TP2 - Recherche de motifs

Un *motif* p est un mot sur un alphabet Σ. Étant donné un mot $t \in \Sigma^*$, on souhaite construire un algorithme qui détermine si le motif p apparaît dans le mot t, c'est-à-dire si $t \in \Sigma^* p \Sigma^*$.
On note $\sigma = |\Sigma|$, $n = |t|$ et $m = |p|$. Étant donnée la suite $p_0, p_1, \dots, p_{m-1}$ des lettres de p, pour tout entier $0 \leqslant k \leqslant m$, on note P_k le k^e préfixe de p. On a $P_0 = \varepsilon$ et $P_k = p_0 \dots p_{k-1}$ si $1 \leqslant k \leqslant m$.

Algorithme naïf

Question 1. Proposer un automate fini non déterministe possédant $m+1$ états, $m + 2\sigma$ transitions, un unique état initial et un unique état final, reconnaissant le langage $\Sigma^* p \Sigma^*$.

Question 2. Écrire une fonction `sub_string : string -> int -> int -> string` telle que `sub_string str idx str_lgt` renvoie la sous-chaîne de `str` de longueur `str_lgt` commençant à l'indice `idx`. La fonction devra traiter le cas où `str_lgt` est trop grand.

Question 3. Écrire une fonction `find_pattern : string -> string -> bool` telle que `find_pattern p t` renvoie `true` si le motif `p` apparaît dans le mot `t` et `false` sinon.

Question 4. Estimer les complexités de ces deux fonctions.

Algorithme KMP

Dans les années 1970, Knuth, Morris et Pratt ont proposé un algorithme dit *KMP* qui résout ce problème avec une complexité $O(m+n)$ à l'aide d'un automate fini déterministe complet.
Le symbole $\sqsupseteq$ désigne la relation *est suffixe de*. Soit la fonction :

$$\pi : \begin{cases} [\![1, m]\!] & \to [\![0, m-1]\!] \\ q & \mapsto \max\{k < q, P_k \sqsupseteq P_q\} \end{cases}$$

Pour tout entier naturel k, on note $\pi^k = \pi \circ \dots \circ \pi$ (k fois). Soit n le plus petit entier naturel non nul tel que $\pi^n[q] = 0$.
On pose :

$$\forall q \in [\![1, m]\!] \quad \pi^*[q] = \{q, \pi[q], \pi^2[q], \dots, \pi^n[q] = 0\}$$

Question 5.
□ **5.1.** Justifier que pour tout entier naturel q, $0 \leqslant \pi[q] < q$. En particulier, que vaut $\pi[1]$?
□ **5.2.** Si $\Sigma = \{a, b\}$ et $p = ababbaba$, déterminer π et calculer les ensembles $\pi^*[q]$ pour $1 \leqslant q \leqslant 8$.

Question 6.
□ **6.1.** Si $0 \leqslant k \leqslant q < m$, montrer que $P_{k+1} \sqsupseteq P_{q+1}$ si et seulement si $P_k \sqsupseteq P_q$ et $p_k = p_q$.
□ **6.2.** En déduire que :

$$\forall q \in [\![1, m-1]\!] \quad \pi[q+1] \leqslant \pi[q] + 1$$

□ **6.3.** Montrer alors que :

$$\forall q \in [\![1, m-1]\!] \quad i \in \pi^*[q] \implies P_i \sqsupseteq P_q$$

Question 7. Soit q un entier de l'intervalle $[\![1, m-1]\!]$. On suppose qu'il existe un entier $j < q$ maximal tel que $P_j \sqsupseteq P_q$ et $j \notin \pi^*[q]$.
□ **7.1.** Montrer qu'il existe j' minimal tel que $j' \in \pi^*[q]$ et $j < j'$.
□ **7.2.** En déduire que $P_j \sqsupseteq P_{j'}$.
□ **7.3.** Prouver que $j = \pi[j']$ puis que $j \in \pi^*[q]$.

Question 8. Montrer finalement que pour tout entier q de $[\![1, m-1]\!]$, on a $\pi^*[q] = \{i, P_i \sqsupseteq P_q\}$.

Question 9. Des questions précédentes, on déduit un algorithme de complexité $O(m)$, écrit dans un pseudo-langage, permettant de calculer $\pi[q]$ pour tout q. On ne demande pas de prouver sa correction. La fonction π est représentée par un tableau.

Programme 1

```
pi[1] <- 0
k <- 0
pour q de 2 à m
    tant_que k > 0 et p[k] <> p[q-1] faire k <- pi[k]
    si p[k] = p[q-1] faire k <- k + 1
```

 mpi23@arrtes.net

```
6     pi[q] <- k
```

Écrire une fonction `prefix_array : string -> int array` telle que si `p` est une une chaîne de caractères, `prefix_array p` renvoie le tableau π associé à cette chaîne par l'algorithme décrit ci-dessus.

Question 10. L'algorithme suivant, écrit dans un pseudo-langage, prend en paramètres un motif p de longueur m, un mot t de longueur n et affiche les positions des occurrences de p dans t. On ne demande pas de prouver sa correction.

Programme 2

```
q <- 0
pour i de 0 à n-1
    tant que q > 0 et p[q] <> t[i] faire q <- pi[q]
    si p[q] = t[i] alors q <- q + 1
    si q = m alors
      afficher ("Le motif apparaît en position ", i - m + 1)
      q <- pi[q]
```

Écrire une fonction `kmp : string -> string -> int list` qui renvoie la liste des positions des occurrences d'un motif dans un mot.

Question 11. Soit $Q = \{0, \dots, m\}$. À l'aide des résultats précédents, proposer un automate fini déterministe complet dont l'ensemble des états est Q, l'unique état de départ est 0, l'unique état final est m, l'alphabet est Σ, et reconnaissant le langage $\Sigma^* p \Sigma^*$. Ses transitions seront décrites au moyen de la fonction π. Il convient de s'inspirer de la fonction `kmp`.

Annexe : complexité de l'algorithme de calcul de la fonction π

Notons $C(P)$ le nombre de comparaisons de caractères effectuées par l'algorithme. Nous allons déterminer un encadrement de $C(P)$ par des quantités dépendant de m, où m est la longueur du motif P. On utilise pour dénombrer ces comparaisons une *méthode comptable* : la variable k désigne une certaine somme d'argent. Au départ, $k = 0$ crédit. On a trois types de comparaisons.

- Les comparaisons de la ligne 5 : elles sont gratuites et peuvent même rapporter 1 crédit si le test réussit. Il y a $m - 1$ telles comparaisons.
- Les comparaisons de la ligne 4 qui font échouer le test `P[k] <> P[q-1]`. Ces comparaisons ne coûtent rien, et ne rapportent rien non plus (k reste inchangé). Il y a au plus $m - 1$ telles comparaisons.
- Les comparaisons de la ligne 4 qui font réussir le test `P[k] <> P[q-1]`. Ces comparaisons sont payantes, car on entre alors dans le corps de la boucle while, ce qui va diminuer la valeur de k.
  Plus précisément, comme $\pi[k] < k$, chacune de ces comparaisons coûte au moins 1 crédit. Pour les dénombrer, il suffit de faire les remarques ci-dessous.
  - Tout au long de l'algorithme, k reste positif ou nul. On n'est donc jamais à découvert.
  - Au cours de l'algorithme, le cumul des gains ne dépasse pas $m - 1$ crédits : le seul endroit où l'on gagne de l'argent, c'est sur la ligne 5, et encore, pas toujours. On a ainsi dépensé au maximum $m - 1$ crédits. Donc, le nombre de comparaisons payantes est au plus égal à $m - 1$.

On obtient ainsi $m - 1 \leqslant C(P) \leqslant 3(m - 1)$. Donc, $C(P) = \Theta(m)$.

La méthode décrite ci-dessus s'applique également au calcul de la complexité de la fonction `kmp`. Cette fois, c'est la variable q qui joue le rôle du crédit disponible. On obtient une complexité en $O(m + n)$, où m est la longueur du motif P et n est la longueur du mot T.