PRALG

Rapport TP6: Distance d'édition

1. :(

2. Soient s_1 et s_2 deux chaînes de caractères. On note leurs tailles respectives N et N' (nombre de caractères). On travaille sur leurs préfixes: soient $i \in [N]$ et $j \in [N']$. La distance de Levenshtein est alors donnée par la relation suivante:

$$d_L(s_1, s_2) = \begin{cases} \max(i,j) & \text{si } i = 0 \text{ ou } j = 0 \\ \min\left(1 + d_L(s_1^{i-1}, s_2^j), 1 + d_L(s_1^i, s_2^{j-1}), \delta_{c_1^i, c_2^j} + d_L(s_1^{i-1}, s_2^{j-1})\right) \end{cases}$$

   où s_k^l représente le préfixe d'ordre l de la chaîne s_k, c_k^l est le l-ème caractère de la chaîne s_k et δ est le symbole de Kronecker.
   Quitte à échanger les chaînes, on peut supposer $N \leq N'$. On suppose qu'on est dans le cas de non sous-optimalité: pour i,j le nombre de modifications pour aller de s_1 à s_2 est optimal, mais pour aller de s_1^{N-1} à s_2^{N-1} (chaînes initiales sans la (les) dernière(s) lettre(s)) il n'est pas optimal. Pour effectuer la dernière étape (de s_1^{N-1} à s_2) il y a plusieurs cas: on note M^* le nombre optimal d'opérations de cette étape et M le nombre d'opérations réellement effectuées dans notre cas. En notant Z^* le nombre d'opérations optimal pour passer de s_1 à s_2, X^* le nombre d'opérations optimal pour passer de s_1^{N-1} à s_2^{N-1} et X le nombre d'opérations effectuées réellement pour passer de s_1^{N-1} à s_2^{N-1}, on a:

$$Z^* \leq X^* + M^*$$

$$Z^* = X + M > X^* + M > X^* + M^*$$

   Il y a contradiction !

3. cf. code

4. **Récursif simple** Il n'y a pas d'allocation mémoire. Cependant, pour ce qui est du nombre d'itération et calculs (complexité en temps et opérations) est de l'ordre de $\mathcal{O}(3^{\max(N,N')})$. Il s'agit d'une complexité exponentielle du au fait qu'on appelle récursivement la fonction trois fois par boucle.
   Récursif mémoïsé Désormais, on enregistre les opérations effectuées: on alloue une mémoire totale d'une taille inférieur au produit des tailles des deux chaînes: $\mathcal{O}(N \times N')$. La complexité en temps est directement liée à ce résultat (on n'effectue pas de récursion si le calcul a déjà était fait et enregistré en mémoire): $\mathcal{O}(N \times N')$. Il s'agit donc de complexités polynomiales.

5. Lorsque l'on fait des tests on constate bien une différence flagrante: l'algorithme mémoisé est de plus en plus rapide et met en évidence la différence de complexité.

6. cf code

7. **Itératif** Dans ce cas, l'algorithme crée une matrice (tableau) de taille $N + 1 \times N' + 1$. C'est dans cette matrice qu'il enregistre toutes les opérations. Cette matrice est construite en bouclant sur s_1 et sur s_2. On déduit que: la complexité en espace est de l'ordre de $\mathcal{O}(N \times N')$ et celle en temps est de l'ordre $\mathcal{O}(N \times N')$. Là encore il s'agit de complexité polynomiale. Remarquons que si l'on cherchait uniquement à obtenir la distance de Levenshtein (et non pas à enregistrer les opérations effectuées), on pourrait garder en mémoire à chaque itérations uniquement la ligne précédent l'actuelle traitée: ce qui donne une complexité en espace de l'ordre de $\mathcal{O}(N')$.

8. Là encore on vérifie la théorie par la simulation: le temps d'exécution de l'algorithme itératif est comparable à celui du récursif mémoïsé, et donc largement inférieur à celui du récursif simple.

9. cf code

10. Notons que l'ajout de la possibilité d'échange de lettres juxtaposée (Damerau-Levenshtein) ne modifie pas fondamentalement les algorithmes et leur complexité (ils restent exponentiels ou polynomiaux comme ils l'étaient).