Programmation dynamique

1 Plus longue sous-suite commune

Soit $a = a_0 \cdots a_{n-1}$ une chaîne de caractères. On dit qu'une chaîne de caractères $c = c_0 \dots c_{p-1}$ est une sous-suite de a lorsqu'il existe $(i_0, \dots, i_{p-1}) \in [\![0, n-1]\!]^p$ tel que

$$\forall k \in [\![0, p-1]\!] \quad c_k = a_{i_k} \qquad \text{et} \qquad \forall k \in [\![0, p-2]\!] \quad i_k < i_{k+1}$$

Par exemple, `"aceg"` est une sous-suite de `"abcdefg"`, mais `"abcgf"` n'en est pas une.
Dans cet exercice, on s'intéresse à la plus longue sous-suite commune (PLSSC) entre deux chaînes de caractères a et b.
Par exemple, `"les oncours e ma"` est une PLSCC de
`"elle a appris son cours de maths"` et `"les concours c'est demain"`.
C'est une question qui apparaît notamment lors de l'étude de la similitude de brins d'ADN, la longueur d'une PLSSC entre deux brins étant alors une façon de mesurer la similitude.

1. Justifier qu'il n'y a pas unicité d'une PLSSC.
2. Écrire une fonction `Est_ss` prenant en paramètre deux chaînes de caractères `sch` et `ch` et renvoyant True si `sch` est une sous-chaîne de `ch` et False sinon.
3. Soient a et b deux chaînes de caractères. En notant $\ell(i, j)$ la longueur d'une PLSSC des chaînes extraites $a_0 \cdots a_{i-1}$ et $b_0 \cdots b_{j-1}$, trouver une relation entre $\ell(i+1, j+1)$, $\ell(i, j)$, $\ell(i+1, j)$ et $\ell(i, j+1)$.
   On pourra distinguer selon que $a_i = b_j$ ou $a_i \neq b_j$.
4. Proposer une fonction `Long_PLSSC_Asc` prenant en paramètre deux chaînes de caractères et renvoyant la longueur d'une PLSSC en utilisant la programmation dynamique de bas en haut.
5. Proposer une fonction `Long_PLSSC_Mem` prenant en paramètre deux chaînes de caractères et renvoyant la longueur d'une PLSSC en utilisant la mémoïsation.
6. En déduire une fonction `PLSSC_Asc` prenant en paramètre deux chaînes de caractères et renvoyant une PLSSC en utilisant la programmation dynamique de bas en haut.
7. Écrire une fonction récursive `PLSSC_Mem` prenant en paramètre deux chaînes de caractères et renvoyant une PLSSC en utilisant la mémoïsation.

2 Rendu de monnaie

On dispose d'un système de monnaie (pièces ou billets, mais pour simplifier dans la suite on parlera uniquement de pièces) de valeurs $v_0 < v_1 < \cdots < v_{n-1}$ et on doit rendre une somme s en utilisant le moins possible de pièces.
En notant $S = \left\{ (p_0, p_1, \dots, p_{n-1}) \in \mathbb{N}^n \mid \sum_{i=0}^{n-1} p_i v_i = s \right\}$, où pour tout $i \in [\![0, n-1]\!]$, p_i représente ici le nombre de pièces de valeurs v_i, il s'agit de minimiser la fonction $f : S \to \mathbb{R},\ (p_0, p_1, \dots, p_{n-1}) \mapsto \sum_{i=0}^{n-1} p_i$.

1. Écrire une fonction `rendu_glouton`, prenant en paramètres un système de monnaie V et une somme s et renvoyant un dictionnaire représentant les pièces à rendre en utilisant la stratégie gloutonne consistant à rendre le nombre de fois nécessaire la plus grande "pièce" et à recommencer avec une somme à rendre diminuée. On proposera une version itérative et une version récursive.
   Par exemple `rendu_glouton`$([1, 2, 5, 10, 20], 56)$ doit renvoyer le dictionnaire $\{20 : 2, 10 : 1, 5 : 1, 1 : 1\}$ car on utilise 2 billets de 20, 1 billet de 10, 1 pièce de 5, 0 pièce de 2 et 1 pièce de 1.

On peut montrer que le système de monnaie $V = [1, 2, 5, 10, 20, 50, 100, 200, 500]$ est canonique, c'est-à-dire que la stratégie gloutonne fournit une solution optimale, mais pas le système de monnaie $W = [1, 2, 4, 5]$
On veut désormais déterminer le rendu de monnaie optimal à l'aide de la programmation dynamique.

2. Soit $V = [v_0, \dots, v_{n-1}]$ un système de monnaie et s la somme que l'on veut former. On suppose que l'on peut toujours le faire (sinon on peut renvoyer un message d'erreur).
   Pour tout $(i, j) \in [\![1, n]\!] \times [\![0, s]\!]$, on note $t_{i,j}$ le nombre minimal de pièces nécessaires pour former la somme j en utilisant les pièces v_0, …, v_{i-1}.
   Trouver une relation sur t et justifier l'intérêt de la programmation dynamique.
3. Écrire une fonction `min_Asc` prenant en paramètres une liste V et un entier s et renvoyant le nombre minimal de pièces nécessaire pour rendre la somme s avec le système de monnaie V, en utilisant la méthode de bas en haut.
4. Écrire une fonction `rendu_Asc` prenant en paramètres une liste V et un entier s et renvoyant un dictionnaire correspondant à un rendu optimal de la somme s avec le système de monnaie V, en utilisant la méthode de bas en haut.
5. Écrire une fonction `min_Mem` prenant en paramètres une liste V et un entier s et renvoyant le nombre minimal de pièces nécessaire pour rendre la somme s avec le système de monnaie V, en utilisant la mémoïsation.
6. Écrire une fonction `rendu_Mem` prenant en paramètres une liste V et un entier s et renvoyant un dictionnaire correspondant à un rendu optimal de la somme s avec le système de monnaie V, en utilisant la mémoïsation.