ÉCOLE POLYTECHNIQUE **FILIÈRE MP**
OPTION INFORMATIQUE

CONCOURS D'ADMISSION 2003

COMPOSITION D'INFORMATIQUE

(Durée : 4 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.
On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

⋆⋆⋆

Routage dans un réseau arborescent

Dans ce problème on aborde une question soulevée dans la conception des réseaux : il s'agit d'affecter des adresses distinctes aux nœuds d'un réseau de façon telle que la route entre deux nœuds puisse être déterminée par un calcul utilisant uniquement leurs deux adresses, sans référence à une connaissance globale du réseau. On s'intéresse plus particulièrement aux réseaux ayant une forme d'arbre.

Le problème est composé de trois parties indépendantes.

Dans tout le problème un *n-arbre* est un arbre dont l'ensemble des *sommets* (on dit aussi parfois les nœuds) est $\{0, 1, \ldots, n-1\}$, et dont la racine est 0. Chaque sommet a possède un *père* noté $pere[a]$, par convention la racine est son propre père, ainsi $pere[0] = 0$. L'ensemble des n-arbres est noté $\mathcal{A}(n)$.

L'ensemble des *ancêtres* du sommet a est constitué de a et des ancêtres de son père $pere[a]$; la racine 0 est ainsi ancêtre de tous les sommets de l'arbre. Le *sous-arbre de racine* a, noté A_a, est formé de tous les sommets de l'arbre A dont a est ancêtre.

Les *fils* d'un sommet a sont ainsi les $b \neq 0$ tels que $pere[b] = a$, (attention la racine n'est pas fils d'elle même). Une *feuille* est un sommet qui n'a pas de fils.

La *hauteur* (on dit aussi parfois profondeur) d'un sommet a, notée $h(a)$ est un entier égal à 0 si a est la racine, elle est égale à la hauteur du père de a augmentée de 1 sinon. Le *plus proche ancêtre commun* à deux sommets a et b, noté $ppac(a, b)$, est le sommet de hauteur maximale qui est à la fois ancêtre de a et de b. Noter que si a est ancêtre de b alors $ppac(a, b) = a$.

Dans tout le problème, on considère que l'entier n est fixé, d'autre part pour un tableau t, t[i] dénote l'élément d'indice i ; une liste contenant les entiers $x_0, x_1, \ldots, x_p$ est représentée par $\langle x_0, x_1, \ldots, x_p \rangle$; la liste vide est notée $\langle\ \rangle$.

On utilisera pour un n-arbre A :

– Un tableau de listes d'entiers fils de taille n tel que fils[i] est la liste des fils de i.

– Un tableau d'entiers pere de taille n tel que pere[i] est le père de i.

On a ainsi en Caml et en Pascal

```
let n = .....;;

type vint == int vect ;;
type lint == int list ;;
type vlint == lint vect ;;
```

```
const n = ...;
type  vint = array[0..n-1] of integer ;
      lint   = ^cellule;
      cellule = record
                val : integer; suiv : lint;
      end;
      vlint = array[0..n-1] of lint;
```

On pourra utiliser en Pascal le constructeur pour les listes qui est donné ci-dessous :

```
function cons (contenu : integer; suivant : lint) : lint;
     begin
     var res : lint;
     new(res); res^.val := contenu;   res^.suiv := suivant;
     cons := res;
end;
```

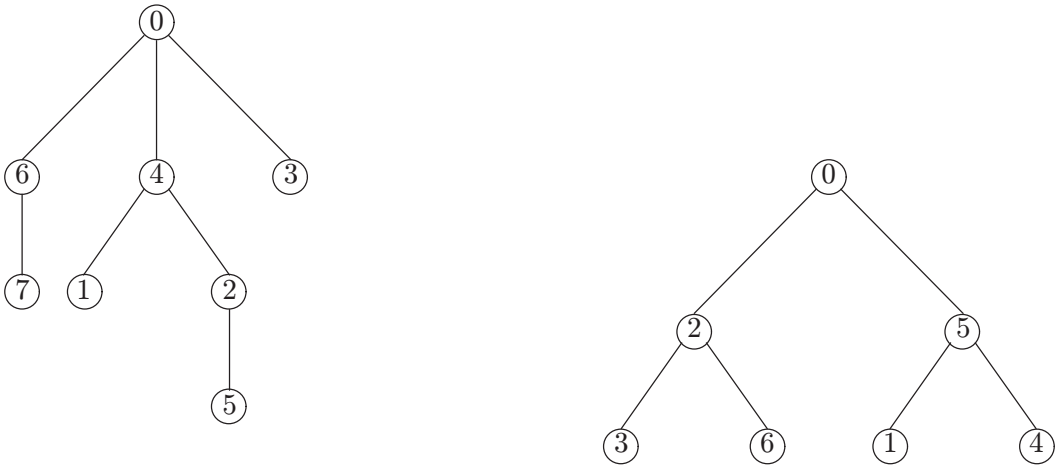

FIG. 1: Un arbre $A \in \mathcal{A}(8)$ et un arbre binaire complet $B \in \mathcal{B}(2)$.

Les tableaux pere et fils de l'arbre A situé à gauche de la Figure 1 sont donnés par :

i	0	1	2	3	4	5	6	7
pere	0	4	4	0	0	2	0	6
fils	$\langle 6,4,3\rangle$	$\langle\ \rangle$	$\langle 5\rangle$	$\langle\ \rangle$	$\langle 1,2\rangle$	$\langle\ \rangle$	$\langle 7\rangle$	$\langle\ \rangle$

I. Fonctions Élémentaires

Question 1. Calculer $ppac(i, j)$ pour tous les couples de sommets de l'arbre A donné plus haut. On présentera le résultat sous forme d'un tableau de 8 lignes et 8 colonnes tel que la case sur la ligne i et la colonne j contienne le plus proche ancêtre de i et j.

Question 2. Écrire une fonction [1]

```
hauteur : vint -> int  -> int
function hauteur (pere : vint; a : integer) : integer
```

qui étant donné un sommet a d'un arbre, pour lequel on donne son tableau des pères, calcule la hauteur de ce sommet.

Question 3. Écrire une fonction

```
filsEnPere : vlint -> vint
procedure filsEnPere (fils : vlint; var pere : vint)
```

qui à partir d'un arbre, donné par ses listes de fils, calcule le tableau des pères.

Question 4. Écrire une fonction

```
ppacMemeH : vint -> int  -> int -> int
function ppacMemeH (pere: vint; a, b: integer) : integer
```

qui calcule le plus proche ancêtre commun de deux sommets a et b supposés de même hauteur dans un arbre pour lequel on donne le tableau des pères.

En déduire une fonction ppac qui effectue le même calcul pour deux sommets a et b n'ayant pas nécessairement la même hauteur.

II. Arbres binaires complets

Un *arbre binaire complet* est un arbre dans lequel tout sommet qui n'est pas une feuille a exactement deux fils, et tel que toutes les feuilles sont à la même hauteur. On note $\mathcal{B}(h)$ l'ensemble des n-arbres binaires complets dans lesquels les feuilles sont à hauteur h. Un exemple d'arbre binaire complet B est donné à droite de la Figure 1.

Question 5. Déterminer pour $0 \leqslant k \leqslant h$ le nombre s_k de sommets de hauteur k dans un arbre B de $\mathcal{B}(h)$, justifiez votre réponse. En déduire le nombre n de sommets d'un arbre de $\mathcal{B}(h)$ en fonction de h.

Dans la suite B est un arbre de $\mathcal{B}(h)$ ayant n sommets ; à chaque sommet a de B on associe une étiquette $\ell(a) \in \{1, 2, \ldots, n\}$ satisfaisant la condition suivante :

pour chaque sommet a ayant pour liste de fils $\langle b, c\rangle$:

$$\max_{u \in B_b} \ell(u) < \ell(a) < \min_{v \in B_c} \ell(v)$$

[1] Dans tout l'énoncé du problème les déclarations de fonctions et procédures sont proposées d'abord en Caml puis en Pascal.

Question 6. Calculer les valeurs de $\ell(a)$ pour les 7 sommets de l'arbre binaire complet représenté sur la droite de la Figure 1 : l'ordre des fils dans les listes correspond à la lecture de gauche à droite de la figure.

Question 7. Écrire une fonction

```
etiquettes : vlint -> vint
procedure etiquettes (fils : vlint; var etiq : vint)
```

qui calcule les étiquettes $\ell(a)$ pour tous les sommets d'un arbre de $\mathcal{B}(h)$ donné par ses listes de fils.

Question 8. Un arbre B de $\mathcal{B}(h)$ de racine 0 ayant pour liste de fils $\langle a, b\rangle$ se décompose en deux sous-arbres B_a et B_b. Déterminer les valeurs de $\ell(0)$, de $\ell(a)$ et de $\ell(b)$ pour les fils a et b de la racine.

Question 9. Démontrer que si a et b sont deux sommets quelconques de B alors $r = \ell(ppac(a, b))$ est déterminé de manière unique à partir de $p = \ell(a)$ et $q = \ell(b)$ par la propriété suivante :

r est parmi les entiers compris (au sens large) entre p et q celui possédant le plus grand diviseur qui soit une puissance de 2.

Question 10. Donner une fonction récursive

```
mu : int -> int -> int
function  mu (p, q : integer) : integer
```

qui détermine $r = \ell(ppac(a, b))$ à partir de $p = \ell(a)$ et $q = \ell(b)$; on supposera que $p \leqslant q$. Votre fonction doit être de complexité $O(\log_2(q - p))$.

III. Arbres généraux

Dans cette partie on utilise les définitions et notations suivantes :

Dans un arbre A le *poids* $w[a]$ d'un sommet a est le nombre de sommets du sous-arbre A_a de racine a, ainsi $w[a] = 1$ si et seulement si a est une feuille, noter que $w[0] = n$ (c'est le nombre de sommets
de A).

Un arbre A est dit *gauche* si pour chaque sommet a (qui n'est pas une feuille) le premier élément de la liste des fils est de poids supérieur ou égal à celui de tous les autres sommets de cette liste. Dans un arbre gauche, le premier élément de la liste des fils d'un sommet est dit *lourd*, les autres sommets sont dits *légers*. La racine est un sommet léger.

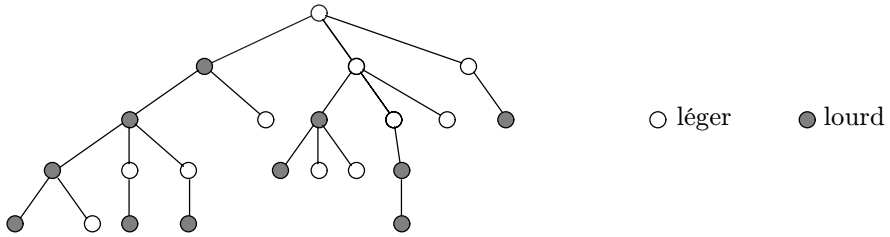

FIG. 2: Sommets lourds et légers.

Question 11. Écrire une fonction

```
poids : vlint -> vint
procedure poids(fils : vlint; var poids : vint)
```

qui calcule les poids de tous les sommets d'un arbre donné par ses listes de fils. Puis une fonction :

```
gauchir : vlint -> vint -> vlint
procedure gauchir (fils : vlint; w : vint; var filsG : vlint)
```

qui calcule un arbre gauche obtenu en réordonnant les fils de chaque sommet de façon telle que le premier fils soit de poids supérieur ou égal à celui des autres. Sont donnés dans cette fonction les listes des fils et le tableau des poids des sommets de l'arbre.

Question 12. Soit a un sommet léger d'un arbre gauche A et b son père, montrer que $w[b] > 2w[a]$. En déduire que le nombre d'ancêtres légers de a est inférieur à $1 + \log_2 n$.

On utilise dans toute la suite la notion de *cime* d'un sommet.

- Si a est léger $cime[a]$ est égale à $pere[a]$.
- Si a est lourd $cime[a]$ est le plus proche ancêtre de a qui est léger.

Ainsi en raison de nos conventions $cime[0] = 0$. On appelle A' l'arbre gauche obtenu en appliquant la fonction `gauchir` à l'arbre A de la figure 1. Les cimes des sommets de l'arbre A' sont alors données par le tableau suivant :

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

Question 13. Écrire une fonction

```
cimes  : vlint -> vint
procedure cimes (fils : vlint; var cime : vint)
```

qui calcule les sommets cimes de tous les sommets à partir des listes de fils d'un arbre gauche.

On se propose d'attribuer des étiquettes aux sommets d'un arbre gauche A. Pour cela on commence par construire deux tableaux d'entiers $t1$ et $t2$ associés aux sommets. Le tableau $t1$ est tel que $t1[0] = 0$ et $t1[a] = i$ si a est le i-ème élément de la liste des fils de son père.

Le tableau $t2$ est tel que $t2[a] = 0$ si a est léger et $t2[a] = t2[pere[a]] + 1$ si a est lourd.

Les étiquettes $\lambda(a)$ des sommets sont alors des listes d'entiers construites de la façon suivante :

$$\lambda(0) = \langle\rangle \qquad \text{et pour} \quad a \neq 0 \quad \lambda(a) = \lambda(cime[a]) \circ \langle t1[a], t2[a]\rangle$$

dans cette formule $\circ$ dénote la concaténation des listes.

Question 14. Donner l'arbre A' défini à la question 12 ; puis donner les valeurs de $t2[a]$ pour tous les sommets a ; calculer enfin leurs étiquettes $\lambda(a)$.

Question 15. Écrire une fonction

```
etiquettes  : vlint -> vint -> vlint
procedure etiquettes (fils : vlint; cime : vint; var lambda : vlint)
```

qui calcule les étiquettes des sommets d'un arbre gauche donné par ses listes de fils et pour lequel on donne aussi le tableau des cimes des sommets.

En Caml on pourra utiliser l'opérateur @ et en Pascal la fonction :

```
function concat (x, y : lint): lint;
```

qui calcule la concaténation de deux listes et dont on ne demande pas le corps.

Question 16. Écrire une fonction :

```
trouve :  vlint -> lint -> int
function trouveSommet(fils : vlint; etiq : lint): integer
```

qui, à partir de l'étiquette d'un sommet d'un arbre gauche A et des listes de fils des sommets de cet arbre, détermine ce sommet.

Question 17.

a) Montrer que, pour tout sommet a d'un arbre, la longueur de l'étiquette $\lambda(a)$ est majorée par 4 fois le nombre de ses ancêtres légers.

b) Indiquer comment on calcule $\lambda(ppac(a, b))$ à partir des deux listes $u = \lambda(a)$ et $v = \lambda(b)$; on ne demande pas de justifier la réponse à cette question.

On a donné dans ce problème une technique permettant d'étiqueter les nœuds des sommets d'un réseau qui a une forme d'arbre. Dans cet étiquetage la recherche du plus proche ancêtre commun de deux sommets (et donc de la route qui les relie) s'effectue uniquement à l'aide de leurs étiquettes. Des techniques plus complexes, qui généralisent les techniques données ici, utilisent des étiquettes de taille $O(\log_2 n)$ bits qui permettent aussi un calcul du plus proche ancêtre commun en $O(1)$ opérations.

* *
*

Rapport de M^{me} Anne Dicky, correctrice.

Rappelons que, si tous les candidats ayant choisi l'option Informatique subissent l'épreuve, seules les copies des candidats admissibles sont corrigées.

J'ai corrigé 216 copies de candidats français. 25 candidats avaient choisi d'écrire leurs programmes en langage Pascal.

L'épreuve

Il s'agissait d'un problème de routage statique : attribuer une adresse à chacun des nœuds d'un réseau, de façon qu'un nœud devant transmettre un message à un autre puisse déterminer, au vu de l'adresse du destinataire, par lequel de ses voisins il doit faire transiter le message (chaque nœud ne connaissant du réseau que ses voisins, et sa propre adresse). La question cruciale est bien entendu la rapidité des calculs de routage.

Le sujet, dont on trouvera plus loin un commentaire détaillé, n'était ni très long (36 candidats ont traité au moins partiellement chacune des questions) ni difficile (de nombreuses questions étaient soit de simples applications des définitions, soit des exercices très classiques sur des points du cours). La plupart des candidats avaient un niveau suffisant pour répondre correctement aux questions, mais n'ont pas compris l'enjeu du problème (on trouve çà et là des réflexions telles que « la complexité pourrait être améliorée mais ce n'est pas demandé » ; une seule copie fait référence au réseau Internet).

Pour obtenir la note maximale il était nécessaire de traiter le problème en entier. Seuls deux candidats ont obtenu la note 20.

La moyenne des notes est de 10,6 avec un écart-type de 4. La moyenne est sensiblement plus faible (9,9) pour les candidats qui ont composé en Pascal. La répartition des notes s'établit ainsi :

	CAML	Pascal	Ensemble
$0 \leq N < 4$	5%	0%	5%
$4 \leq N < 8$	23%	10%	25%
$8 \leq N < 12$	34%	36%	34%
$12 \leq N < 16$	24%	12%	22%
$N \geq 16$	14%	12%	14%

Remarques générales

Dans l'ensemble, les copies étaient soignées et agréables à lire, péchant plutôt par excès de concision (manque de justification des algorithmes utilisés, démonstrations réduites à des constatations). La plupart des candidats maîtrisent la notion de récursivité et l'algorithmique des listes. Ils sont beaucoup moins à l'aise dans les arbres (la structure de données n'est pas vue comme récursive), et plus généralement dans le passage de

l'algorithmique, où ils font preuve de réflexion et d'imagination, à la programmation, qui semble perçue comme une corvée de routine. La notion de complexité asymptotique est rarement maîtrisée (peu s'enhardissent à des calculs, encore sont-ils souvent faux).

Les candidats qui programment en Pascal, ou en CAML avec un fort accent Pascal, commettent peu d'erreurs, mais semblent ne concevoir la programmation que comme une mise en forme répétitive (à tous les sens du terme) du cours d'algorithmique, et s'arrêtent dès qu'ils ne « savent plus faire ».

La plupart de ceux qui programment en CAML réinventent à longueur de copie `map`, `do_list` et `it_list`. Peut-être n'utilisent-ils pas les fonctions de bibliothèque de peur de se tromper dans leur maniement, ou de susciter l'ire du correcteur ; une façon très simple de dissiper ces craintes (systématiquement employée dans les meilleures copies) consiste à donner en annexe une implémentation de ces fonctions.

Dans cette épreuve, les erreurs de syntaxe et de type les plus fréquentes étaient des erreurs vénielles relatives aux fonctions de bibliothèque `vect_item` et `vect_assign`, dont beaucoup de candidats n'avaient probablement jamais entendu parler (aucune copie ne mentionne même le nom de ces fonctions).

D'autre part, l'identification de motifs est trop systématiquement employée, donnant lieu tantôt à des complications ridicules (préférer « `if a=b then...` » à « `match (a,b) with (x,y) when x=y ->...` » permettrait au moins d'éviter l'erreur classique « `match (a,b) with (a,a) ->...` »), tantôt à des erreurs révélant une incompréhension de la notion (j'ai ainsi relevé un curieux « `match n with p+1 -> p` »).

Commentaire détaillé

Le problème comportait 17 questions, dont plusieurs destinées à faciliter la compréhension des questions suivantes (ou à éviter d'y commettre des erreurs grossières), et qui n'ont été comptées que dans la mesure où les questions suivantes avaient été traitées (au moins partiellement) : c'est ainsi que les rares copies dont l'auteur n'avait montré aucune connaissance en algorithmique ni en programmation ont obtenu des notes particulièrement basses.

Pour les questions comportant des programmes à écrire, les critères principaux étaient la correction de l'algorithme sous-jacent et l'effectivité du code (tout programme non conforme aux spécifications de l'énoncé, aux erreurs de syntaxe et aux étourderies près, a été noté en dessous de la moyenne) ; les critères secondaires étaient l'efficacité du programme (en particulier, les fonctions d'étiquetage de complexité quadratique ou pire ne pouvaient rapporter que la moitié des points de la question), sa lisibilité et sa justification (quand le passage des spécifications à l'algorithme n'était pas évident).

Pour chaque groupement de questions sont indiqués entre crochets le pourcentage de candidats ayant obtenu au moins la moitié des points correspondants, et, s'il n'est pas

nul, le pourcentage de candidats ayant obtenu la note 0 (que la question ait été traitée ou non).

Partie I [91%]

D'assez nombreux candidats ont été décontenancés par les définitions de « père » et d'« ancêtre », certains oubliant qu'un sommet était son propre ancêtre, d'autres considérant que la racine était son propre père (en raison de la convention `pere[0] = 0`) : ce qui a donné lieu tantôt à des erreurs, tantôt à des critiques sans objet de certains énoncés.

Questions 1 et 2 [95%]

Ne présentaient aucune difficulté.

Question 3 [75% - zéros : 6%]

Pour convertir au format « tableau des pères » un arbre donné sous le format « tableau des listes de fils » il suffit d'un seul parcours de l'arbre.

Question 4 [92%]

La fonction `ppacMemeH` se prêtait à une implémentation récursive évidente (si deux sommets distincts ont même hauteur, leur plus petit ancêtre commun est aussi celui de leurs pères) ; on pouvait en déduire une fonction `ppac` sous la forme

```
let rec ppac pere a b =
    let ha = hauteur pere a and hb = hauteur pere b in
        if ha = hb then ppacMemeH pere a b
        else if ha > hb then ppac pere pere.(a) b
        else ppac pere a pere.(b)
```

mais cette implémentation fait appel à des calculs de hauteurs évitables (si l'on connaît la hauteur d'un sommet on peut en déduire celle de son père). Une version plus efficace consistait à transmettre les hauteurs calculées :

```
let ppac pere a b =
       let rec ppacAvecH a b ha hb =
           if ha = hb then ppacMemeH pere a b
           else if ha > hb then ppacAvecH pere.(a) b (ha - 1) hb
           else ppacAvecH a pere.(b) ha (hb - 1)
       in ppacAvecH a b (hauteur pere a) (hauteur pere b)
```

Partie II [42%]

Question 5 [95% - zéros : 1%]

Question sans aucune difficulté.

L'ordre des questions 6 à 9 correspondait à une certaine démarche : constater sur un exemple que ranger dans un arbre binaire de recherche équilibré les nombres de 1 à $2^h - 1$ revenait à numéroter les sommets dans l'ordre infixe (question 6) et programmer cet étiquetage (question 7), puis établir (question 8) un résultat permettant de caractériser l'étiquette du ppac (question 9). Beaucoup de candidats ont suivi une autre démarche, qui les conduisait à modifier l'ordre de traitement des questions.

Questions 6 et 7 [50% - zéros : 4%]

Moins de la moitié de candidats ont utilisé (ou réinventé) le parcours infixe. La plupart ont utilisé une formule récursive permettant de calculer les étiquettes dans un parcours préfixe : encore fallait-il se donner la peine sinon de démontrer cette formule, du moins de l'expliciter. Une bonne solution consistait à programmer l'étiquetage bijectif d'un arbre binaire complet sur un intervalle (l'étiquette de la racine étant le milieu de l'intervalle). Une moins bonne (dans la mesure où cela conduisait à un calcul de puissances[1]) était de programmer l'étiquetage d'un arbre binaire complet de hauteur h à partir d'un numéro d, l'étiquette de la racine étant alors $d - 1 + 2^h$.

Question 8 [91% - zéros : 4%]

Question sans aucune difficulté.

Question 9 [38% - zéros : 41%]

Question difficile à traiter correctement, sans se contenter d'affirmations non démontrées. Ainsi, l'inégalité $p \leq r \leq q$ a souvent été traitée avec légèreté (« évident d'après la définition de l'étiquetage ») : encore fallait-il établir que si deux sommets ne sont pas en ligne directe, alors l'un a pour ancêtre le fils droit et l'autre le fils gauche de leur ppac. À l'inverse, dans des copies par ailleurs très bonnes on est allé jusqu'à « démontrer » des propriétés telles que « $l(a) \leq l(c) \leq l(b)$ si et seulement si c est un ancêtre commun à a et b », ce qui annulait l'ensemble de la question.

Beaucoup de candidats, c'est tout à leur honneur, se sont donné la peine de justifier la formulation de l'énoncé : « celui des entiers de $[p, q]$ qui est divisible par la plus grande puissance de 2 », en démontrant l'unicité de l'entier ainsi défini.

Pour caractériser l'étiquette du ppac, la plupart des candidats ont soit constaté, soit démontré par récurrence, une relation entre hauteurs et étiquettes (les sommets de hauteur k sont les sommets d'étiquette $2^{h-k}m$ où m est un entier impair inférieur à 2^{k+1}). Le raisonnement le plus élégant était une induction structurelle (une vraie), remarquant que deux sommets d'un arbre de hauteur h, si leur plus petit ancêtre commun n'est pas la racine, sont soit tous deux dans le sous-arbre gauche (avec les étiquettes qu'ils auraient dans ce sous-arbre), soit tous deux dans le sous-arbre droit (avec les étiquettes augmentées de $2^h + 1$, translation qui conserve la propriété caractérisant l'étiquette du ppac).

[1] Rappelons que ni `2**h`, ni `2^h`, ne sont acceptables dans un programme, que ce soit en CAML ou en Pascal.

Question 10 [24% - zéros : 65%]

Il suffisait d'écrire l'inégalité $p \leq 2^i m \leq q$ pour concevoir une définition récursive de $\mu(p, q)$: p si $p = q$, $2\mu(\lceil \frac{p}{2} \rceil, \lfloor \frac{q}{2} \rfloor)$ sinon.

Plusieurs candidats ont proposé des algorithmes de complexité $O(\log_2 q)$, ce qui était acceptable (la recherche brutale à partir de la caractérisation, au mieux linéaire en $q - p$, ne l'était pas) : il est en effet difficile de concevoir que des opérations arithmétiques sur des données de taille $\log_2 q$ puissent être faites en temps constant (ce qui a cependant été indiqué dans la meilleure copie).

Aucune réponse comportant une évaluation inacceptable de la complexité n'a été considérée. (Peut-être faudrait-il bannir la formule « diviser pour régner », dont beaucoup de candidats se servent pour justifier n'importe quel algorithme récursif, et qui les conduit à affirmer qu'aller de Paris à Lyon, puis de Lyon à Marseille, prend deux fois moins de temps que le trajet direct Paris-Marseille.)

Partie III [36%]

Question 11 [57% - zéros : 16%]

Poids [31% - zéros : 34%]

Cette question était notée 0 pour un algorithme faux, et moins de la moyenne pour un algorithme non linéaire. Le résultat laisse à penser qu'on a tort d'imputer à la seule incurie des énarques la difficulté d'estimer les effectifs de la Fonction Publique.

La proportion de zéros est sensiblement moindre (20%) pour les candidats ayant choisi le langage Pascal : calculer récursivement les poids des sommets fils, puis faire la somme, est une procédure naturelle. Ceux qui programment en CAML (sans accent Pascal) tentent de se limiter à une seule récursive (généralement appelée `aux`, ce qui ne facilite pas la détection d'erreurs). C'est possible, par exemple sous la forme

```
let poids = function fils ->
    let tabPoids = make_vect n 0 in
      let rec poidsCommeSiLesFilsEtaient = function
          [] -> 1
        | s::q -> let p = poidsCommeSiLesFilsEtaient fils.(s)  in
                      tabPoids.(s) <- p ; p + poidsCommeSiLesFilsEtaient q
      in tabPoids.(0) <- poidsCommeSiLesFilsEtaient fils.(0) ; tabPoids ;;
```

mais il n'est pas évident d'expliciter la sémantique de `poidsCommeSiLesFilsEtaient f` (qui retourne le poids d'un arbre virtuel V dont la racine aurait pour fils les éléments de la liste `f`, après avoir correctement affecté `tabPoids[i]` pour tous les sommets `i` de Vdistincts de la racine) ; et il suffit de se tromper dans l'initialisation pour mettre tout à zéro, ou ne compter que les feuilles, ou n'importe quoi.

Gauchir [57% - zéros : 26%]

Beaucoup de programmes modifiaient l'arbre donné en paramètre au lieu d'en construire un nouveau, ce qui a été compté comme une erreur. En effet, même si l'énoncé de la question (« calculer un arbre gauche obtenu en réordonnant les fils de chaque sommet ») pouvait prêter à équivoque (ce n'était pas le cas en Pascal), le fait que les données soient modifiées en cours d'exécution nécessite un mot de justification (qui n'a été donné que par un candidat) sur les invariants de chaque modification.

La plupart des candidats ayant traité la question ont évité de trier les listes, et proposé un algorithme linéaire.

Question 12 [72% - zéros : 3%]

La première inégalité (le poids du père d'un sommet léger a est supérieur à deux fois celui de a) était facile à établir. La seconde nécessitait plus de soin : pour pouvoir la déduire de la première, il fallait remarquer que, même si un sommet est lourd, il est de poids inférieur à celui de son père. Au lieu de quoi beaucoup de candidats n'ont démontré la propriété que pour les sommets dont tous les ancêtres sont légers, arguant qu'il s'agissait du « pire » (parfois appelé « meilleur ») des cas : un type de raisonnement par lequel on pourrait prouver, par exemple, que « tout entier positif p a moins de 2 diviseurs premiers » (en effet, « dans le pire des cas », tous les diviseurs de p supérieurs à 1 sont premiers, mais alors p lui-même est premier, donc a un seul diviseur premier).

Question 13 [62% - zéros : 21%]

La plupart des candidats ont directement utilisé la définition récursive de la cime, ce qui nécessitait que l'on connût le père de chaque sommet. Trop nombreux sont ceux qui, négligeant le coût en temps et en espace de la conversion de format, ont écrit des tests `estLeger` appelant systématiquement la procédure `filsEnPere`. En fait, si on connaît la cime et le rang d'un sommet, on peut en déduire la cime de son fils lourd et celles de ses fils légers : le calcul des cimes pouvait se faire en un seul parcours récursif.

Questions 14 et 15 [51% - zéros : 21%]

La question 14 ne présentait aucune difficulté.

Question 15 : même parmi les candidats maîtrisant le parcours en profondeur, peu ont écrit une fonction attribuant les étiquettes en un seul parcours de l'arbre ; et parmi ceux-là, aucun n'a pensé à décentraliser le calcul des étiquettes (le père transmettant à chacun de ses fils toutes les informations nécessaires), ce qui n'aurait bien sûr rien changé à l'ordre de complexité de l'algorithme.

Question 16 [25% - zéros : 64%]

Il suffisait d'interpréter l'étiquette comme une suite de déplacements dans l'arbre à partir de la racine. Une recherche linéaire dans l'océan des étiquettes permettait de grappiller quelques points en CAML (non en Pascal, à moins de définir l'égalité des listes). Quelques candidats ont repris la définition récursive pour analyser l'étiquette de droite

à gauche, ce qui exigeait des manipulations aussi complexes (et exaspérantes) que la recherche de la *définition* 1.1.3.2.1 lorsqu'on a réussi à trouver le *théorème* 1.1.3.2.1.

Question 17 [20% - zéros : 68%]

La première partie de la question (établir la majoration) a été généralement bien traitée. La deuxième partie (expliquer le calcul de l'étiquette du petit ancêtre commun) s'est souvent limitée à « on prend le plus grand préfixe commun », ce qui ne convenait pas dans le cas où ce préfixe était de longueur impaire (il fallait alors ajouter au préfixe commun le plus petit des entiers qui le suivaient dans les étiquettes).