

Initiation à la programmation dynamique : la distance d'édition

On définit sur les mots trois opérations élémentaires :

- la *substitution*: on remplace une lettre par une autre,
- l'*insertion*: on ajoute une nouvelle lettre,
- la *suppression*: on supprime une lettre.

Par exemple, sur le mot **carie**, si on substitue **c** en **d**, **a** en **u** et si on insère **t** après le **i**, on obtient **durite**. La *distance d'édition* entre deux mots **U** et **V** est le nombre minimal d'opérations pour passer de **U** à **V**. Ainsi, la distance de **carie** à **durite** est 3: deux substitutions et une insertion. La distance de **aluminium** à **albumine** est 4: une insertion, **b**, une substitution, **i** en **e** et deux suppressions, **u** et **m**.

Une formule de récurrence pour calculer la distance d'édition est

$$\begin{aligned}
 d(\varepsilon, v) &= |v| \text{ (on fait } |v| \text{ insertions)} \\
 d(u, \varepsilon) &= |u| \text{ (on fait } |u| \text{ suppressions)} \\
 d(ua, va) &= d(u, v) \\
 d(ua, vb) &= 1 + \min(d(u, v), d(ua, v), d(u, vb)) \\
 &\quad \text{(la dernière opération est une substitution de } a \text{ en } b, \\
 &\quad \text{ou une insertion de } b, \text{ ou une suppression de } a)
 \end{aligned}$$

où **a** et **b** sont deux lettres quelconques distinctes et où ε est le mot vide.

Question 1. Écrire une fonction `Distance` qui calcule la distance d'édition de deux chaînes de caractères. Testez-la sur **carie** et **durite**, **aluminium** et **albumine**, puis sur un couple de mots un peu plus longs.

Le problème de la fonction `Distance` est qu'elle effectue de nombreux appels récursifs redondants. Cela conduit à une complexité exponentielle. La solution est de stocker les appels récursifs dans une table **D** de dimension 2, telle que $D(i, j)$ soit la distance de $U(U'first..i)$ à $V(V'first..j)$. On ajoute une colonne et une ligne pour traiter le mot vide. Cela donne finalement une table indexée par $U'first-1..U'last$ et $V'first-1..V'last$. Le résultat est $D(U'last, V'last)$. Par exemple, en supposant que les chaînes de caractères sont indexées à partir de 1, la table pour **carie** et **durite** est

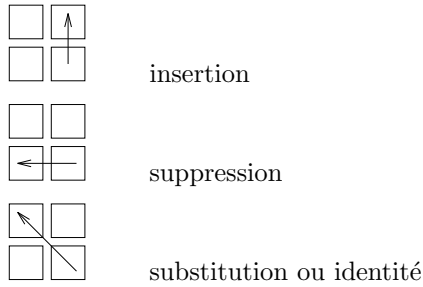
		c a r i e					
		0	1	2	3	4	5
d u r i t e	0	0	1	2	3	4	5
	1	1	1	2	3	4	5
	2	2	2	2	3	4	5
	3	3	3	3	2	3	4
	4	4	4	4	3	2	3
	5	5	5	5	4	3	3
6	6	6	6	5	4	3	

Question 2. Écrire une fonction `Distance_dynamique` qui calcule la distance de deux chaînes de caractères sans appels récursifs, en construisant la table **D**.

On veut maintenant connaître la suite d'opérations qui mène de U à V . Pour cela, on peut visualiser les transformations par un petit schéma:

c	a	r	i	-	e		a	l	-	u	m	i	n	i	u	m
d	u	r	i	t	e		a	l	b	u	m	i	n	e	-	-

Deux lettres identiques sont signalées par $|$. Un espace dans le mot de départ correspond à une insertion, un espace dans le mot d'arrivée correspond à une suppression, et une substitution est représentée par les deux lettres face à face, sans $|$. Il est possible de connaître la dernière opération appliquée en regardant comment la valeur de $D(U'last, V'last)$ a été obtenue. Par exemple, si $D(U'last, V'last)$ est égal à $D(U'last, V'last-1)+1$, alors la dernière opération est une insertion de $V(V'last)$.



Question 3. Retrouvez à la main à partir de la table de l'exemple la suite des transformations pour passer de *carie* à *durite*.

Question 4. Écrire une procédure récursive qui prend arguments deux chaînes de caractères et affiche la suite d'opérations sous la forme décrite ci-dessus. Pour cela, vous devez utiliser la table D et construire l'historique des transformations en remontant dans la table à partir de $(U'last, V'last)$.