Exercice : Distance d'édition – Eléments de correction

Les séquences de caractères peuvent encoder de nombreuses informations de nature différente, par exemple du texte, de la voix ou des séquences ADN. L'alignement de deux chaînes des caractères consiste à comparer deux séquences de caractères afin d'évaluer la similarité entre les deux.

La distance d'édition (ou de Levenshtein, ou d'Ulam) est une mesure de la similarité entre deux chaînes de caractères. Cette distance est le nombre minimal de caractères qu'il faut supprimer, insérer ou substituer pour passer d'une chaîne à l'autre.

La distance d'édition entre 2 mots a et b correspond à **la longueur de la plus courte suite de transformations pour passer de a à b**, avec les transformations suivantes :

- insertion d'une nouvelle lettre
- suppression d'une lettre
- remplacement d'une lettre par une autre

Notation : | a | désigne le cardinal de a, c'est-à-dire le nombre de caractères de la chaîne.
a[0] désigne le premier caractère de la chaîne a.

On suppose que :
• supprimer un caractère,
• insérer un caractère,
• substituer un caractère,
sont des opérations qui ont toute un coût unitaire (1).

Si le caractère est identique, la substitution ne coûte rien (0).
La distance d'édition est définie par induction de la manière suivante :

$$d(a,b) = \begin{cases} \max(|a|,|b|) & \text{si } \min(|a|,|b|) = 0 \\ d(a[1:],b[1:]) & \text{si } a[0] = b[0] \\ 1 + \min \begin{cases} d(a[1:],b) \\ d(a,b[1:]) \\ d(a[1:],b[1:]) \end{cases} & \text{sinon} \end{cases}$$

1. La distance d'édition de "chien" à "niche" vaut 4. Expliquer pourquoi

Solution : On écrit chien et niche l'un au-dessous de l'autre. On opère sur "niche" : deux suppressions (n et i), deux substitutions (c et h), une insertion (i), une substitution (e) et une insertion (n). Comme les deux premières substitutions sont des correspondances (ce sont les mêmes lettres), elles ne coûtent rien. Donc la distance d'édition de ces deux mots vaut 4.

2. La distance d'édition représente-t-elle un problème à sous-structure optimale ? Justifier.

Solution : Oui car on arrive à exprimer une solution optimale en fonction des solutions optimales des sous-problèmes.

3. Ecrire une fonction distanceEdition_rec(a,b) qui calcule la distance d'édition de 2 chaînes de manière naïve.

```
def distanceEdition_rec(a, b):  # approche naive
    if len(a) == 0 or len(b) == 0:
        return max(len(a), len(b))
    elif a[0] == b[0]:
        return distanceEdition_rec(a[1:], b[1:])
    else:
        return 1+min(distanceEdition_rec(a[1:], b), distanceEdition_rec(a[1:], b[1:]), distanceEdition_rec(a, b[1:]))
print ( distanceEdition_rec( "niche", "chien"))
print ( distanceEdition_rec( "pirate", "pilote"))
```

4. Écrire une **fonction distanceEdition_PD_rec(a, b, mem)** en programmation dynamique avec memoïsation et comparer les résultats. On pourra tester sur :
- "AGTTC" et "AGCTC",
- "chien" et "niche"
- "mardi" et "mercredi".

```
def distanceEdition_PD_rec(a, b, mem):
    if (a, b) in mem:
        return mem[(a, b)]  #  deja calculé
    else:
        if len(a) == 0 or len(b) == 0:
            mem[(a, b)] = max(len(a), len(b))
        elif a[0] == b[0]:
            mem[(a, b)] = distanceEdition_PD_rec(a[1:], b[1:], mem)
        else:
            mem[(a, b)] = 1 + min(distanceEdition_PD_rec(a[1:], b, mem), distanceEdition_PD_rec(a[1:], b[1:], me
m), distanceEdition_PD_rec(a, b[1:], mem))
        return mem[(a, b)]
```

Si on souhaite programmer dynamiquement par le bas en utilisant un tableau S contenant les distances, il est utile d'exprimer le résultat d'une case en fonction de celles dont elle dépend dans le schéma dynamique :

$$S[i,j] = \begin{cases} \max(i,j) & \text{si } \min(i,j) = 0 \\ S[i-1,j-1] & \text{si } a[i] = b[j] \\ 1 + \min(S[i-1,j], S[i,j-1], S[i-1,j-1]) & \text{sinon} \end{cases}$$

5. On souhaite utiliser la programmation dynamique. Compléter à la main le tableau associé à la distance d'édition de "chien" à "niche".

i \ j	0	1,n	2,i	3,c	4,h	5,e
0						
1,c						
2,h						
3,i						
4,e						
5,n						

Solution : On note :
- x pour suppression, déplacement horizontal,
- s pour substitution, déplacement en diagonal,
- i pour insertion, déplacement vertical.

i \ j	0	1,n	2,i	3,c	4,h	5,e
0	0	1x	2x	3	4	5
1,c	1	1	2	2s	3	4
2,h	2	2	2	3	2s	3
3,i	3	3	2	3	3i	3
4,e	4	4	3	3	4	3s
5,n	5	4	4	4	4	4i

6. Écrire une **fonction distanceEdition_PD_ite(a,b)** qui calcule la distance d'édition de deux chaînes de caractères par programmation dynamique de bas en haut.

```
def distanceEdition_PD_ite(ch1, ch2):
    lg1, lg2 = len(ch1), len(ch2)
    mat_edition    = np.zeros((lg1+1,lg2+1), dtype=int)
    # Constitution de la 1e colonne
    for i in range(0, lg1+1):
        mat_edition[i,0] = i
    # Constitution de la 1e ligne
    for j in range(0, lg2+1):
        mat_edition[0,j] = j
    # remplissage du reste de la matrice
    for i in range( 1, lg1+1 ):
        for j in range( 1, lg2+1 ):
            if ch1[i-1] == ch2[j-1]:
                mat_edition[i,j] = mat_edition[i-1,j-1]
            else:
                mat_edition[i,j] = 1+ min( mat_edition[i-1,j-1] , mat_edition[i-1,j] , mat_edition[i,j-1]  )
        # mat_edition[lg1,lg2] : correspond à la distance d'edition
    return mat_edition[ lg1,lg2 ]

if __name__ == "__main__":
    words = [("sunday", "saturday"), ("chien", "niche"), ("AGATGC", "AGTATCT"),("Levenshtein","Lavenshtein"),
             ("sitting", "kitten"), ('mots', 'mois'), ('janvier', 'février'), ('aminci', 'machine'),
             ('aviron', 'avion'),
             ('courir', 'mourir'), ('mourir', 'mourrir'), ('courir', 'partir'), ('adroit', 'gauche'),
             ('liens', 'links')]
    for a, b in words:
            ite = distanceEdition_PD_ite(a, b)  # [len(a), len(b)]
            mem = distanceEdition_PD_rec(a, b, {})
            print(f"{a} --> {b} :  Bottom/Up --> {ite}, memoisation --> {mem}")
            assert ite == mem, f"Different results --< {a} vs {b}"
```