Programmation dynamique de bas en haut [1]

Exercice 1 (⋆⋆ 𝄞)**.** Dans le prénom **Yanik**, si on modifie trois lettres et qu'on en ajoute une autre, on tombe sur **génies** [2]. Dans cet exercice, nous allons considérer qu'un mot peut subir trois types de modifications :

- Un ajout d'une lettre
- Une suppression d'une lettre
- Une modification d'une lettre

Ainsi, étant donnés deux mots, on peut toujours à partir du premier mot arriver au second par une succession de modifications, par exemple :
`Yanik`→ `ganik` → `génik` → `génie` → `génies`
Ici, on a fait quatre modifications, on peut se convaincre que c'est le minimum possible. Considérons deux mots `mot1` et `mot2`. On appelle distance de Levensthein entre `mot1` et `mot2` le nombre minimum d'opérations qu'il faut faire pour passer de `mot1` à `mot2`. Le but est donc de trouver un algorithme qui calcule cette distance. On pose `n=len(mot1)` et `p=len(mot2)`.

1. Pour `i` dans $[\![0\,;n]\!]$ et `j` dans $[\![0\,;p]\!]$, on note `D[i][j]` la distance de Levensthein des mots `mot1[:i]` et `mot2[:j]`. Que valent `D[0][j]` et `D[i][0]` ? Relier `D[n][p]` à l'objectif de l'exercice.

On prend maintenant `i` dans $[\![1\,;n]\!]$ et `j` dans $[\![1\,;p]\!]$.

2. Si `mot1[i-1]`=`mot2[j-1]`, comparer `D[i][j]` à `D[i-1][j-1]`.
3. Si `mot1[i-1]`≠`mot2[j-1]`, alors pour aller de `mot1[:i]` à `mot2[:j]` il faut bien changer la dernière lettre, il y a donc trois possibilités :
   - On rajoute la dernière lettre de `mot2[:j]`
   - On supprime la dernière lettre de `mot1[:i]`
   - On modifie la dernière lettre de `mot1[:i]` pour qu'elle vaille la dernière lettre de `mot2[:j]`

   Pour chaque cas, exprimer `D[i][j]` en fonction de `D[i-1][j]`, `D[i-1][j-1]`, et `D[i][j-1]` ? Trouver une formule de `D[i][j]` dans tous les cas en se rappelant que l'on cherche la solution la plus courte.
4. Écrire l'arbre de dépendances des calculs pour `D[3][3]`.
5. Programmer une fonction `distance(mot1,mot2)` par méthode de bas en haut. Créer d'abord une liste de listes `D` destinée à contenir tous les `D[i][j]` initialement remplie de 0. Indiquer d'abord les valeurs de `D[i][j]` si `i` ou `j` vaut 0, puis faire une double boucle `for` pour remplir la valeur de `D[i][j]` pour `i`⩾ 1 et `j`⩾ 1 suivant la valeur de récurrence trouvée (attention à bien distinguer les cas).
6. Tester avec `distance("Yanik","génies")`, puis avec les mots `informatique` et `affirmative`.
7. Quelle est la complexité temporelle et spatiale de la fonction `distance` en fonction de `n` et `p` ?
8. Récupérer la liste des mots français du TP1, puis écrire une fonction `ListeMotsPlusProche(chaine)` qui étant donné une chaîne de caractère, notée `chaine`, renvoie renvoie la liste des mots les plus proches (pour la distance de Levensthein) de `chaine`.
9. Votre professeur d'imffaurmathiqe étant très mauvais en orthographe, trouver la liste des mots français les plus proche de ce mot mal orthographié. Si vous êtes arrivés là, félicitations, vous venez de programmer un correcteur orthographique !

Exercice 2 (⋆)**.** Reprendre l'exercice sur le plus grand carré blanc du précédent TP, et réécrivez-le avec une méthode bas en haut.

Exercice 3 (𝄞⋆)**.** Écrire une version récursive avec mémoïsation de la distance de Levensthein.

Exercice 4 (⋆⋆⋆)**.** On revient maintenant dans le cadre de l'exercice 1. et on souhaite écrire une fonction récursive `chemin(mot1,mot2)` qui, étant donnés deux mots, donne un chemin minimum de l'un à l'autre sous forme de liste comme [3] `[Yanik,ganik,génik,génie,génies]`.

1. Traiter les cas où `mot1` est vide (idem pour `mot2`)
2. Si `mot1` et `mot2` finissent par la même lettre, comparer `Chemin(mot1,mot2)` à `Chemin(mot1[:n-1],mot2[:p-1])`.
3. Si la dernière lettre `mot1` est différente de celle de `mot2` alors, la dernière étape sera soit un ajout, soit une suppression soit une modification de la dernière lettre. Grâce à la liste de liste `D`, distinguez les cas, pour savoir s'il faut comparer `Chemin(mot1,mot2)` à `Chemin(mot1[:n-1],mot2)` ou à `Chemin(mot1,mot2[:p-1])` ou bien à `Chemin(mot1[:n-1],mot2[:p-1])`
4. Tester avec `Chemin("informatique","affirmative")`

1. J'espère donc que vous êtes au top.
2. Coïncidence ? Je ne crois pas.
3. Il n'y a pas unicité.