Recherche de motifs

****

Exercice 1 : Algorithme naïf

(a) Écrire en python une fonction rechercheMotifNaif(C,M) implémentant l'algorithme de recherche naïf d'un motif M dans une chaîne de caractères C. On souhaite que l'algorithme retourne la liste des positions dans C où le motif M apparaît.

(b) On suppose maintenant que le motif M peut-être "à trous" (il peut y avoir autant de trous qu'on le souhaite dans le motif, mais chacun est de longueur fixée). Chaque lettre appartenant à un trou est représentée par le caractère '*'. Par exemple, la chaîne M='art**per' sera considérée comme un motif contenant les lettres 'a', 'r' et 't' aux trois premières positions, les lettres 'p', 'e' et 'r' au trois dernières avec deux lettres quelconques en quatrième et cinquième positions. Adapter l'algorithme précédent pour qu'il prenne en compte ce genre de motif.

(c) Donner la complexité des deux algorithmes de recherche écrits ci-dessus.

(d) On suppose maintenant que C et M sont des listes bi-dimensionnelles : C est donc une matrice de taille $n \times m$ et M une matrice de taille $n_0 \times m_0$ avec $n_0 \leqslant n$ et $m_0 \leqslant m$. Soit $0 \leq i \leqslant n - n_0 - 1$, $0 \leqslant j \leqslant m - m_0 - 1$, on désigne par C_i^j la sous-matrice de C constituée des lignes $i, i+1, ..., i+n_0-1$ et des colonnes $j, j+1, \ldots, j+m_0-1$. On dit que le motif M apparaît dans le "tableau" C à la position (i,j) si $M = C_i^j$.

Ecrire en python une fonction rechercheMotif2DNaif(C,M) qui prend en entrée deux matrices C et M et recherche les occurrences d'apparition du motif M dans C.

Exercice 2 : 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 :

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

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 :

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

(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 3 : Recherche d'un motif à l'aide d'un automate

On s'intéresse à nouveau à la recherche des occurences d'un motif M dans une chaîne de caractères C. On appelle Σ l'alphabet (vous pouvez par exemple prendre $\Sigma = \{a, b, \ldots, z\}$).

Un mot $S = S_1 \ldots S_q$ est appelé *suffixe* de $T = T_1 \ldots T_p$ si

$$q \leqslant p \text{ et } S_{q-i} = T_{p-i} \text{ pour tout } i \in \{0, \ldots, q-1\}.$$

Pour un mot $T = T_1 \ldots T_p$ et $\ell \in \{0, \ldots, p\}$, on appelle suffixe de longueur ℓ de T le mot $T_{p-\ell+1} \ldots T_p$ (pour $\ell = 0$, ce suffixe est le mot vide).

(a) (Phase de pre-processing.) Ecrire une fonction qui étant donné un motif $M = M_1 \ldots M_m$ et un alphabet Σ, retourne une table T vérifiant la propriété suivante :

> Pour tout $\ell \in \{0, \ldots, m\}$ et $Y \in \Sigma$,
> $T[\ell, Y] = \ell'$ où $\ell' \in \{0, \ldots, \min(\ell+1, m)\}$ est le plus grand entier tel que $M_1 \ldots M_{\ell'}$ est un suffixe du mot $M_1 \ldots M_\ell Y$.

(b) (Phase de recherche.) La table calculée à la question précédente correspond à la fonction de transition d'un automate fini permettant lors du parcours d'un texte C de gauche à droite de connaître à toute étape le plus long préfixe du motif M vu à cet instant (le préfixe de longueur ℓ du mot M étant le mot $M_1 \ldots, M_\ell$).

En déduire un algorithme calculant la liste des occurences d'un motif M dans un texte C. Ecrire la fonction correspondante en Python.

(c) Donner la complexité de l'algorithme implémenté ci-dessus (on pensera bien à analyser les deux parties : construction de l'automate, puis phase de recherche). Dans quel cas est-il intéressant d'utiliser cet algorithme de recherche de motif ?