907 Algorithmique du texte. Exemples et applications

I Recherche de motifs

1) Objectif et algorithme naïf

On cherche à trouver les occurrences d'une chaîne de caractères $M[1..m]$ dans un texte $T[1..n]$ $(m \leq n)$.

· L'algorithme naïf associé est la méthode dite des fenêtres glissantes :

T : a b c a b a a b c a b a c

M → → → a b a a

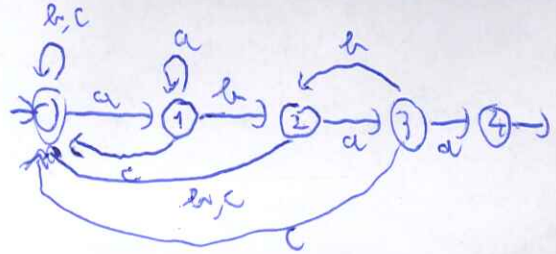
Recherche naïve(T, M)
```
pour s de 0 à n-m
  si P[1..m] = T[s+1...s+m]
    afficher : "M apparaît à la position" s
```

Complexité temporelle : $O((n-m+1)m)$

2) Automates des occurrences, Knuth-Morris Pratt

UVPT

· Idée : On construit un automate reconnaissant le motif M et on rend T en mot d'entrée



La construction de la fonction de transition se fait à l'aide de la fonction suffixe $\sigma(x) = \max\{h, x$ est suffixe de $M[1..h]\}$ en posant :

$$\boxed{\delta(q, a) = \sigma(M[1..q]a)}$$

Complexité temporelle :

· Construction de l'automate $O(m^3|\Sigma|)$

· Recherche : $O(n)$

Amélioration : algorithme de Knuth Morris Pratt

Principe : On évite le calcul de la fonction de transition en introduisant une nouvelle fonction précalculable en temps linéaire :

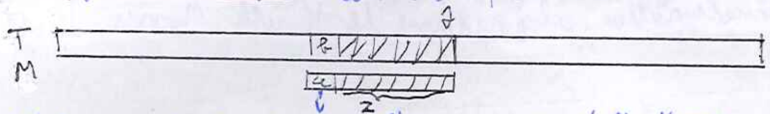
$\pi[q]$ = la longueur du plus long préfixe de M qui est un suffixe propre de $M[1..q]$.

On a alors l'algorithme :

```
KMP(T, M)
q = 0
pour i de 1 à n
  tant que q>0 et M[q+1] ≠ T[i]
    q = π(q)
  si M[q+1] = T[i]
    q = q+1
  si q = m
    afficher "le motif apparaît en position" i-m
    q = π(q)
```

3) Algorithme de Boyer-Moore

Principe : On applique une variante de la méthode des fenêtres glissantes où l'on commence par comparer la fin du motif au texte à chaque tentative.



Notons z le plus long suffixe commun à M et $T[c, \dots p]$.

Si $z \neq M$, on doit effectuer un décalage :

- si bz apparaît dans M, on aligne son occurrence dans T avec son occurrence la plus à droite dans M
- sinon on aligne le plus grand préfixe de M qui est un suffixe de z.

Rq : le calcul du décalage à effectuer est indépendant du texte, et peut être précalculé à partir de M.

Complexité temporelle

$$\boxed{\begin{array}{l}\text{Prétraitement : } O(m|\Sigma|) \\ \text{Recherche : } O(n)\end{array}}$$

4) Table des suffixes

Principe : On construit une table triée par ordre lexicographique contenant les suffixes de T et on recherche M par dichotomie.

Complexité de la recherche : $O(m \log n)$.

Le prétraitement du texte se fait à l'aide d'une méthode de tri par base en $O(n \log n)$] OVPT

5) Bilan et applications

- Boyer-Moore est l'algorithme le plus communément utilisé (ctrl+F, grep)
- La table des suffixes se prête à des recherches multiples sur un même texte.

II Comparaison de chaînes de caractères

1) Plus longue sous séquence commune

- Soit $X[1 \dots m]$ et $Y[1 \dots n]$ deux chaînes de caractères. Une plus longue séquence commune (PLSC) de X et Y est une chaîne $Z[1, \dots k]$ telle qu'il existe φ, σ croissantes tq $\forall i$ $Z[i] = X[\varphi(i)] = Y[\sigma(i)]$.

- Propriété : Sous-structure d'une PLSC
  - si $X[m] = Y[n]$, $Z[k] = X[m] = Y[n]$ et $Z[1, \dots k-1]$ est une PLSC de $X[1 \dots m-1]$ et $Y[1 \dots n-1]$
  - sinon, - si $Z[k] \neq X[m]$, Z est une PLSC de Y et $X[1 \dots m-1]$
    - si $Z[k] \neq Y[n]$ Z est une PLSC de X et $Y[1 \dots n-1]$

De cette propriété, on déduit un algorithme de programmation dynamique permettant de calculer une PLSC, pour une complexité temporelle en $O(m \cdot n)$

2) Distance d'édition

On cherche à passer d'une chaîne X à une chaîne Y en un nombre minimum d'opérations de type :

- remplacer un caractère par un autre
- supprimer un caractère
- insérer un caractère

On obtient un algorithme de programmation dynamique calculant la distance pour tous les préfixes de X et Y en se servant du calcul de la distance pour les préfixes plus petits

Exemple : pour passer de POMME à POIRE

		P	O	M	M	E
	0	1	2	3	4	5
P	1	0	1	2	3	4
O	2	1	0	1	2	3
I	3	2	1	1	2	3
R	4	3	2	2	2	3
E	5	4	3	3	3	2

} initialisation

2 : distance d'édition

initialisation

NB : On remplit le tableau de gauche à droite et de haut en bas

Application : - correction orthographique
- commande "diff"

3) Application à la recherche approchée : recherche avec k différences

On revient au problème de la recherche d'un motif M dans un texte T, mais on se permet de renvoyer des occurrences qui diffèrent de M par au plus k opérations.

Pour cela, on applique une variante de l'algorithme de distance d'édition :

- La ligne d'initialisation de T vaut 0 partout, pour permettre à la recherche de prendre en compte les occurrences de M partant dans T
- A la dernière ligne, on a "lu" M dans T, aussi toutes les cases de cette ligne indiquant une distance inférieure à k indiquent <u>la présence d'occurrences approchées</u>

III Compression : Codage de Huffman

Problème : On considère un ensemble de caractères dotés d'une fréquence d'apparition. On cherche un <u>codage préfixe optimal</u> de cet ensemble, c'est à dire une fonction associant à chaque caractère une suite binaire telle que :

- <u>aucune suite n'est préfixe d'une autre</u>
- <u>le nombre moyen de caractères binaires est minimal parmi les codages préfixes</u>.

On utilise pour cela <u>l'algorithme glouton de Huffman</u> :

- On construit un ensemble de feuilles étiquetées par les caractères et leurs fréquences d'apparition
- tant qu'il y en a au moins 2, on choisit les deux plus petites, que l'on combine pour former un <u>arbre dont la racine est étiquetée par la somme des fréquences</u>.

Ex : $\underbrace{(f,5)\ (e,9)}$ (c,12) (b,13) (d,16) (a,45)

1 étape

(c,12) (b,13) (14) (d,16) (a,45)

(14) : (f,5) (e,9)

Arbre final :

Automate des occurrences

P motif de longueur m, T texte de longueur n

$\uparrow i$ le $i^{\text{ème}}$ préfixe

$\mathcal{A}$ déterministe : $Q=\{0..m\}$, $q_0=0$, $F=\{m\}$, $\delta(q,a)=\sigma(P_q a)$

où $\sigma(x)=\max\{k\in\{0..m\} \mid P_k \sqsupset x\}$

↳ bien défini car $P_0=\varepsilon \sqsupset x$ $\forall x\in\Sigma^*$.

Thm : $\mathcal{A}$ reconnait $\Sigma^* P$ et $\mathcal{A}$ est minimal.

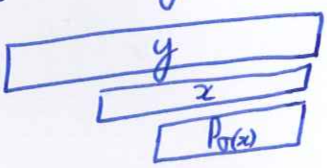
Montrons plus précisément que : $\forall i\in\{0..n\}$, $\delta(q_0,T_i)=\sigma(T_i)$.

Ce qui donnera le résultat car :

il y a une occurrence de P qui fini à la position i dans T

ssi $P \sqsupset T_i$ ssi $\sigma(T_i)=m$.

Lm 1 $x \sqsupset y \Rightarrow \sigma(x)\leq\sigma(y)$

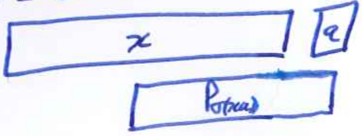


$P_{\sigma(x)} \sqsupset y$ donc $\sigma(x)\leq\sigma(y)$ par maximalité de $\sigma(y)$.

Lm 2 $\sigma(xa)\leq\sigma(x)+1$

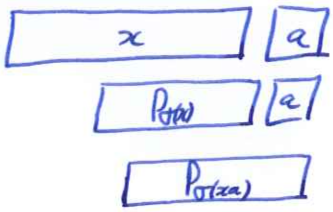
Si $\sigma(xa)=0$ ok.

Sinon



$P_{\sigma(xa)-1} \sqsupset x \Rightarrow \sigma(xa)-1\leq\sigma(x)$ par maximalité de $\sigma(x)$.

Lm 3 $\sigma(xa) = \sigma(P_{\sigma(x)}\, a)$

* 

$\underbrace{P_{\sigma(x)}\, a}_{\alpha} \sqsupset x\,a \Rightarrow \sigma(\alpha) \leq \sigma(xa)$ (lm 1)

* $\underbrace{P_{\sigma(xa)}}_{\beta} \sqsupset xa$ Mais $|\alpha| = \sigma(x) + 1 \geq |\beta| = \sigma(xa)$ (lm 2)

donc $\beta \sqsupset \alpha$

d'où $\sigma(xa) \leq \sigma(\alpha)$ par maximalité de $\sigma(\alpha)$.

Conclusion rec sur i

$\underline{i=0}$ $\delta^*(q_0, \varepsilon) = q_0$ ok

$\underline{i \to i+1}$ $\delta^*(q_0, T_{i+1}) = \delta(\underbrace{\delta^*(q_0, T_i)}_{\sigma(T_i)}, a_{i+1}) = \sigma(P_{\sigma(T_i)}\, a_{i+1})$

$= \sigma(T_i\, a_{i+1})$ (lm 3)

$= \underline{\sigma(T_{i+1})}$.

Minimalité

Pour $k \in \{0..m\}$ on pose $P = P_k Q_k$,

on a $\delta^*(q_0, P_k) = k$.

Soient $i < j \in \{0..m\}$. MQ $i \not\sim j$, càd $L_i \neq L_j$.

* $\underbrace{\delta^*(q_0, P)}_{m} = \delta(\underbrace{\delta^*(q_0, P_j)}_{j}, Q_j)$ d'où $Q_j \in L_j$.

* $\delta^*(q_0, P_i Q_j) = \delta^*(i, Q_j) \neq m$ car $|P_i Q_j| < m$

d'où $\underline{\mathcal{A} \text{ minimal}}$.

Tri des Suffixes

Naïf : $O(\underset{\substack{\uparrow\\ \text{durée}\\ \text{1 comparaison}}}{n} \times \underset{\substack{\uparrow\\ \text{nb comparaisons}}}{n \log n})$

$w \in \Sigma^*$, $|w| = n$, $S = \{ w_i \mid i \in \{1..n\} \}$, $\underline{\Sigma = alph(w)}$.
donc $|\Sigma| = O(n)$

w [| w_i] le i^e suffixe
i

On cherche à trier S par ordre lexicographique : on veut $\pi \in \mathfrak{S}_n$ tq $w_{\pi(i)}$ soit le $i^{\text{ème}}$ élément de S trié.

k premières lettres

On note $w[i..k] = w[i..n]$ si $k \geq n$.

$S_k = \{ w[i..i+k-1] \mid i \in \{1..n\} \}$ $\quad (S_n = S,\ |S_k| \leq |S| = n)$.

$R_k[i]$ = le rang de $w[i..i+k-1]$ dans S_k trié, $\underset{\text{si } i > n}{R_k[i] = 0}$

$\pi_k \in \mathfrak{S}_n$ tq $w[\pi_k(i)..\pi_k(i)+k-1]$ soit le $i^{\text{ème}}$ élément de S_k

↳ à rang égal : stable vis à vis de π_{k-1}

Rq : $R_k[i] \in \{1..|S_k|\}$ alors que $\pi_k(i) \in \{1..n\}$

Idée : si $|S_k| = n = |S|$, $R_k = R_n$ a n élts $\neq$
donc $\pi = R_n^{-1}$

k = 1 $\quad R_1[i]$ = rang de $w[i]$ dans Σ

k → 2k $\quad$ Lm de dédoublement : $R_{2k}[i]$ est le rang de $(R_k[i], R_k[i+k])$ dans la liste couples $(i \in \{1..n\})$ triée.

Preuve $\quad \underbrace{w[i..i+2k-1]}_{w(i)} = \underbrace{w[i..i+k-1]}_{u(i)} \underbrace{w[i+k..i+2k-1]}_{v(i)}$

$w(i) < w(j)$ ssi $u(i)v(i) < u(j)v(j)$
ssi $(u(i), v(i)) < (u(j), v(j))$
ssi $(R_k[i], R_k[i+k]) < (R_k[j], R_j[j+k])$

d'où le résultat.

Rq importante : Poser $R_k[i] = 0$ pour $i > n$ revient à considérer le mot wa^∞ où a est une lettre inférieure à toutes les autres $\longrightarrow$ compatible avec l'ordre lexicographique.

Ex $\quad w = ababca$, $n = 6$, $S = \{w, babca, abca, bca, ca, a\}$

* $S_1 = \{a, b, c\}$

$R_1 = \boxed{1|2|1|2|3|1} \to (12)(21)(12)(23)(31)(10)$

$\pi_1 = \boxed{1|3|6|2|4|5}$

* $S_2 = \{ab, ba, bc, ca, c\}$

$R_2 = \boxed{2|3|2|4|5|1} \longrightarrow (22)(34)(25)(41)(50)(10)$

$\pi_2 = \boxed{6|1|3|2|4|5}$

* $R_4 = \boxed{2|4|3|5|6|1}$

$\pi = \pi_4 = \boxed{6|1|3|2|4|5}$

Algorithme

Utilise le tri par dénombrement :

TRID(π_1, n, R) renvoie $\pi_2 \in \mathcal{O}_n$ tq $R[\pi_2(i)]$ est le $i^{\text{ème}}$ élément de $\{R[i] \mid i \in \{1..n\}\}$ et stable vis à vis de π_1 (si $R[\pi_1(i)] = R[\pi_1(j)]$ avec $\pi_1(i) < \pi_1(j)$ alors $\pi_2(i) < \pi_2(j)$).

↳ s'exécute en $\mathcal{O}(n^2)$ (en supposant $\{R[i]\} \subseteq \{1..n\}$).

TRID(π_1, n, A) =

pour $i = 1$ à n faire $C[i] \leftarrow 0$ // $n = \max_i R[i]$

pour $j = 1$ à n faire $C[A[j]] \leftarrow C[A[j]] + 1$ // $n = |w|$

pour $i = 1$ à n faire $C[i] \leftarrow C[i] + C[i-1]$

pour $j = n$ à 1 faire

| $\pi_2(j) \leftarrow C[A[\pi_1(j)]]$

| $C[A\pi_1 j] \leftarrow C[A\pi_1 j] - 1$

$\boxed{\mathcal{O}(n)}$

TRI-SUFFIXES (w) =

- $k \leftarrow 1$
- $\pi \leftarrow \text{TRID}(id, n, w[\cdot])$ $\quad O(n)$
- $\alpha \leftarrow 1$, $R_1[\pi(1)] \leftarrow 1$
- <u>pour</u> $i = 2$ à n <u>faire</u>
  - <u>si</u> $w[\pi(i)] \neq w[\pi(i-1)]$ <u>alors</u> $\alpha \leftarrow \alpha + 1$ $\quad O(n)$
  - $R_1[\pi(i)] \leftarrow \alpha$
- <u>tant que</u> $\alpha < n$ <u>faire</u> $\quad // \alpha = |S_k|$
  - $\pi \leftarrow \text{TRID}(\pi, n, R_k[\cdot + k])$
  - $\pi \leftarrow \text{TRID}(\pi, n, R_k)$ $\quad O(n)$
  - $\alpha \leftarrow 1$, $R_{2k}[\pi(1)] \leftarrow 1$
  - <u>pour</u> $i = 2$ à n <u>faire</u> $\quad O(n)$
    - <u>si</u> $(R_k[\pi(i)], R_k[\pi(i)+k]) \neq (R_k[\pi(i-1)], R_k[\pi(i-1)+k])$ <u>alors</u> $\alpha \leftarrow \alpha + 1$
    - $R_{2k}[\pi(i)] \leftarrow \alpha$
  - $k \leftarrow 2k$
- <u>renvoyer</u> π $\quad \square$

Il y a au pire $\log_2 n$ itérations de la boucle tant que

$$\Rightarrow \boxed{O(n \log n)}$$