Nom : Ilyass RAMDANI
3A Génie Industriel

[C++] PRALG : Rapport sur le TP Programmation dynamique, À l'attention de monsieur Pascal Monasse

Question 1 :

Le sous problème pour trouver le Nombre minimal d'opérations pour passer de s à s' est de trouver le nombre minimal d'opérations pour passer de s_i à s'_i.

Supposons que pour arriver a si a partir de si, nous avons pris le chemin optimal C. et que pour arriver à s'_{i-1}, nous avons pris le chemin C'

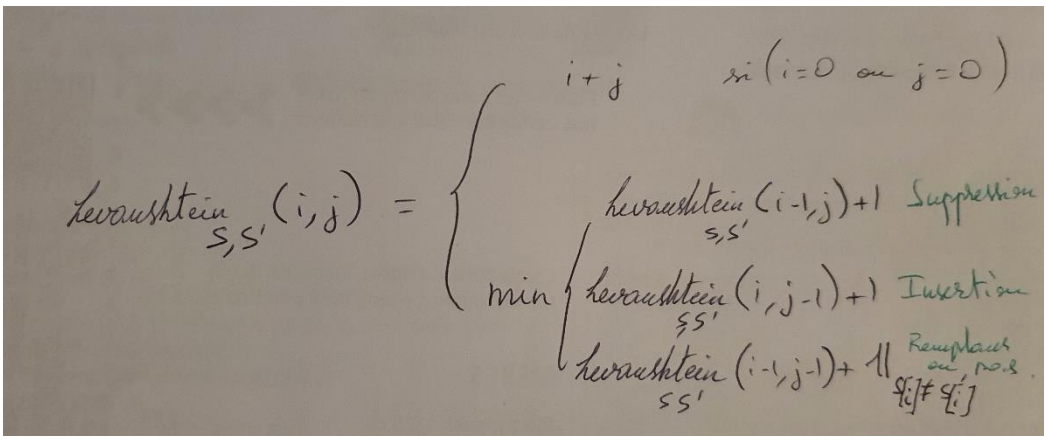
Supposons par l'absurde que la solution optimale du problème de contient pas les solutions optimales des sous problèmes. i.e qu'il existe un sous chemin C'' encore meilleur que C' pour arriver à s'_{i-1}.

Alors dans ce cas il existerait un chemin $C'' + \begin{cases} 0 & if\ c'_i = c_i \\ 1 & else \end{cases}$ qui serait donc meilleur que C qui est le chemin optimal pour arriver à C. ➔ **Absurde !**

Ainsi, la solution optimale C contient les solutions optimales des sous problèmes.

Equation de Bellman

Voici l'équation de Bellman qu'on peut déduire du problème. Le résultat final sera donné par Levenshtein (s.length, s'.length).



Complexité en temps et en espace

Si on pose n (resp. m) la longueur maximale de la chaine de caractère s, (resp. s').

Récursive sans mémoisation

Les complexités spatiale et temporelle sont exponentielles $\Theta(2^{\max(n,m)})$.

Récursive avec mémoisation

Les complexités spatiale et temporelle sont polynomiales : $\Theta(mn)$

Itérative

Les complexités spatiale et temporelle sont polynomiale : $\Theta(mn)$

Temps d'exécution

Voici les 3 résultats qu'on obtient :

Méthode	Temps d'exécution (s)
Récursive	4.60e-04
Récursive avec mémoisation	9.32e-05
Itérative	1.13e-05

Il nous apparait cohérent d'avoir la version itérative qui bat la version récursive car nous passons d'une complexité exponentielle à une complexité polynomiale.

Pas de différence entre mémoisation et sans mémoisation ?

Nous n'obtenons pas une grande difference entre la version avec memoisation et pour la version sans memoisation. Cela est dû au fait que la méthode avec memoisation sera plus rapide dès lors qu'on allonge les chaine de caractere et qu'on recalcule la distance.

Ici la version avec memoisation sera plus rapide vu que la distance pour les 6 premiers caractères sera déjà calculée et en mémoire.

Voici l'exemple avec s = « écoles et fer » et s' = « éclose et mer »

Méthode	Temps d'exécution (s)
Récursive	59.03
Récursive avec mémoisation	0.0004123

La différence entre les deux est claire, la mémorisation est bien plus efficace.

Afficher la suite de modifications élémentaires pour passer de s à s'

Pour afficher la suite des modifications élémentaires, nous allons d'abord retourner la matrice produite dans notre fonction Levenshtein itératif, puis en sortir le chemin optimal.

Décodage

Cela revient à faire le décodage de la matrice suivante :

```
    "" E C L O S E

"" [0, 1, 2, 3, 4, 5, 6]
     \
E  [1, 0, 1, 2, 3, 4, 5]
      \
C  [2, 1, 0, 1, 2, 3, 4]
       ^
O  [3, 2, 1, 1, 1, 2, 3]
        \
L  [4, 3, 2, 1, 2, 2, 3]
          \
E  [5, 4, 3, 2, 2, 3, 2]
           \ <--
S  [6, 5, 4, 3, 3, 2, 3]
```

L'idée générale est que les solutions optimales suivent toutes un **"chemin", du coin supérieur gauche au coin inférieur droit** (ou dans l'autre sens). Sur ce chemin, les valeurs des cellules de la matrice restent les mêmes ou augmentent de 1, en commençant à 0 et en finissant au nombre optimal d'opérations pour les chaînes en question (0 à 3 dans notre cas).

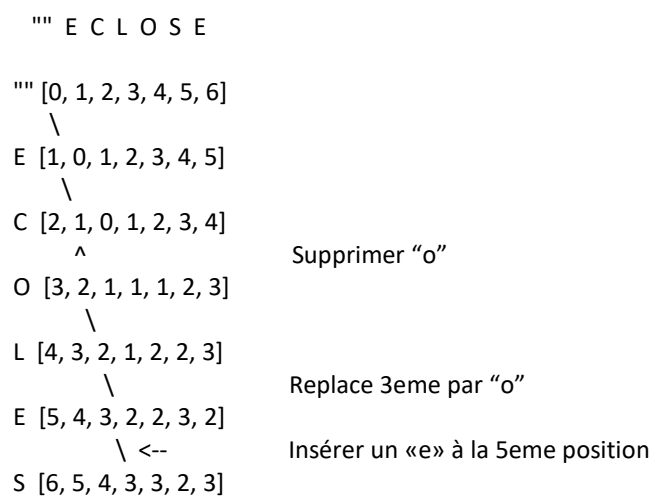
Et pour se terminer dans le coin inferieur droit, elle doit provenir de l'une des 3 cellules

- Immédiatement à sa gauche
- Immédiatement au-dessus
- Immédiatement en diagonale

En sélectionnant une cellule parmi ces trois cellules, une qui satisfait à notre exigence "même valeur ou diminuant d'une unité", nous choisissons effectivement une cellule sur l'un des chemins optimaux. En répétant l'opération jusqu'à ce que nous arrivions dans le coin supérieur gauche (ou même jusqu'à ce que nous atteignions une cellule avec une valeur 0), nous revenons effectivement sur un chemin optimal.

Notez que dans mon script de **remonterModif**, j'ai dû ajouter des conditions supplémentaires si l'élément dans la diagonale est le même que l'élément actuel. Il pouvait y avoir une suppression ou une insertion en fonction des valeurs dans les positions verticale (en haut) et horizontale (à gauche).

Dans notre exemple



Solution itérative linéaire en espace

Dans notre version itérative, nous gardions toute la matrice en mémoire ce qui nous donnait une complexité de $O(n \times m)$.

Ici on a décidé **de garder que les deux dernières lignes** de la matrice ce qui nous mène à une complexité linéaire en espace $O(n)$.