Recherche des facteurs à distance d'édition au plus k

Réf: Crochemore : Algorithms on strings

Th : Etant donné un texte t et un motif x, on peut trouver toutes les occurences à distance au plus k de x dans t en temps $O(kn)$, où $n=|t|, m=|x|$.

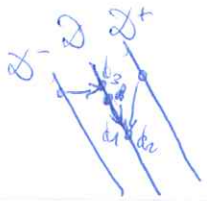
Démo • Première couche de programmation dynamique: on relache le problème en posant $D[i,j] = \min_p (dist(x[1..i], t[p..j]) : p=1,..,j+1)$
Ainsi, chaque j tq $D[m,j] \leq k$ correspond à une occurence de x à distance $\leq k$ à la position $j-m$.
Et le tableau D vérifie la relation de récurrence suivante :

* $D[0,j] = 0 \quad \forall j \leq n$ * $D[i,0] = i \quad \forall i \leq m$
* $D[i,j] = \min \begin{cases} D[i-1,j]+1 & \leftarrow \text{suppression} \\ D[i,j-1]+1 & \leftarrow \text{insertion} \\ D[i-1,j-1]+1_{x[i] \neq t[j]} & \leftarrow \text{éventuelle substitution} \end{cases}$
(pour $i,j>0$)

On obtient un algorithme dynamique qui résout le problème en temps $O(m \times n)$.

• Première idée d'amélioration: dans la relation de récurrence, on voit que si $x[i] = t[j]$, alors $D[i,j] = D[i-1,j-1]$ dans certains cas.
Pour pouvoir améliorer l'algorithme, il faut d'abord le modifier un peu : on voudrait calculer de manière itérative toutes les cases contenant j, pour $j = 0..k$, ainsi on évite de remplir les cases dont la distance est $> k$.

• On définit ainsi une case ℓ-spéciale : une case est ℓ-spéciale si elle contient la valeur ℓ et que aucune case plus basse sur sa diagonale ne contient la valeur ℓ.
Voyons maintenant comment calculer les cases $(\ell+1)$-spéciales étant donné les cases ℓ-spéciales. Etant donné une diagonale et sa case ℓ-spéciale, la case $(\ell+1)$-spéciale provient soit de la diagonale au dessus, soit de celle d'en



dessous, soit de la même diagonale. On calcule donc les trois cases candidates d_1, d_2, d_3 et on garde la plus basse. Puis on descend le long de la diagonale tant que on voit $\ell+1$, ie tant que $x[i] = t[j]$.

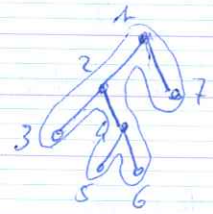
- Comme pour tout ℓ, on a au plus $n+m$ cases ℓ-spéciales (une par diag), si on arrive à faire la descente de diagonale en $O(1)$, notre algo tournera en $O(\underbrace{k}_{\text{nbre d'étapes}} \times \underbrace{(m+n)}_{\text{cases } \ell\text{-spec}}) = O(kn)$

- On cherche donc à résoudre le problème suivant : étant donné i et j on veut calculer le plus long préfixe entre $x[i+1..m]$ et $t[j+1..n]$, qui sont respectivement suffixe de x et de t. On va donc construire l'arbre des suffixes de la chaîne $z = x\#t\$$ et on veut l'ancêtre commun entre les noeuds correspondant à $x[i+1..m]$ et $t[j+1..n]$, ce qui peut se faire en temps constant après un pré calcul linéaire (cf calcul de l'ancêtre commun via tableaux)
(qui contient tous les facteurs)

- Ainsi, étant donné x et t, on calcul l'arbre des suffixes en temps linéaire, puis le précalcul pour l'ancêtre commun aussi linéaire. Enfin on lance l'algo dynamique qui fait des requêtes en $O(1)$, d'où une complexité en $O(k \cdot n)$.

□

Annexe : Ancêtre commun : on étiquette l'arbre selon un parcours eulérien de gauche à droite, ce qui nous donne un tableau représentant l'arbre.



$\rightsquigarrow [1,2,3,2,4,5,4,6,4,2,1,7,1]$