

Tranié Alexandre

TP distance d'édition

IMI

2) D'après la propriété de Bellman: Un problème possède une sous structure optimale si et seulement si une solution optimale du problème contient une solution optimale pour les sous problèmes. Pour la distance de Levenshtein, cela signifie que la distance entre deux chaînes  $s$  et  $t$  peut être déterminée à partir de distances entre leurs préfixes.

### Preuve de la sous-structure optimale

Soit  $s = s_1, \dots, s_m$  et  $t = t_1, \dots, t_n$ . On définit  $d(i, j)$  comme la distance entre les  $i$  premiers caractères de  $s$  et les  $j$  premiers caractères de  $t$ .

On a 
$$d(i, j) = \begin{cases} |i - j| & \text{si } i = 0 \text{ ou } j = 0 \\ \min \begin{cases} d(i-1, j) + 1 & \text{(suppression)} \\ d(i, j-1) + 1 & \text{(insertion)} \\ d(i-1, j-1) + c(s_i, t_j) & \text{(substitution)} \end{cases} & \text{autre cas} \end{cases}$$

où  $c(s_i, t_j) = 0$  si  $s_i = t_j$ , sinon 1

Montrons que les solutions des sous problèmes dans une solution optimale sont elles-mêmes optimales.

Raisonnons par l'absurde  
Pour chaque sous problème  $d(i,j)$  dépend de  $d(i-1,j)$ ,  
 $d(i,j-1)$  et  $d(i-1,j-1)$

Si pour un de ces sous problèmes on a une distance  
plus faible alors le min sera effectué, donc  $d(i,j)$  le sera  
également.  $d(s,t)$  plus globalement. On aurait donc construit  
une solution encore meilleure ce qui est absurde.

#### 4) Algorithme récursif simple

- \* Temps: La fonction récursive calcule la distance pour chaque paire de suffixes possibles des chaînes. On a donc une explosion exponentielle du nombre d'appels récursifs. On a une complexité de  $O(3^{\max(m,n)})$  où  $m$  et  $n$  sont les longueurs des chaînes
- \* Espace:  $O(\min(m,n))$ . La profondeur max de la pile d'appels est la longueur de la chaîne la plus courte

#### Algorithme récursif avec mémorisation

- \* Temps: En mémorisant les résultats intermédiaires chaque sous problème  $(i,j)$  est résolu une fois. Or on a  $\min(m,n)$  sous problèmes résolus en  $O(1)$ . On a donc un temps polynomial
- \* Espace: On a besoin de mémoriser les résultats intermédiaires donc ce sera en  $O(\min(m,n))$

De nos jours on possède une grande capacité de stockage et de mémmoire, c'est plutôt la complexité temporelle à améliorer.  
Il vaut mieux prendre avec mémorisation. En testant sur des longues chaînes de caractères ( $\sim 50$ ) le résultat

est sans appel.

## 7) Algorithme itératif

\* Temps: L'algorithme itératif de programmation dynamique parcourt un tableau de taille  $m \times n$ . Chaque cellule a un temps constant donc en  $O(1)$ . On a donc une complexité totale de  $O(mn)$

\* Espace: On a un tableau 2d de taille  $m \times n$  donc une complexité de  $O(mn)$

Si on compare cet algorithme avec celui récursif sans mémoisation, celui itératif est exponentiellement plus rapide car il calcule chaque sous-problème qu'une seule fois.

On a un temps d'exécution similaire avec celui récursif avec mémoisation. Toutefois l'itératif peut être légèrement plus rapide car il n'implique pas les surcouts liés à la gestion de la pile d'appels et des structures de cache.