TD02 : Programmation dynamique

1 Distance d'édition ou « distance de Levenshtein »

1.1 Présentation du problème

On considère deux mots M et P sur un alphabet donné Σ.
On cherche à minimiser le nombre d'opérations pour passer du mot M au mot P, en ne s'autorisant que trois opérations élémentaires suivantes :

- le remplacement d'un caractère
- l'ajout d'un caractère
- la suppression d'un caractère

Par exemple, on peut passer du mot « TARTE » au mot « PARTI » à l'aide de deux remplacements, du mot « DEVOIR » au mot « AVOIR » par un ajout et un remplacement.
La question est de savoir si le nombre d'opérations pour passer d'une chaîne à l'autre est minimal.

- Définition :

On appelle **distance de Levenshtein**, ou encore **distance d'édition**, entre deux mots ce nombre minimal d'opérations pour passer de M à P.
Il s'agit d'une distance au sens mathématique sur l'ensemble des mots sur un alphabet Σ.

On notera $d(M,P)$ le nombre minimal d'opérations élémentaires pour passer d'un mot M à un mot P.

Question 1. En considérant un pire cas, donner un majorant, en fonction de m et p, du nombre minimal d'opérations élémentaires, $d(M,P)$, pour passer d'un mot Mde longueur m à un mot P de longueur p.

Question 2. En reprenant les notations du *slicing*, et en considérant le devenir de la dernière lettre du mot M, justifier que, pour passer d'un mot M à un mot P, $d(M,P)$ nécessairement égal à l'une des valeurs suivantes :

$d(M[:-1],P[:-1])$, $d(M[:-1],P[:-1])+1$, $d(M[:-1],P)+1$ ou $d(M,P[:-1])+1$.

1.2 Relation de récurrence

Notons m et p les longueurs des mots M et P, et $L(i,j)$ la distance de $M[:i]$ à $P[:j]$ pour $(i,j) \in [\![0;m]\!] \times [\![0;p]\!]$.

On donne les relations de récurrence suivantes :

$$(1)\ \forall\, i \in [\![0;m]\!], \qquad L(i,0) = i$$
$$(2)\ \forall\, j \in [\![0;p]\!], \qquad L(0,j) = j$$
$$(3)\ \forall\, (i,j) \in [\![1;m]\!] \times [\![1;p]\!] \quad L(i,j) = \min\big(L(i-1,j)+1, L(i,j-1)+1, L(i-1,j-1)+\delta(i,j)\big)$$

où $\delta(i,j)$ vaut 0 si $M[i-1] = P[j-1]$ et 1 sinon.

(1) traduit que pour passer de façon optimale de $M[:i]$ à une chaîne vide, il en coûte au mieux i suppressions.
(2) traduit que pour passer de façon optimale d'une chaîne vide à $P[:j]$, il en coûte au mieux j ajouts.
(3) traduit que pour passer de façon optimale de $M[:i]$ à $P[:j]$, on peut :

- soit, supprimer la dernière lettre de $M[:i]$, puis passer de façon optimale de $M[:i-1]$ à $P[:j]$;
- soit, passer de façon optimale de $M[:i]$ à $P[:j-1]$, puis ajouter la dernière lettre de $P[:j]$;
- soit, passer de façon optimale de $M[:i-1]$ à $P[:j-1]$, et alors, soit conserver la lettre en position $i-1$ dans M et en position $j-1$ dans P si elles sont égale (on a alors $\delta(i,j)=0$), soit, sinon, remplacer la lettre en position $i-1$ dans M par la lettre en position $j-1$ dans P (on a alors $\delta(i,j)=1$).

Ces relations de récurrences étant données, on peut construire un algorithme de détermination de la distance entre deux mots :

- soit de façon récursive ;
- soit par programmation dynamique.

1.3 Programmation dynamique

Pour deux mots M et P donnés, de longueurs respectives m et p, on note $T_{\substack{0\leq i\leq m\\0\leq j\leq p}}$ la matrice définie par

$$T_{i,j} = L(i,j).$$

Pour tous $i \in [\![0,m]\!]$ et $j \in [\![0,p]\!]$, on peut calculer directement les valeurs « bordantes » $T_{i,0} = i$ et $T_{0,j} = j$.
Puis, pour tout $i \in [\![1,m]\!]$ et $j \in [\![1,p]\!]$, on peut calculer $T_{i,j}$ connaissant les valeurs des coefficients $T_{i-1,j}$, $T_{i,j-1}$ et $T_{i-1,j-1}$, *i.e.* les valeurs situées immédiatement à gauche, au-dessus ou au-dessus à gauche, de $T_{i,j}$.

Question 3. Construire explicitement la matrice T pour passer des mots « ADA » à « DATA ».

Question 4. Définir une fonction `delta(M, P, i, j)` calculant la valeur de $\delta(i,j)$ pour deux mots M et P de longueurs respectives m et p et tout $(i,j) \in [\![0,m-1]\!] \times [\![0,p-1]\!]$.

Question 5. Définir une fonction `init(M, P)` ayant pour arguments deux mots M et P et renvoyant, sous la forme d'une liste de listes, la matrice T, initialisée avec $T_{i,0} = i$ et $T_{0,j} = j$ tous $i \in [\![0,m]\!]$ et $j \in [\![0,p]\!]$ et tous ses autres coefficients nuls.

Question 6. Définir une fonction `mise_a_jour(T, i, j, M, P)`, faisant appel à la fonction delta et mettant à jour le coefficient $T_{i,j}$, en supposant que les coefficients $T_{i-1,j}$, $T_{i,j-1}$ et $T_{i-1,j-1}$ ont déjà été calculés.

Question 7. Définir une fonction `matrice_distances(M, P)`, construisant la matrice T pour deux mots M et P et qui en calcule tous les coefficients, puis la renvoie.
On fera appel aux fonctions `init` et `mise_a_jour`.

Question 8. Définir une fonction `distance(M, P)`, renvoyant la distance de Levenshtein entre les mots M et P.
On fera appel à la fonction `matrice_distances`.

Question 9. Définir une fonction `optimum_distance(M, P)`, qui renvoie, sous la forme d'une liste de mots, une succession de transformations de M à P par des opérations élémentaires de suppression, remplacement ou ajout d'un caractère, réalisant la distance de Levenshtein de M à P.
Indication : on pourra commencer par définir une fonction `chemin_optimum(M, P)` renvoyant, sous la forme d'une liste de couples (i,j) une succession de cellules correspondant à une transformation de M à P réalisant l'optimum.

2 Plus longue sous-séquence commune

Une chaîne de caractères $X = x_1 x_2 \ldots x_{n-1}$ étant donnée, on appelle sous-séquence de X toute chaîne $Z = x_{i_1} x_{i_2} \ldots x_{i_p}$ telle que $0 \leq p < n$ et $1 \leq i_1 < i_2 < \cdots < i_p < n$.
On s'intéresse ici à la recherche d'une plus longue sous-séquence commune entre deux chaînes de caractères X et Y.
Ce problème a des applications en bio-informatique où l'on s'intéresse à trouver des similitudes entre des brins d'ADN qui peuvent être modélisés comme des mots sur l'alphabet des quatre nucléotides A, C, G et T.

Question 10. Écrire une fonction `gen_sequence(n)` qui génère aléatoirement un mot de longueur n sur l'alphabet $\{'A','C','G','T'\}$.

Question 11. Écrire une fonction `sous_sequences(X)`, qui renvoie une liste de toutes les sous-séquences d'une chaîne X. En donner une version récursive et une version itérative.
Évaluer la complexité de cette fonction.

Question 12. Écrire une fonction `PLSC_rec(X, Y)` renvoyant une plus longue sous-séquence commune à deux chaînes X et Y en faisant appel à la fonction `sous_sequences`.
Évaluer la complexité de cette fonction.

TOURNER LA PAGE

2.1 Sous-structure optimale

On note PLSC(X, Y) la plus longue sous-séquence commune à deux chaînes X et Y.
On reprend les notations du *slicing* pour désigner les séquences X et Y privées de leur dernier caractère : X[:-1] et Y[:-1].

Question 13. On note Z une plus longue sous-séquence commune à X et Y.
Pour formaliser la situation, on pourra noter :

$$X = x_1 x_2 \dots x_{n-1} \text{ et } Y = y_1 y_2 \dots y_{m-1}.$$

Une plus longue sous-séquence commune à X et Y est alors une chaîne Z de la forme

$$Z = z_1 z_2 \dots z_p = x_{i_1} x_{i_2} \dots x_{i_{p-1}} = y_{j_1} y_{j_2} \dots y_{j_{p-1}}.$$

Afin de garder trace de l'emplacement des caractères constituant Z, on peut, pour plus de précision, décrire Z par la liste

$$L = \left[(i_1, j_1), (i_2, j_2), \dots, (i_{p-1}, j_{p-1})\right]$$

telle que $L[k] = (i_k, j_k)$ donne les emplacements respectifs, dans X et dans Y, auxquels le caractère z_k été emprunté.

a. Si les derniers caractères de X et de Y sont les mêmes, que peut-on dire de Z ?

b. Si les derniers caractères de X et Y sont différents, que peut-on dire de Z si son dernier caractère est le dernier caractère de X ou de Y ?

c. Déduire de **a.** et **b.** une décomposition de la recherche d'une PLSC en trois sous-problèmes.

Question 14. Justifier que l'on peut ramener le problème de la recherche d'une PLSC au calcul des valeurs $c(i,j)$ pour $(i,j) \in [\![0;n]\!] \times [\![0;m]\!]$ où $c(i,j)$ est la longueur d'une plus longue sous-séquence commune entre X[:i] et Y[:j], avec les formules de récurrence suivantes :

$$c(i,j) = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0 \\ c(i-1, j-1) + 1 & \text{si } i,j > 0 \text{ et } x_i = y_j \\ \max\big(c(i, j-1), c(i-1, j)\big) & \text{si } i,j > 0 \text{ et } x_i \neq y_j \end{cases}$$

Question 15. En déduire un algorithme de résolution par programmation dynamique du problème de la recherche d'une plus longue sous-séquence commune à deux chaînes X et Y, similaire à celle utilisée pour la distance d'édition, utilisant un tableau à deux dimensions.
On pourra écrire une fonction `tableauPLSC`, construisant et renvoyant le tableau des $c(i,j)$, puis une fonction `longueurPLSC`, renvoyant la longueur d'une plus longue sous-séquence.
Évaluer la complexité en temps et en espace de cet algorithme.

Question 16. Proposer une modification de l'implémentation précédente, inspirée de celle proposée pour la distance d'édition, renvoyant une plus longue sous-séquence commune à deux chaînes X et Y.
On pourra garder la décomposition en deux fonctions.

On voudrait maintenant obtenir toutes les plus longues sous-séquences communes entre deux chaînes X et Y.

Question 17. Proposer une fonction récursive permettant de résoudre ce problème à partir de la lecture du tableau des $c(i,j)$.