des textes en langues naturelles, en pratique on constate un temps d'exécution sous-linéaire

Plan

- Introduction
- Règle du mauvais caractère
- Règle du bon suffixe
- Phase de recherche
- **Extensions et améliorations**

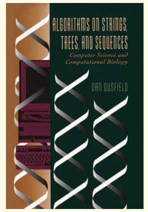
Améliorations

- Des modifications simples de BM peuvent améliorer la complexité :
  1. [Galil, 79] mémorisation des préfixes
     - → complexité en temps $O(n)$,
     - → espace supplémentaire $O(1)$
  2. [Apostolico, Giancarlo, 1986] mémorisation de tous les suffixes
     - → complexité en temps $\leqslant 1,5 \times n$,
     - → espace supplémentaire $O(m)$
  3. [Crochemore et al, 1991] mémorisation des derniers suffixes
     - → complexité en temps $\leqslant 2 \times n$,
     - → espace supplémentaire $O(1)$

⇒ Programme TurboBM

Références

[Lecroq, 11] Ce cours a été construit à partir du cours de Thierry Lecroq : Text searching, 2011 INTERNATIONAL SPRING SCHOOL IN FORMAL LANGUAGES AND APPLICATIONS. Tarragona, Spain April 18-22, 2011. *En anglais*



[Gusfield, 97] Dan Gusfield, Algorithms on Strings, Trees and Sequences - Computer Science and Computational Biology, University of California, Davis. ISBN :9780521585194. Août 1997, 556 pages. *En anglais*