Mines Informatique optionnelle MP 2018 — Corrigé

Ce corrigé est proposé par Martin Guy (ENS Lyon) ; il a été relu par Hugo Menet (ENS Lyon) et Benjamin Monmege (enseignant-chercheur à l'université).

L'épreuve est divisée en quatre parties portant sur la recherche d'un motif s dans une chaîne de caractères t.

- Le sujet commence par une introduction générale au problème et définit toutes les notions nécessaires. Après avoir prouvé des propriétés basiques sur les motifs et implémenté des fonctions Caml les manipulant (sous forme de liste), on étudie un premier algorithme naïf de recherche de motif. Cette première partie permet de vérifier que l'on maîtrise les bases de Caml et des études de complexité.
- La deuxième partie, indépendante de la première, présente les automates finis déterministes à repli (AFDR). Les premières questions permettent de s'approprier la notion, qui se différencie subtilement des automates classiques au programme. Il s'ensuit des implémentations en Caml, accompagnées de calculs de complexité. C'est le moment de vérifier sa maîtrise des types en Caml.
- La troisième partie, la plus technique et la plus conséquente, introduit les automates de Knuth-Morris-Pratt, basés sur l'algorithme éponyme. On prouve différentes propriétés utiles pour implémenter un algorithme de recherche de motif plus efficace. Cette partie permet de vérifier les acquis en manipulation de mots et en programmation.
- Enfin, on étend la recherche à un ensemble de motifs dans un dictionnaire S. On utilise des AFDR reconnaissant plusieurs motifs, et on termine par une nouvelle approche de reconnaissance pour un ensemble de motifs. La dernière question, ouverte, est intéressante.

Ce sujet plutôt long propose un bon équilibre entre implémentation en Caml et preuves sur les automates, ce qui en fait un très bon exercice technique.

Dans le corrigé, les codes sont écrits en OCaml plutôt qu'en Caml Light, comme le demandait pourtant l'énoncé, car c'est le langage qui deviendra obligatoire aux concours à partir de la session 2019.

Indications

Partie I

1 Trouver un contre-exemple où s apparaît deux fois dans t de sorte que les deux occurrences se superposent dans t.

2 Sans se soucier des lettres dans s et dans t, quelles sont les positions où l'on peut trouver le motif s dans t ?

4 Parcourir en même temps les deux listes `s` et `t` et comparer lettre par lettre. Pour la complexité, séparer selon que k est plus petit ou plus grand que n.

5 Tester chaque position possible à l'aide de la fonction `prefixe`. Il y a une erreur dans l'énoncé : les arguments en entrée sont de type `int list` et non `string`.

Partie II

7 Utiliser le fait que pour $q \in \mathcal{Q}_{\mathcal{A}}$, on a $\rho(q) < q$ et qu'une suite d'entiers strictement décroissante bornée devient stationnaire.

8 Commencer par redessiner les états et les transitions existantes. Ensuite, compléter l'automate avec les transitions manquantes en utilisant la caractérisation des états de repli $\rho^j(q_{i-1})$ avec j le plus petit entier tel que $\delta(\rho^j(q_{i-1}), u_i)$ est défini.

10 Itérer sur chaque ligne pour copier la matrice de transitions.

11 Pour tout état q et tout symbole α où $\delta(q, \alpha)$ n'est pas défini, utiliser la transition depuis l'état de repli $\rho(q)$.

13 Utiliser une fonction auxiliaire récursive pour parcourir le motif u, en faisant attention à l'ordre de reconstruction de la liste d'indices.

Partie III

14 Placer les états puis les transitions, et enfin calculer la fonction de repli ρ en trouvant parmi les préfixes de s le plus grand suffixe possible étant aussi un préfixe. On pourra faire un tableau où, pour chaque préfixe de s, on affiche le plus grand suffixe possible.

16 Cette question comporte une erreur pour $i = 1$. Considérer $i \geqslant 2$. Remarquer que le plus long suffixe $u_1 \dots u_j$ de $u_1 \dots u_i$ peut être vu comme le plus long suffixe de $u_1 \dots u_{i-1}$ auquel on rajoute le symbole u_i. Exhiber alors une récurrence.

17 Séparer le code en trois morceaux : définir les structures de données et initialiser avec les bonnes valeurs ; calculer toutes les transitions ; puis calculer la fonction de repli à l'aide de la caractérisation de la question 16.

18 Remarquer que pour $j \leqslant i$, $\rho^j(i) \leqslant i - j$ et se servir du fait que $\delta(i-1, u_i) = i$. Pour la seconde partie de la question, faire apparaître une somme télescopique.

19 Calculer séparément la complexité de chaque morceau de code et utiliser le résultat de la question 18.

Partie IV

23 Il suffit d'appeler `recherche_kmp` sur chaque élément du dictionnaire et de rassembler les résultats dans une seule et même liste.

24 Plutôt que de traiter chaque motif indépendamment comme en question 23, créer un automate à l'image de la question 22 pour ensuite y trouver les occurrences.

I. Recherche naïve d'un motif

1 Soient s et t deux chaînes de caractères sur l'alphabet $\Sigma = \{a, b\}$. En prenant $s = abba$ et $t = abbabba$, on trouve des occurrences de s dans t aux positions $y = 4$ et $y' = 7$. La chaîne s est de longueur $k = 4$, ce qui donne $y' - k + 1 = 4$. Ainsi,

> Il est possible d'avoir deux occurrences y et y' de s dans t avec $y < y'$ et $y \geqslant y' - k + 1$.

2 Soient $s = \alpha_1 \dots \alpha_k$ et $t = \beta_1 \dots \beta_n$ deux chaînes de caractères sur un alphabet Σ. Par définition, une occurrence indique la position du dernier caractère du motif s dans la chaîne t. Ce motif s étant de longueur k, la première occurrence de s ne peut se trouver avant la position k. On peut trouver d'autres occurrences aux positions $k+1$, $k+2$ … et ce jusqu'à la position n. Par conséquent, on peut en trouver au maximum $n - k + 1$. Ainsi,

> Le nombre maximal d'occurrences d'un motif de longueur k dans une chaîne de longueur $n \geqslant k$ est $n - k + 1$.

Cette borne est atteinte dans le cas où $\alpha_i = a$ pour tout $i \leqslant k$ et $\beta_j = a$ pour tout $j \leqslant n$ (on trouve alors une occurrence pour chaque position possible). Autrement dit,

> La borne est atteinte si s et t ne sont constitués que du même symbole.

3 On calcule récursivement la taille de la liste en comptant 1 à chaque élément vu.

```
let rec longueur l = match l with
  | [] -> 0
  | t::q -> 1 + longueur q;;
```

Le nombre $\mathrm{C}(n)$ d'appels récursifs lors de l'appel de `longueur` avec une liste de longueur $n \in \mathbb{N}$ vérifie la définition par récurrence :

$$\begin{cases} \mathrm{C}(0) = 0 \\ \forall n \in \mathbb{N}^*, \mathrm{C}(n) = 1 + \mathrm{C}(n-1) \end{cases}$$

On a donc $\mathrm{C}(n) = n$, pour tout $n \in \mathbb{N}$. À chaque appel de `longueur`, on exécute un nombre constant d'opérations élémentaires. Ainsi,

> La fonction `longueur` a une complexité en $\mathrm{O}(n)$.

4 On parcourt récursivement les deux listes représentant les chaînes de caractères s et t simultanément. On arrête le parcours dès que deux lettres de s et t sont différentes. On conclut également négativement dans le cas où le motif s est plus long que la chaîne t, auquel cas s ne peut pas être un préfixe.

```
let rec prefixe s t = match s,t with
  | [], _ -> true
  | si::q, [] -> false
  | si::q, ti::v -> (si = ti) && (prefixe q v);;
```

Séparons l'étude de la complexité de `prefixe s t` en deux cas, en notant k et n les longueurs respectives de s et t. Si $k > n$, on parcourt chaque symbole de t, ce qui fait n opérations. Si $k \leqslant n$, on fait au plus k opérations. Dans tous les cas,

> La fonction `prefixe` a une complexité en $\mathrm{O}(\min(k, n))$.

5 La recherche la plus directe est de tester, pour chaque position possible i dans t, si s est un préfixe de $t_i \dots t_n$. Si tel est le cas, on a trouvé une occurrence de s dans t à la position $i+k-1$. Ainsi, la première occurrence testée est celle en position k.

Pour cela, on implémente une fonction auxiliaire récursive `parcourt`, de type `int list -> int list -> int list`, permettant de parcourir le texte t, tout en maintenant la position i de la première lettre. Initialement, t est le texte complet et i vaut k. Pour chaque position dans t, si s est un préfixe, on garde la position, et dans tous les cas on appelle récursivement la fonction en incrémentant i. La liste des occurrences est ainsi construite récursivement, dans l'ordre croissant.

```
let recherche_naive s t =
  let k = longueur s in
  let rec parcourt t i = match t with
    | [] -> []
    | _::q -> if prefixe s t then
                i::(parcourt q (i+1))
              else
                parcourt q (i+1)
  in parcourt t k;;
```

> L'énoncé indique que `recherche_naive` est de type `string -> string -> int list`, mais il doit s'agir d'une erreur étant donné qu'il est aussi indiqué que le type `string` ne sera jamais utilisé.

6 Pour chaque position possible d'un motif dans t, c'est-à-dire $\{k, \dots, n\}$, on appelle la fonction `prefixe`, ce qui fait $n-k+1$ appels d'une complexité $\mathrm{O}(\mathrm{Min}\,(k,n))$. On effectue de plus un appel à `longueur` de complexité $\mathrm{O}(k)$. Ainsi, en supposant $k \leqslant n$,

La complexité de `recherche_naive` est en $\mathrm{O}((n-k+1)\times k)$.

II. Automates finis déterministes à repli

7 Pour un état $q \in \mathcal{Q}_\mathcal{A}$, la suite des $(\rho^j(q))_{j\in\mathbb{N}}$ est une suite d'entiers naturels strictement décroissante tant que $\rho^j(q) \neq 0$ (car $\rho(p) < p$ pour $p \neq 0$). Elle converge donc vers l'état $q = 0$, où elle devient stationnaire car $\rho(0) = 0$ par définition. Ainsi, il existe un rang j à partir duquel $\rho^i(q) = 0$ pour tout $i \geqslant j$. De plus, par définition des AFDR, on impose que pour tout $\alpha \in \Sigma$, $\delta(0,\alpha)$ soit défini. On a donc que

Pour tout $q \in \mathcal{Q}_\mathcal{A}$, pour tout $\alpha \in \Sigma$, il existe $j \geqslant 0$ tel que $\delta(\rho^j(q),\alpha)$ est défini.

8 L'AFDC à construire doit avoir le même nombre d'états et reconnaître le même langage. On commence donc par redessiner les mêmes états en ajoutant les transitions déjà existantes. Enfin, il reste à compléter l'automate avec les transitions manquantes, en posant pour tout état q, $\delta(q,\alpha) = \delta(\rho^j(q),\alpha)$ avec j le plus petit entier tel que la transition soit définie.

Prenons par exemple l'état 1. Il ne manque que la transition $\delta(1,a)$. On remarque que pour $j=1$, $\rho(1) = 0$ et $\delta(0,a) = 1$. On ajoute donc une transition de l'état 1