TP7 : recherche de motifs et programmation dynamique

AP3 Algorithmique et programmation — L2 mathématiques

24 novembre 2016

Exercice 1 (Algorithme de Karp-Rabin). (a) Le schéma de Horner d'un polynôme $P(X)$ en une variable X est une présentation particulière de celui-ci qui permet de l'évaluer rapidement en un point lorsqu'on a la liste de ses coefficients. Soit

$$P(X) = \sum_{i=0}^{n} a_i X^i = a_0 + a_1 X + a_2 X^2 + \cdots + a_n X^n.$$

Pour l'évaluer en un point x_0, on peut le faire de la façon suivante en définissant une suite intermédiaire $(b_i)_{0 \leqslant i \leqslant n}$:

$$\begin{cases} b_n = a_n \\ b_{n-1} = a_{n-1} + b_n x_0 \\ \vdots \\ b_0 = a_0 + b_1 x_0 \end{cases}$$

Dans ce cas, $P(x_0) = b_0$. Informellement, cela donne une présentation du polynôme $P(X)$ sous la forme :

$$P(X) = a_0 + X(a_1 + X(a_2 + ... + X(a_{n-1} + X a_n) ...)).$$

Montrer que cette approche permet de calculer $P(x_0)$ en $\Theta(n)$ opérations.

(b) Ecrire une fonction python récursive `recHorner(w)` prenant en entrée une chaîne de caractères `w='a_0a_1\ldots a_n'` où, $a_i \in \{0, ..., 9\}$, $i \leqslant n$, et qui renvoie :

$$P(10) = a_0 + 10 \cdot (a_1 + 10 \cdot (a_2 + ... + 10 \cdot (a_{n-1} + 10 \cdot a_n) ...)).$$

(c) Quelle type de récursivité est utilisée dans la procédure précédente ? Ecrire une version itérative `itHorner(w)` de ce même algorithme.

(d) En s'aidant de la réponse à la question précédente (pour la représentation d'un mot par un entier - sans passer par la fonction de conversion de python), écrire une fonction python `KarpRabin(C,M)` qui implémente l'algorithme de Karp-Rabin vu en cours.

Exercice 2 (Plus long sous-suite commune)**.** (a) Soit Σ un alphabet fini. Une *sous-suite commune* à deux mots $u, v \in \Sigma^*$ est une suite $w \in \Sigma^*$, tel qu'il existe deux suites croissantes d'indices $i_1 < i_2 < \ldots < i_{|w|}$ et $j_1 < j_2 < \ldots < j_{|w|}$ vérifiant

$$w_k = u_{i_k} = v_{j_k} \text{ pour } 1 \leqslant k \leqslant |w| \,.$$

Le problème de *la plus longue sous-suite commune* (LCS) consiste à déterminer une sous-suite de longueur maximale commune à u et v.

Par la suite, A sera un tableau à deux dimensions indexé par $\{0, \ldots, \text{len(u)}\} \times \{0, \ldots, \text{len(v)}\}$ et $A[i, j]$ désignera la longueur de la plus longue sous-suite commune à $u[1, \ldots, i]$ et $v[1, \ldots, j]$.

On peut prouver (cf cours) que le tableau A vérifie la propriété suivante :

Si $A[0, j] = A[i, 0] = 0$ alors pour $0 \leqslant i < n$ et $0 \leqslant j < m$:

$$A[i+1, j+1] = \max \begin{cases} A[i, j] + \mathbf{1}_{u[i+1]=v[j+1]} \\ A[i+1, j] \\ A[i, j+1] \end{cases}$$

Écrire une fonction python `TabDyn(u,v)` renvoyant un tableau A comme ci-dessus i.e. tel que, pour tout $i \leqslant n$ et $j \leqslant m$, $A[i, j]$ contient la longueur de la plus longue sous-suite commune à $u[1, \ldots, i]$ et $v[1, \ldots, j]$.

(b) Écrire une fonction python `LCS(u,v)` renvoyant une plus longue sous-suite commune à u et v.

(c) Déterminer la complexité de cet algorithme.