Distance d'édition (appelée aussi distance de Levenshtein).

Soit l'alphabet usuel A (c'est-à-dire l'ensemble de caractères).

On note A^* l'ensemble des mots finis sur A (c'est-à-dire l'ensemble des chaînes de caractères).

On dit qu'on passe d'un mot y à un mot z par

- une insertion si $y = uv$ et $z = uav$, avec $u, v \in A^*$ et $a \in A$

- une suppression si $y = uav$ et $z = uv$, avec $u, v \in A^*$ et $a \in A$.

Etant donnés deux mots y et z, la distance d'édition $d(y, z)$ est le plus petit entier k tel qu'on puisse passer de y à z par une succession de k **insertions ou suppressions**, appelées désormais transformations dans la suite de l'énoncé.

On note $|y|$ la longueur de y.

1) a) Montrer que $d : A^* \times A^* \to \mathbb{N}$ est bien définie.

b) Montrer que d vérifie les propriétés suivantes : $d(x, y) = d(y, x)$, $d(x, y) = 0$ ssi $x = y$, et $d(x, z) \leq d(x, y) + d(y, z)$.

2) (★) Montrer que

$$d(ya, zb) = \begin{cases} \min(d(y, zb) + 1, d(ya, z) + 1) \text{ si } a \neq b \\ d(y, z) \text{ si } a = b \end{cases}$$

Indication : On pourra noter que deux transformations qui portent sur des éléments différents commutent entre elles.

3) a) Soient y et z deux mots. Proposer un algorithme qui calcule $d(y, z)$ en temps $O(nm)$, où $n = |y|$ et $m = |z|$.

Indication : On utilisera le principe de la programmation dynamique : en particulier, on notera $p_i = y_0 y_1 ... y_i$ et $q_j = z_0 z_1 ... z_i$ les préfixes de y et z, où $0 \leq i \leq m$ et $0 \leq j \leq n$. Ainsi, $p_0 = \varepsilon$ et $p_m = y$, et pour $0 \leq i < n$, on a $p_{i+1} = p_i y_i$.

b) Implémenter l'algorithme en Python. Il s'agit de définir une fonction `distance(y,z)` qui étant données deux chaînes de caractères y et z renvoie leur distance d'édition.

Distance d'édition (appelée aussi distance de Levenshtein). Corrigé.

1) a) On peut passer de x à ε en $|x|$ suppressions et de ε à y en $|y|$ insertions, donc de x à y en $|x|+|y|$ transformations.
Donc $d(x,y)$ est bien défini et $d(x,y) \leq |x|+|y|$.

b) On passe de x à y en 0 opération ssi $x=y$, donc : $d(x,y)=0$ ssi $x=y$.
Comme une insertion et une suppression sont des transformations inverses l'une de l'autre, alors on peut passer de x à y en k transformations ssi on peut passer de y à x en k transformations, donc $d(x,y)=d(y,x)$.
Comme on peut passer de x à y en $k=d(x,y)$ transformations et de y à z en $l=d(y,z)$ transformations, alors on peut passer de x à z en $(k+l)$ transformations, donc $d(x,z) \leq k+l = d(x,y)+d(y,z)$.

2) On a $d(ya,zb) \leq d(y,zb)+d(zb,z) = d(y,zb)+1$ et de même $d(ya,zb) \leq d(ya,z)+1$.
Donc $d(ya,zb) \leq \min(d(y,zb)+1, d(ya,z)+1)$.
- Supposons $a \neq b$. Il existe une succession de k transformations permettant de passer de ya à zb, où $k=d(ya,zb)$.
Supposons que a soit supprimé dans y au cours des transformations. Comme cette transformation commute avec les suivantes, on peut la déplacer en dernier. Donc les $(k-1)$ premières transformations consistent à transformer y en zb (en faisant abstraction de la présence du a), d'où $k-1 \geq d(y,zb)$. Donc $d(ya,zb) \geq d(y,zb)+1$.
Supposons que a ne soit pas supprimé dans y. Comme $a \neq b$, alors nécessairement le b de zb est ajouté à une certaine étape. Considérons la (dernière) transformation où le b obtenu finalement est ajouté. Cette transformation commute avec les suivantes et peut donc être placée en dernier. Donc les $(k-1)$ premières transformations consistent à transformer ya en z Donc $k-1 \geq d(ya,z)$. Donc $d(ya,zb) \geq d(ya,z)+1$.
On en conclut que si $a \neq b$, alors $d(ya,zb) = \min(d(y,zb)+1, d(ya,z)+1)$.
- Supposons $a=b$. De façon immédiate, $d(ya,za) \leq d(y,z)$ puisqu'il suffit d'appliquer à ya les transformations permettant de passer de y à z. Il reste à prouver que $d(y,z) \leq d(ya,za)$.
Considérons une succession de k transformations permettant de passer de ya à za, où $k=d(ya,za)$.
Si le a de ya est aussi celui de za, alors $d(y,z) \leq k$.
Sinon, le a de ya est supprimé ou bien celui de ya est ajouté.
D'où $d(ya,za) \geq 1+\min(d(ya,z), d(ya,z)) \geq d(y,z)$, car $d(ya,z) \leq d(ya,y)+d(y,z) = 1+d(y,z)$ et $d(y,az) \leq 1+d(y,z)$.
On en conclut que $d(y,z)=d(ya,za)$.

3) a) On procède par récurrence forte sur $|y|+|z|$ en utilisant la formule du 2).

Afin d'éviter de refaire des calculs déjà faits, **on stocke** au fur et à mesure les $d(p_i,q_j)$, où $p_i = y[0:i]$ et $q_j = z[0:j]$ sont les préfixes de y et z de longueurs respectives i et j.
D'où un nombre d'opérations en $O(|y|\,|z|)$.

Il s'agit d'un exemple de programmation dynamique avec mémoïsation.

b) En pratique on stocke $d(p_i,q_j)$ par une matrice M (construite comme liste de listes ou avec un dictonnaire).
Autrement dit, si on note n et m les longueurs respectives de y et z, on définit

$$M([i,j]) = d(p_i,q_j) \quad \text{pour } 0 \leq i \leq n \text{ et } 0 \leq j \leq m$$

On a d'abord $M([i,0])=i$ et $M([0,j])=j$. Et la définition par récurrence forte :

$$M\left([i+1,j+1]\right) = \begin{cases} 1+\min(M([i,j+1]), M([i+1,j])) \text{ si } y[i] \neq z[j] \\ M\left([i,j]\right) \text{ si } y[i+1]=z[j+1] \end{cases}$$

```
def distance(y,z) :
    n = len(y) ; m = len(z) ; M = {}
    for i in range(n+1) :  M([i,0])=i
    for j in range(m+1) :  M([0,j])=j
    for i in range(n) :
        for j in range(m) :
            if y[i]==z[j] :  M([i+1,j+1])=M([i,j])
            else :  M([i+1,j+1]) = 1+min(M([i+1,j]),M([i,j+1]))
    return M([n,m])
```