E3A Informatique MP 2008 — Corrigé

Ce corrigé est proposé par Benjamin Monmege (ENS Cachan) ; il a été relu par Vincent Puyhaubert (Professeur en CPGE) et Guillaume Batog (ENS Cachan).

Le sujet comporte sept exercices indépendants. Ils se répartissent entre trois grandes thématiques :

- **Programmation** :
  - L'exercice 1 propose de trier une liste par dénombrement et de calculer la complexité d'un tel algorithme, dans le pire ou le meilleur des cas ainsi qu'en moyenne. Une bonne maîtrise du calcul de complexité est donc nécessaire.
  - L'exercice 2 est composé de deux questions indépendantes. La première construit un programme pour remplir une matrice à partir des coefficients d'une liste donnée ; la seconde calcule le plus grand nombre de valeurs identiques consécutives dans un tableau donné.
  - L'exercice 3 propose de trouver ce que calculent trois programmes écrits en pseudo-code. Il s'agit d'un exercice difficile dans lequel il faut avoir une bonne intuition, comprendre le but de chacune des variables utilisées et savoir rédiger une réponse satisfaisant le correcteur.
  - L'exercice 4 demande d'écrire un programme restituant une liste d'entiers vérifiant une condition particulière visible sur leur écriture en base 10.
  - L'exercice 5, nettement plus difficile d'un point de vue algorithmique, propose de représenter une union finie d'intervalles fermés disjoints à l'aide de la liste ordonnée de leurs extrémités. Il faut alors réaliser une fonction qui décide si une liste ordonnée correspond à une telle réunion, puis écrire deux fonctions qui réalisent respectivement les opérations d'intersection et d'union.
- **Automates** : l'exercice 6 propose de calculer le cardinal de l'ensemble des mots de longueur n d'un langage rationnel. Il nécessite une bonne aptitude à manipuler les langages et les automates finis, et utilise le formalisme mathématique de l'algèbre linéaire (en particulier le théorème de Cayley-Hamilton).
- **Logique** : l'exercice 7 étudie une classe particulière de fonctions booléennes pour lesquelles il est possible d'appliquer un théorème des fonctions implicites, en détaillant deux exemples. Pour le résoudre, il faut avoir bien compris la différence entre une fonction booléenne et son évaluation en une valeur de vérité.

Il s'agit d'un sujet typique du concours E3A : une longue suite d'exercices très éloignés les uns des autres, de difficultés différentes. On a tout intérêt à le lire en entier au début de l'épreuve afin de sélectionner l'ordre dans lequel traiter les exercices. Il n'est pas nécessaire de faire l'ensemble du sujet pour avoir une note correcte : il vaut mieux explorer en détail quelques exercices plutôt qu'aborder superficiellement chacun d'entre eux.

Indications

1.a Faire attention à la convention Caml qui numérote les tableaux à partir de 0. Un seul passage sur la liste suffit à résoudre le problème.

1.b Pour la complexité en moyenne, supposer que les entiers N et n sont fixés. Combien y a-t-il alors de listes L possibles ? Quel est le coût d'une exécution de l'algorithme sur une liste fixée, en fonction de ses coefficients ? En déduire une majoration la plus simple possible de la complexité en moyenne.

2.a Initialiser la matrice M avec la matrice nulle puis remplacer ses coefficients par ceux de la liste L tant que c'est possible.

2.b Parcourir le tableau T en mémorisant la longueur du plus grand plateau rencontré et la longueur du plateau courant (remarquer qu'un tableau se décompose en plateaux, éventuellement réduits à un seul élément).

3.a Commencer, au brouillon, par décrire l'exécution du programme sur l'entrée $[1, 2, 2, 3, 1]$, puis essayer de trouver le rôle de chacune des variables.

3.b Écrire proprement les appels récursifs sur l'exemple $(\mathrm{N} = 7, p = 3)$ et relier ce calcul au nombre de manières de décomposer N en une somme de p entiers strictement positifs.

3.c Les deux premiers tests inclus dans la boucle `pour` sont à comprendre comme des cas disjoints. En particulier, ne pas rentrer dans le second test lorsque le premier a été positif. Corriger cette erreur puis essayer de faire tourner l'algorithme sur le tableau $[5, -1, 1, 2, -3, -2, 1, 3, 2]$.

4 Décomposer le problème en commençant par écrire une fonction qui teste si un entier à trois ou quatre chiffres est un nombre d'Armstrong.

5.a Écrire une fonction auxiliaire récursive prenant en entrée une liste L et un entier i et vérifiant que L est une liste convenable, dont le premier élément est strictement supérieur à i.

5.b, 5.c Il est possible d'écrire ces fonctions de manières récursives, à condition de bien penser aux cas de base.

6.2.b.i Préférer écrire un automate déterministe reconnaissant le langage $\mathcal{L}(n)$. Commencer par les petites valeurs de n.

6.2.b.iii Conjecturer au brouillon la valeur de la suite récurrente linéaire, puis prouver le résultat par récurrence.

6.3.b Raisonner par double inclusion, en utilisant la définition du langage $\mathcal{L}_j(n)$.

6.3.c Montrer que, pour tout $n \geqslant 2$, $\mathrm{V}(n) = \mathrm{M}\,\mathrm{V}(n-1)$ en exprimant les différentes coordonnées des deux vecteurs, puis conclure par récurrence.

6.4.b Utiliser l'idée développée à la question 6.3.d pour exprimer le terme $u_n^{\mathcal{L}}$ dans ce cas particulier.

7.2 Pour le sens direct, ne pas oublier qu'une fonction booléenne ne peut prendre que deux valeurs (0 ou 1).

7.3 Pour exhiber la fonction g, distinguer le cas du $(n-1)$-uplet $(x_1, \ldots, x_{n-1})$ dont toutes les composantes valent 1, des autres.

Rappelons ici quelques conseils que le jury livre chaque année dans ses rapports. L'épreuve étant essentiellement tournée vers la programmation, il est primordial que les candidats se préparent à cette épreuve : il est en effet plus délicat de proposer un programme correct sans pouvoir le tester sur machine. Même si le jury est clément sur les petites erreurs de syntaxe, le candidat doit s'efforcer de produire un code clair, indenté et largement commenté. Il peut aussi être intéressant de décrire rapidement le rôle d'une variable introduite dans un programme en expliquant son initialisation, ou de découper un programme long en plusieurs sous-programmes dont on décrit la finalité. Les exercices plus théoriques doivent être résolus comme s'ils faisaient partie d'une épreuve de mathématiques : les réponses doivent être concises, mais argumentées précisément.

Exercice 1

1.a Afin d'écrire un programme correct en Caml, dans lequel un tableau de taille N est numéroté de 0 à $N-1$, on choisit de modifier légèrement le résultat demandé par l'énoncé. Ainsi la fonction `tri_denombrement` prend en entrée un entier N et une liste L d'entiers compris entre 1 et N et renvoie un tableau M de taille $N+1$ dont la case d'indice 0 contient 0 et dont la case d'indice $k \in [\![1 ; N]\!]$ contient le nombre d'éléments égaux à k dans la liste L.

Le programme écrit ci-dessous utilise une fonction auxiliaire `passage`, récursive, qui prend en entrée une liste et remplit convenablement le tableau M, élément après élément : à chaque fois que la valeur k est rencontrée dans la liste L, on incrémente la case d'indice k du tableau M.

```
let tri_denombrement N L =
  let M = make_vect (N+1) 0 in
  let rec passage liste =
    match liste with
      | [] -> M
      | k::reste -> M.(k)<-M.(k)+1; passage reste
  in passage L;;
```

Cette fonction n'est pas un algorithme de tri : pour faire cela, il faut reconstituer la liste triée des éléments de L à partir du tableau M. Présentons un exemple de fonction implémentant cette amélioration. Le programme récursif auxiliaire permet de limiter le nombre d'accès à une case du tableau M, puisque cette opération est supposée coûteuse dans cet exercice.

```
let tri N L =
  let M = tri_denombrement N L in
  let rec range i k =
    (* renvoie la liste triée comprenant *)
    (* tous les éléments de L inférieurs à i *)
    if (i=N)&&(k=0) then []
    else if k=0 then range (i+1) M.(i+1)
    else i::(range i (k-1))
  in range 0 M.(0);;
```

1.b La seule opération élémentaire considérée ici est l'accès à une case de tableau : on suppose ici que cette opération a un coût linéaire en l'indice de la case voulue.

> Cette hypothèse, imposée par l'énoncé, est assez étrange puisque, contrairement à une liste, on a coutume de considérer que l'accès à un élément donné d'un tableau se fait en temps constant en Caml.

Appelons n la longueur de la suite L. Remarquons que la seule opération coûteuse est l'accès à la case d'indice x du tableau M : cela coûte $\mathrm{O}(x)$.

- **Complexité dans le meilleur des cas** : tous les éléments de la liste L sont égaux à 1 et donc chaque accès à une case a une complexité en $\mathrm{O}(1)$.

  La complexité dans le meilleur des cas est en $\mathrm{O}(n)$.

- **Complexité dans le pire des cas** : tous les éléments de la liste L sont égaux à N et donc chaque accès à une case a une complexité en $\mathrm{O}(\mathrm{N})$.

  La complexité dans le pire des cas est majorée par $\mathrm{C}\,n\,\mathrm{N}$, où C est une constante indépendante de n et N.

  > Mieux vaut éviter la notation $\mathrm{O}(n\mathrm{N})$ puisque les notations de Landau ne sont traditionnellement définies que pour une suite dépendant d'un unique paramètre.

- **Complexité en moyenne** : précisons ici l'énoncé. On suppose que N et n sont des entiers fixés et on fait varier la liste L donnée en entrée de l'algorithme. L'ensemble des listes L possibles est un ensemble fini. Chaque élément de la liste peut prendre une valeur entre 1 et N. Comme elle est de taille n, l'ensemble des listes est de cardinal N^n. On considère donc que chaque liste a une chance sur N^n d'apparaître.
  Afin de calculer la complexité en moyenne, il faut donc trouver le coût de calcul sur chacune des listes. Considérons une liste fixée L : elle contient les éléments $\ell_1, \dots, \ell_n$. La complexité totale de l'exécution de l'algorithme sur la liste L est majorée par $\mathrm{C}\,(\ell_i + \dots + \ell_n)$ où C est une constante indépendante de la liste L et des entiers n et N. Ainsi, en sommant sur l'ensemble des listes L, la complexité en moyenne est majorée par

  $$\frac{\mathrm{C}}{\mathrm{N}^n} \sum_{\ell_1=1}^{\mathrm{N}} \cdots \sum_{\ell_n=1}^{\mathrm{N}} (\ell_1 + \dots + \ell_n)$$

  Quitte à réordonner la somme, il suffit de compter le nombre de fois qu'apparaît chaque valeur $k \in [\![1\,;\mathrm{N}]\!]$ parmi les termes ℓ_i. Le nombre total de termes étant $n\,\mathrm{N}^n$, et chaque valeur apparaissant un nombre identique de fois pour des raisons de symétrie, on en déduit que chaque valeur $k \in [\![1\,;\mathrm{N}]\!]$ apparaît $n\,\mathrm{N}^{n-1}$ fois. Ainsi, la complexité en moyenne est majorée par

  $$\frac{\mathrm{C}}{\mathrm{N}^n} \sum_{k=1}^{\mathrm{N}} k\,n\,\mathrm{N}^{n-1} = \frac{\mathrm{C}\,n}{\mathrm{N}} \sum_{k=1}^{\mathrm{N}} k = \frac{\mathrm{C}\,n}{\mathrm{N}} \frac{\mathrm{N}\,(\mathrm{N}+1)}{2} = \frac{\mathrm{C}\,n\,(\mathrm{N}+1)}{2}$$

  Le tri par dénombrement a une complexité en moyenne majorée par $\dfrac{\mathrm{C}\,n\,(\mathrm{N}+1)}{2}$.