X Informatique MP 2003 — Corrigé

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

Il y a beaucoup de code à écrire dans ce problème, qui ne nécessite que peu de connaissances de cours. Il propose de manipuler les arbres (notamment les manières de les parcourir), et fait intervenir de nombreuses récurrences. Certaines questions théoriques, comme la question 9, sont difficiles.

- La première partie consiste en l'écriture de quelques fonctions simples sur les arbres : hauteur d'un sommet, conversion entre deux représentations d'un arbre et *plus proche ancêtre commun*, notion définie dans l'énoncé.
- La deuxième travaille sur les arbres binaires complets. Elle propose un étiquetage des sommets permettant de calculer rapidement le plus proche ancêtre commun de deux sommets en n'utilisant que leurs étiquettes.
- La troisième fait la même étude sur les arbres généraux et définit pour cela les notions d'*arbre gauche* et de *cime*.

Ce sujet est une bonne préparation aux concours, à condition de tester ses fonctions pour qui ne maîtrise pas encore Caml. En effet, c'est principalement l'algorithme qui est pris en compte par le correcteur, mais trop d'erreurs de syntaxe peuvent agacer. Comme souvent, la difficulté est progressive au sein de chaque partie, même si l'alternance de questions théoriques et des fonctions à implanter peut modifier la difficulté perçue. Il ne faut donc pas hésiter à passer à la partie suivante, quitte à revenir sur les dernières questions d'une partie.

Les concepts étudiés dans ce problème servent surtout de support pour faire des exercices et ne sont pas très généraux. On retiendra de ce sujet les techniques mises en œuvre (en particulier les parcours d'arbre et les récurrences) et le style de programmation.

Indications

2 Traduire la définition de la hauteur en Caml, en utilisant un filtrage.

3 Pour chaque sommet, chercher les sommets dont il est le père.

4 Pour `ppacMemeH`, que peut-on dire des pères des deux arguments ?

5 Deviner une formule simple sur de petits exemples et la montrer par récurrence.
Les puissances de 2 ne sont pas loin...

6 Étiqueter d'abord un arbre binaire à trois sommets pour voir comment ça marche.

7 Cette fois, un parcours en profondeur est utile. Préfixe, infixe ou postfixe ?

8 S'inspirer de la question précédente ; une récurrence n'est pas indispensable. Pour $\ell(a)$, quel est l'ensemble des étiquettes de B_a ? Idem pour B_b.
Indication supplémentaire : penser également à la médiane.

9 Procéder en plusieurs étapes :

- Montrer que $r \in [\![p\,;\, q]\!]$.
- Soit g le plus proche ancêtre commun de a et b. Remarquer que tout sommet f tel que $h(f) \leqslant h(g)$ vérifie $\ell(f) \notin [\![p\,;\, q]\!]$. Utiliser la question 7 et raisonner par l'absurde.
- Prouver que
$$\forall c, d \qquad h(c) > h(d) \iff \mathrm{val}_2(\ell(c)) < \mathrm{val}_2(\ell(d))$$
$\mathrm{val}_2(\ell(c))$ est la valuation en 2, c'est-à-dire l'exposant de 2 dans la décomposition en facteurs premiers de $\ell(c)$. Utiliser la question 8 et faire une récurrence sur la hauteur de l'arbre.
- Conclure.

10 Utiliser la question 9 et oublier la représentation arborescente.
Distinguer le cas $p = q$. Pour l'autre cas, ne considérer que les nombres pairs de $[\![p\,;\, q]\!]$ (à justifier) et écrire une fonction récursive.

12 Première question : s'inspirer de la fonction `poids` de la question 11.
Seconde question : faire une récurrence sur le nombre d'ancêtres légers du sommet qui en a le plus.

13 Faire un parcours en profondeur en donnant comme argument à la fonction récursive le plus proche ancêtre léger.
Indication supplémentaire : le traitement d'un sommet consiste à calculer la cime de ses fils (et non la sienne).

15 Encore un parcours d'arbre. La principale difficulté est de s'assurer que $\lambda(\mathrm{cime}[a])$ est calculée lorsque l'on arrive au calcul de $\lambda(a)$.

16 Descendre l'arbre en choisissant le bon fils d'après les couples $(t1(a), t2(a))$. On peut descendre plusieurs étages en une fois.

17.a Étudier, dans l'ordre de profondeur croissante, la nature des sommets b tels que le couple $(t1(b), t2(b))$ fasse partie de $\lambda(a)$, en particulier leurs liens de parenté et leur poids.

17.b S'inspirer de la fonction `trouveSommet` de la question précédente : descendre l'arbre et s'arrêter sur le plus proche ancêtre commun.

L'énoncé commence par plusieurs notations, rappels et définitions de concepts nouveaux. Ces derniers sont utiles dès la première question, qui vise à vérifier leur compréhension. D'autres éléments sont introduits au début des parties suivantes. Certains autres sujets introduisent au contraire tous les nouveaux objets dont ils ont besoin au début ; il est alors bon de ne pas perdre de temps sur ces définitions pour n'y revenir que lorsque l'on aborde les question qui les utilisent.

La définition des types `vint`, `lint` et `vlint` est assez étrange, elle n'apporte aucune information et ne sert apparemment qu'à économiser un peu d'encre. Toute variable de type `vlint` sera ici la représentation d'un arbre par ses listes de fils (sauf une fois) ; on aurait donc pu choisir un nom de type plus explicite (comme `arbre`). Mais il faut bien sûr se conformer à l'énoncé pour ne pas agacer le correcteur.

C'est une erreur de syntaxe en Caml de faire commencer le nom d'une fonction par une majuscule. Il est donc recommandé par les concepteurs du langage eux-mêmes de séparer les mots par des soulignés (exemple : `fils_en_pere`) au lieu de majuscules (`filsEnPere`) comme le fait le sujet.

Les fonctions comme `do_list` ou `map_vect` sont expliquées dans le manuel Caml (rubrique « the core library ») disponible sur `caml.inria.fr`, ou en téléchargement sur `ftp.inria.fr/INRIA/caml-light`. Si vous avez installé Caml sous Windows par le programme cl74win.exe, ce dernier doit avoir créé un manuel dont le nom est proche de « cl74refman ». Ces fonctions sont très utiles et `map` doit absolument être maîtrisée. Elles sont également présentées à la fin de ce corrigé.

I. Fonctions élémentaires

1 Quelques remarques permettent de remplir une bonne partie du tableau :

- Le tableau est symétrique car $\text{ppac}(a, b) = \text{ppac}(b, a)$ pour tous a, b ;
- pour tout a, $\text{ppac}(0, a) = 0$ car le seul ancêtre de la racine est 0 ;
- pour tout a, $\text{ppac}(a, a) = a$.

Le reste se fait en appliquant la définition du plus proche ancêtre commun.

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	4	0	4	4	0	0
2	0	4	2	0	4	2	0	0
3	0	0	0	3	0	0	0	0
4	0	4	4	0	4	4	0	0
5	0	4	2	0	4	5	0	0
6	0	0	0	0	0	0	6	6
7	0	0	0	0	0	0	6	7

Cette question sert, comme dans beaucoup d'autres sujets, à rassurer le candidat en lui faisant manipuler les concepts qui viennent d'être introduits.

> Pour faire un peu de gymnastique et continuer à manipuler ces concepts, on peut voir ppac(a, b) comme la racine du plus petit sous-arbre contenant a et b.
>
> Une session Caml à la fin de ce corrigé montre comment calculer automatiquement ce tableau ; c'est d'ailleurs ce qui a été fait ici.

2 Appliquons la définition de la hauteur :

$$h(a) = \begin{cases} 0 & \text{si } a \text{ est la racine} \\ 1 + h(\text{pere}[a]) & \text{sinon} \end{cases}$$

ce qui se traduit par :

```
let rec hauteur pere = function
  | 0 -> 0
  | a -> 1 + hauteur pere pere.(a)
;;
```

3 Commençons par une évidence : a est le fils de b si et seulement si b est le père de a. Il suffit donc, pour chaque sommet, de le « déclarer » comme père de ses fils.

Précisément, on crée le tableau `peres`, qui sera renvoyé. Puis on définit la fonction `declare_pere` qui prend un sommet et parcourt la liste de ses fils. Pour chaque fils, elle écrit qui est son père dans le résultat. On l'applique alors à tous les sommets. Il faut enfin régler le cas de la racine, la convention étant ici qu'elle a un père.

```
let filsEnPere fils =
  let n = vect_length fils in
  let peres = make_vect n (-1) in
  let declare_pere p =
    do_list (fun son -> peres.(son) <- p) fils.(p)
  in
  for p=0 to n-1 do
    declare_pere p
  done;
  peres.(0) <- 0;
  peres
;;
```

> Le tableau `peres` est initialisé avec -1, ce qui aide à déboguer : si le résultat contient encore des -1, c'est qu'il y a eu un problème. Cela ne marche que si les sommets ont toujours des numéros positifs.
>
> Un parcours d'arbre est inutile pour cette question.

4 Définissons une fonction récursive : si les deux sommets sont égaux, alors leur plus proche ancêtre commun est l'un des deux, sinon c'est le plus proche ancêtre commun de leurs pères, qui sont à la même hauteur.

```
let rec ppacMemeH pere a b =
  if a = b
  then a
  else ppacMemeH pere pere.(a) pere.(b)
;;
```