Mines Informatique MP 2004 — Corrigé

Ce corrigé est proposé par Vincent Puyhaubert (Professeur en CPGE) ; il a été relu par Samuel Mimram (ENS Lyon).

Ce problème est composé de deux problèmes complètement indépendants. Le premier problème propose l'écriture et l'étude d'un algorithme de tri, tandis que le second porte sur une notion de longueur discriminante de deux automates : c'est-à-dire la longueur minimale d'un mot qui soit reconnu par l'un des automates et pas par l'autre.

- L'algorithme de tri proposé dans le premier problème est celui du tri par baquets (« bucket sort »). Le principe de tri est le suivant : pour trier une liste de nombres, on commence par partitionner cette liste en 10 piles, suivant la valeur du premier chiffre de chaque nombre. Puis on concatène les piles (sans changer l'ordre de chaque pile), et on recommence en séparant cette fois selon la valeur du second chiffre et ainsi de suite. Ce tri a l'avantage de fonctionner en temps linéaire en la longueur de la liste, si l'on suppose que le nombre de chiffres de chaque nombre est inférieur à une certaine valeur fixée $maxc$.

  Ce problème comporte beaucoup de programmes à rédiger. La difficulté est bien dosée, puisque les questions et les codes à écrire pour les deux premières parties sont très simples. Les difficultés techniques commencent lors de la troisième partie avec notamment une preuve de terminaison de programme et des questions de complexité délicates à rédiger, ainsi que certains codes plus compliqués à composer. Les dernières questions sont vraiment difficiles.

- Le second problème porte sur les automates et n'exige la rédaction d'aucun code. Étant donnés deux automates $\mathcal{A}$ et $\mathcal{A}'$ à n et n' états qui ne reconnaissent pas le même langage, on cherche une majoration de la longueur d'un plus petit mot reconnu par seulement l'un des deux automates. Cette longueur ℓ est exprimée en fonction de n et n' et est appelée longueur discriminante des automates.

  L'étude débute par des questions de cours concernant quelques constructions très classiques sur les automates (construction d'un automate reconnaissant le complémentaire d'un langage, l'intersection de deux langages). On en déduit une première borne grossière de ℓ.

  Dans la seconde partie, on utilise une représentation, assez inhabituelle en classe préparatoire, d'automates par des applications linéaires. Il en résulte une borne plus fine de cette longueur discriminante. Cette partie est plutôt formelle, mais raisonnablement difficile si l'on garde les idées claires : aucune question ne nécessite de longue démonstration ou une dextérité particulière.

Il s'agit d'un sujet particulièrement intéressant et bien rédigé. Il touche à deux parties importantes du programme d'informatique en classe préparatoire et permet même de découvrir une approche originale des automates grâce à l'algèbre linéaire. Il s'agit donc d'un entraînement idéal aux épreuves d'informatique des concours !

Indications

Problème d'algorithmique et de programmation

1 Commencer par écrire un programme qui compte, pour tout entier k compris entre 0 et `MAX_VAL` le nombre d'occurrences de la valeur k dans la table.

3 Remarquer que vider une table revient simplement à mettre la première valeur de la table à 0.

4 Ajouter un élément revient à exécuter trois opérations : récupérer la longueur n de la table, mettre sa valeur dans la case d'indice n, puis incrémenter n.

5 Le plus simple consiste à faire une boucle pendant laquelle on ajoute l'un après l'autre les éléments de la table T2 à la table T1.

6 Exprimer le nombre d'opérations élémentaires effectuées en fonction de la longueur de la boucle de la question 5.

8 Prouver que le nombre de chiffres de p est le plus grand entier k tel que 10^{k-1} soit inférieur à p.

9 Combiner les deux programmes précédents.

10 Il s'agit de la somme des complexité des programmes précédents.

11 Remarquer qu'à la fin de la première étape, les données sont triées par ordre croissant de leur chiffre de rang 1.

12 Utiliser l'invariant de boucle suivant :

$\mathscr{P}$(i) : À l'issue de la i^{e} boucle, la table tronquée au rang i est triée.

où la table tronquée au rang i est la table obtenue en ne conservant que les chiffres de rang 0 à i de chaque donnée.

13 Traduire quasiment mot pour mot l'étape b) proposée par l'énoncé en début de troisième partie.

14 Même remarque en reprenant cette fois l'algorithme tout entier.

15 Compter le nombre d'opérations élémentaires effectuées lors de chaque boucle, en utilisant les résultats des questions de complexité précédentes.

16 Remarquer que si les n données sont distinctes, leur maximum est de l'ordre de n et en déduire l'ordre du nombre de chiffres de ce maximum.

17 Modifier l'étape d) de l'algorithme, en ne remettant dans la table T, à la fin de l'étape r, que les données supérieures à 10^r. Les autres données sont recopiées dans une autre table `table_finale` au fur et à mesure.

Problème sur les automates

18 Un des sens est évident. Pour l'autre, considérer le plus petit mot de $\mathcal{L}(\mathcal{A})$ et raisonner par l'absurde.

19 Considérer un automate avec les mêmes états, transitions et états initiaux, mais avec pour états finaux $\mathrm{Q} \setminus \mathrm{F}$.

20 Considérer un automate dont les états sont $\mathrm{Q} \times \mathrm{Q}'$, les états initiaux $\mathrm{I} \times \mathrm{I}'$, les états finaux $\mathrm{F} \times \mathrm{F}'$ et les transitions bien choisies.

21 Remarquer que les automates ne sont pas équivalents si et seulement si l'un des deux langages $\mathcal{L}(\mathcal{A}) \cap \overline{\mathcal{L}(\mathcal{A}')}$ ou $\mathcal{L}(\mathcal{A}') \cap \overline{\mathcal{L}(\mathcal{A})}$ est non vide.

22 Déterminiser les deux automates.

23 Chercher des exemples d'automates tous deux à un seul état.

24 C'est extrêmement formel : pour un des sens, partir d'un calcul de i à j d'étiquette m. Écrire $m = a_1 \dots a_k$ et utiliser la définition de φ_m comme la composée des applications $\varphi_{a_1}, \dots, \varphi_{a_k}$ et la définition de ces applications.
Pour la réciproque, introduire les vecteurs $\varphi_{a_1}(e_i), \varphi_{a_2}(\varphi_{a_1}(e_i)), \dots$ et remarquer qu'ils sont tous non nuls, et donc des éléments de la base canonique.

25 Le produit scalaire d'un vecteur de la base canonique avec z vaut 0 ou 1. Déterminer dans quel cas il peut valoir 1, étant donnée la définition de z.

26 Remarquer que l'on a, par définition du produit scalaire canonique,

$$\Phi(\mathrm{E}) \cdot \mathrm{Z} = \varphi_m(e_1) \cdot z - \varphi'_m(e'_1) \cdot z'$$

27 Utiliser le fait que si A et B sont deux ensembles de vecteurs et que A est inclus dans B, alors Vect A est inclus dans Vect B.

28 C'est une conséquence de la même propriété pour les applications φ_m, φ_a et φ_μ (respectivement φ'_m, φ'_a et φ'_μ).

29 Si w est un élément de V_k, introduire un mot m de longueur au plus k tel que $w = \Phi_m(\mathrm{E})$, puis utiliser le résultat de la question précédente.

30 Prendre w appartenant à V_{k+2}, introduire un mot de longueur inférieure ou égale à $k+2$ tel que $w = \Phi_m(\mathrm{E})$. Poser alors $m = \mu a$ et utiliser l'hypothèse de la question pour l'élément $\Phi_\mu(\mathrm{E})$.

31 Remarquer que V_0 est de dimension 1 et que les espaces $(\mathrm{V}_k)_{k\in\mathbb{N}}$ sont de dimension au plus $n + n'$.

32 Il existe par hypothèse un mot m de longueur k tel que $\Phi_m(\mathrm{E}) \cdot \mathrm{Z}$ soit non nul. Montrer alors qu'on peut trouver un mot m' de longueur h qui satisfait cette même propriété, en utilisant le résultat de la question précédente.

33 Raisonner comme pour la question 22.

I. Problème d'algorithmique et de programmation

Première partie

Tri d'entiers compris entre 0 et MAX_VAL

1 La méthode la plus rapide consiste à utiliser un tableau `occurrences` de longueur `MAX_VAL`, dont toutes les cases sont initialisées à 0. On ne parcourt alors qu'une fois le tableau : pour chaque valeur T[i] rencontrée, on incrémente de 1 la valeur de la T[i]$^{\text{e}}$ case de `occurrence` .

> Si l'on reprend mot à mot le principe de l'énoncé, on est tenté d'effectuer deux boucles : pour tout entier k compris entre 0 et `MAX_VAL`, on parcourt les n cases du tableau en comptant le nombre d'occurrence de la valeur k. Ce faisant, la complexité de l'algorithme est donnée par le produit n `MAX_VAL` ce qui ne correspond à celle demandée !

Cette première étape utilise $2n$ opérations élémentaires (on effectue n fois un accès au tableau suivi d'une écriture dans occurrences). Une fois celle-ci terminée, il n'y a plus qu'à réécrire le tableau T

2 Les programmes ci-dessous reprennent les principes de la question précédente. Le premier programme crée le tableau `occurrence`. Le second utilise ce dernier pour reconstituer une version triée du tableau T.

```
let occurrence T =
  let n = T.(0) in
    let a = make_vect MAX_VAL 0 in
      for i=1 to n do
        a.(T.(i)) <- a.(T.(i)) + 1;
      done;
a;;
```

> On remarque que le code de ce premier programme utilise la variable `MAX_VAL` qu'il faut penser à définir avant de rédiger le programme. Il faudra donc, par exemple, commencer par écrire
>
> ```
> let MAX_VAL = 10 ;;
> ```

```
let tri_simple T =
  let s = ref 1 in
    let p = occurrence T in
      for i=0 to (MAX_VAL - 1) do
        for j=1 to p.(i) do
          T.(!s) <- i;
          incr s
        done;
      done;;
```