DISTANCE D'EDITION ET FACTEURS

Leçons 907, 921

Référence Crochemore

Navarro, Flexible pattern matching in strings.

Théorème

Soit Σ un alphabet

$s, t \in \Sigma^*$.

On définit la distance d'édition de s à t comme le nombre d'opérations élémentaires (insertion, suppression, substitution) pour transformer l'une en l'autre.

On peut calculer cette distance en $O(|s| \cdot |t|)$ par programmation dynamique.

De plus, l'algorithme peut facilement être adapté pour rechercher les occurences à distance au plus k de s dans t en temps $O(k|t|)$.

Résumé

I - Relation de récurrence et sous-problème optimal.

II - Adaptation pour les facteurs.

III - Calcul de la complexité.

I - Un alignement de s et t est une façon d'observer leur similarité, de la forme, par exemple :

$$\begin{pmatrix} A & C & G & - & - & A \\ A & T & - & C & T & A \end{pmatrix}$$

(T : substitution, - : suppression, T : insertion)

C'est un mot sur $(\Sigma \cup \{\varepsilon\})^2 \setminus \{(\varepsilon, \varepsilon)\}$.

Le coût est le nombre d'opérations.

La distance d'édition $d(s,t)$ est donc

$$d(s,t) = \min_{\text{alignements}} \text{coût}$$

On a alors le sous-problème optimal suivant :

si α est un alignement optimal de longueur n,
$\alpha[1,\dots,n-1]$ est un alignement optimal.

Cela permet de définir la relation de récurrence :

$$\begin{cases} d(\varepsilon,\varepsilon) = 0 \\ d(x,\varepsilon) = |x| \\ d(\varepsilon,y) = |y| \\ d(x,y) = \min \begin{cases} d(x[1,\dots,|x|-1],y) + 1 \\ d(x,y[1,\dots,|y|-1]) + 1 \\ d(x[1,\dots,|x|-1],y[1,\dots,|y|-1]) + \delta_{x[|x|],y[|y|]} \end{cases} \end{cases}$$

On peut le traduire en algorithme :

```
DISTANCE(s,t):
    D = [[0]]
    Pour i = 0 à |s| :   D[i][0] = i
    Pour j = 0 à |t| :   D[0][j] = j
    Pour i = 1 à |s| :
        Pour j = 1 à |t| :
            δ = 0
            Si s[i] ≠ s[j] : δ = 1
            D[i][j] = min(D[i-1][j]+1, D[i][j-1]+1, D[i-1][j-1]+δ)
    Renvoyer D[|s|][|t|].
```

$\Rightarrow$ Complexité $O(|s| \cdot |t|)$.

Exemple

	ε	b	a	a	c	b
ε	(0,0)	(1,0)	2,0	3,0	4,0	5,0
a	1,1	1,1	(1,0)	2,0	3,1	4,1
b	2,2	1,1	2,1	(2,1)	3,2	3,1
c	3,3	2,2	2,2	3,2	(2,1)	3,2
a	4,4	3,3	2,2	2,2	(3,2)	4,2
b	5,5	4,4	3,3	3,3	3,3	(3,2)

→ $\begin{pmatrix} b & a & a & c & - & b \\ - & a & b & c & a & b \end{pmatrix}$

on obtient un alignement en se rappelant des choix qu'on a faits.

II) On peut adapter l'algo pour rechercher des facteurs ~~d'édition~~ à distance d'édition $\leq k$ par exemple.

Pour cela on remarque que le mot vide est facteur à distance d'édition nulle de tout mot.

→ on change donc juste $d(\varepsilon, y) = 0$.

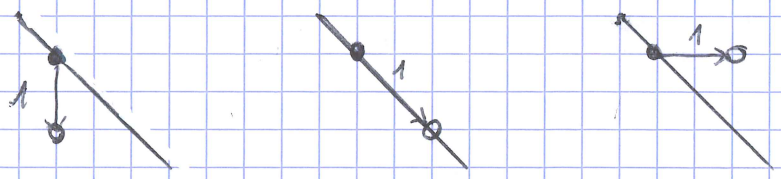
On trouve donc tous les facteurs en $O(|s| \cdot |t|)$

III) En fait on peut faire mieux. Pour cela on définit les cellules actives : une cellule du tableau est active si elle a une valeur $\leq k$.

La position de la cellule active la plus basse ne peut alors être incrémentée que de 1 à chaque lettre du texte, dans le cas où on lit deux fois la même lettre.

III] On a besoin de définir deux types de nœuds : sur chaque diagonale orientée ↘ :

* un d-nœud : la dernière paire (i,j) telle que $D[i][j] = d$ sur une diagonale
* un d-nœud spécial : un nœud que l'on peut atteindre en un coup depuis un $(d-1)$-nœud.



On doit ensuite descendre sur les diagonales selon le plus long chemin de poids 0 pour trouver les d-nœuds.

→ c'est-à-dire, pour un d-nœud spécial (i,j), on cherche le plus long préfixe de $s[i-|s|]$ commun à $t[j-|t|]$.

Comme il y a $|t|$ diagonales, on peut passer des $(d-1)$-nœuds aux d-nœuds en $O(|t|)$. Ensuite, on peut descendre le long d'une diagonale en $O(1)$ car cela revient à chercher le dernier ancêtre commun de deux suffixes dans l'arbre des suffixes communs à s et t (*)

De plus le cas $d=0$ correspond à une recherche sans erreur, qui se fait en temps linéaire.

(*) La construction linéaire de l'arbre se fait avec l'algorithme d'Ukkonen, mais c'est pas évident, et après il faut calculer les ancêtres communs en temps linéaire, ce qui en rajoute une couche.