Corrigé : routage dans un réseau arborescent (X 2003)

Partie I. Fonctions élémentaires

Question 1. Le tableau demandé étant à l'évidence symétrique, on n'en remplit que la moitié :

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

Question 2.

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

Question 3.

```
let filsEnPere f =
  let n = vect_length f in
  let p = make_vect n 0 in
  let rec aux j = function
    | []   -> ()
    | i::q -> p.(i) <- j ; aux j q
  in for j = 0 to n-1 do aux j f.(j) done ;
  p ;;
```

Question 4. Si les deux sommets a et b ont même hauteur, leurs pères aussi, ce qui conduit à la définition inductive suivante :

$$\text{ppacMemeH}(a,b) = \begin{cases} a & \text{si } a = b \\ \text{ppacMemeH}(p[a], p[b]) & \text{sinon} \end{cases}$$

```
let rec ppacMemeH p = fun
  | a b when a = b -> a
  | a b            -> ppacMemeH p p.(a) p.(b) ;;
```

Si les deux sommets a et b n'ont pas même hauteur, supposons par exemple $h(a) < h(b)$. Dans ce cas, le plus proche ancêtre commun à a et b est aussi le plus proche ancêtre commun à a et au père de b. Ceci conduit à la définition inductive suivante :

$$\text{ppac}(a,b) = \begin{cases} \text{ppacMemeH}(a,b) & \text{si } h(a) = h(b) \\ \text{ppac}(a, p[b]) & \text{si } h(a) < h(b) \\ \text{ppac}(p[a], b) & \text{si } h(b) < h(a) \end{cases}$$

```
let ppac p a b =
  let rec aux = fun
    | (a, ha) (b, hb) when ha = hb -> ppacMemeH p a b
    | (a, ha) (b, hb) when ha < hb -> aux (a, ha) (p.(b), hb - 1)
    | (a, ha) (b, hb)              -> aux (p.(a), ha - 1) (b, hb)
  in aux (a, hauteur p a) (b, hauteur p b) ;;
```

Bien entendu, on ne calcule qu'une seule fois la hauteur de chacun des deux sommets.

Partie II. Arbres binaires complets

Question 5. Tout sommet de hauteur $k < h$ n'est pas une feuille donc a deux fils, ce qui conduit à la relation de récurrence $s_{k+1} = 2s_k$. Sachant que $s_0 = 1$ on en déduit immédiatement que $s_k = 2^k$ pour $k \in [\![0, h]\!]$.
Le nombre n de sommets d'un arbre de $\mathscr{B}(h)$ vaut donc : $n = \sum_{k=0}^{h} s_k = \sum_{k=0}^{h} 2^k = 2^{h+1} - 1$.

Question 6. Il s'agit de munir B d'une structure d'arbre binaire de recherche à l'aide d'étiquettes deux à deux distinctes. Il suffit donc de parcourir B par ordre infixe pour obtenir l'étiquetage suivant :

a	0	1	2	3	4	5	6
$\ell(a)$	4	5	2	1	7	6	3

Question 7. Dans le cas d'un arbre binaire complet, chaque liste du tableau f est vide (les fils d'une feuille) ou de longueur 2 (les fils d'un nœud). Le parcours infixe prend donc la forme suivante :

```
let etiquettes f =
  let n = vect_length f in
  let l = make_vect n 0 in
  let rec aux k a = match f.(a) with
    | [fg; fd] -> let j = aux k fg in l.(a) <- j ; aux (j+1) fd
    | _        -> l.(a) <- k ; k + 1
  in let _ = aux 1 0 in l ;;
```

La fonction **aux** renvoie la valeur de l'indice à affecter dans la suite du parcours infixe.

Question 8. B_a et B_b sont des arbres binaires complets de hauteur $h-1$ qui possèdent $2^h - 1$ sommets. On a donc $\ell(0) = 2^h$, $\ell(a) = 2^{h-1}$ et $\ell(b) = 2^h + 2^{h-1}$.

Question 9. Considérons un nœud x de hauteur $h_x < h$ et ses deux fils y (son fils gauche) et z (son fils droit).
y est la racine d'un arbre binaire complet ; son fils droit (s'il existe) est la racine d'un arbre à $2^{h-h_x-1} - 1$ sommets. On a donc $\ell(x) = \ell(y) + 2^{h-h_x-1}$.
De même, z est la racine d'un arbre binaire complet et son fils gauche (s'il existe) est la racine d'un arbre à $2^{h-h_x-1} - 1$ sommets donc $\ell(z) = \ell(x) + 2^{h-h_x-1}$.
Posons maintenant pour tout sommet s, $\ell(s) = 2^{h-h_s}\ell'(s)$. Les relations ci-dessus se simplifient en : $\ell'(y) = 2\ell'(y) - 1$ et $\ell'(z) = 2\ell'(x) + 1$. Sachant en outre que $\ell(0) = 2^h$ et donc que $\ell'(0) = 1$ nous venons de prouver que pour tout sommet s, $\ell'(s)$ est un entier impair et donc que $h - h_s$ *est la valuation 2-adique de* $\ell(s)$.
Considérons maintenant deux sommets a et b de hauteurs respectives h_a et h_b, c leur plus petit ancêtre commun, de hauteur h_c. Si $c \notin \{a, b\}$ on peut sans perte de généralité supposer que a appartient au fils gauche de c et b à son fils droit ; on a dans ce cas $\ell(a) < \ell(c) < \ell(b)$. Dans le cas où $c \in \{a, b\}$ on a $\ell(c) = \ell(a)$ ou $\ell(c) = \ell(b)$ et donc dans tous les cas, $\ell(a) \leqslant \ell(c) \leqslant \ell(b)$, soit $p \leqslant 2^{h-h_c}\ell'(c) \leqslant q$.
Considérons maintenant un entier $r \in [\![p, q]\!]$; il existe un sommet x tel que $r = \ell(x) = 2^{h-h_x}\ell'(x)$. Lors du parcours infixe de l'arbre x est parcouru après a et avant b donc fait partie des descendants de c. Sa hauteur est donc supérieure à celle de c, ce qui prouve que $h - h_x \leqslant h - h_c$, avec égalité si et seulement si $x = c$. $\ell(c)$ est donc bien l'unique élément de valuation 2-adique maximale dans l'intervalle $[\![p, q]\!]$.

Question 10. La complexité demandée suggère une approche « diviser pour régner ». On cherche le plus grand entier k et un entier impair r' tels que $p \leqslant 2^k r' \leqslant q$.
Si $p = q$ on a nécessairement $2^k r' = p = q$; si $p < q$ il y a forcément un entier pair dans $[\![p, q]\!]$ et donc $k \geqslant 1$. L'encadrement ci-dessus est donc équivalent à : $\left\lceil \frac{p}{2} \right\rceil \leqslant 2^{k-1} r' \leqslant \left\lfloor \frac{q}{2} \right\rfloor$. D'où la fonction :

```
let rec mu = fun
  | p q when p = q -> p
  | p q            -> 2 * (mu ((p + 1) / 2) (q / 2)) ;;
```

Partie III. Arbres généraux

Question 11. Pour calculer le poids d'un sommet il faut connaître le poids de ses fils, ce qui suggère de procéder à un parcours en profondeur.

```
let poids f =
  let n = vect_length f in
  let w = make_vect n 1 in
  let rec aux s = function
    | []   -> ()
    | t::q -> aux t f.(t) ; w.(s) <- w.(s) + w.(t) ; aux s q
  in aux 0 f.(0) ; w ;;
```

La fonction **gauchir** consiste à placer en tête de la liste des fils celui de poids le plus élevé :

```
let gauchir f w =
  let n = vect_length f in
  let fg = make_vect n [] in
  let rec aux = function
    | [] -> []
    | a::q -> match aux q with
                | b::r when w.(a) < w.(b) -> b::(a::r)
                | r                       -> a::r
  in for i = 0 to n-1 do fg.(i) <- aux f.(i) done ;
  fg ;;
```

Question 12. Si a est un sommet léger, il possède au moins un frère a' lourd, et $w(b) \geqslant 1 + w(a) + w(a') > 2w(a)$. Notons $a_1, \ldots, a_k$ les ancêtres légers de a, rangés par hauteur croissante (a_1 est donc la racine de A et $a_k = a$). Notons b_k le père de a_k. Puisque a_{k-1} est un ancêtre de b_k nous avons $w(a_{k-1}) \geqslant w(b_k) > 2w(a_k)$, ce qui montre que $w(a_1) > 2^{k-1} w(a_k) \geqslant 2^{k-1}$. Or $w(a_1) = n$ donc $k < 1 + \log_2 n$.

Question 13. On utilise les observations suivantes :

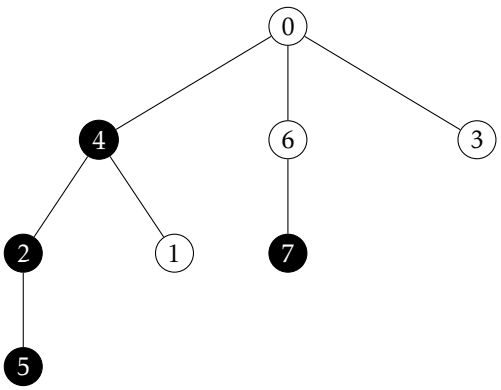
- si a est léger, ses fils ont une cime égale à a ;
- si a est lourd, ses fils légers ont une cime égale à a ;
- si a est lourd, son fils lourd a même cime que a.

On peut donc remplir le tableau des cimes à l'aide d'un parcours en profondeur de l'arbre.

```
let cime f =
  let n = vect_length f in
  let c = make_vect n 0 in
  let rec aux a pl fl = function
    | []                    -> ()
    | x::q  when pl && fl   -> c.(x) <- c.(a) ;
                               aux x true true f.(x) ; aux a true false q
    | x::q                  -> c.(x) <- a ;
                               aux x fl true f.(x) ; aux a pl false q
  in aux 0 false true f.(0) ; c ;;
```

La fonction **aux** agit sur une liste de fils dont **a** est le père ; **pl** et **fl** sont des valeurs booléennes qui traduisent si respectivement a et le nœud en tête de liste sont lourds.

Question 14. L'arbre gauche A′ est représenté ci-dessous (les nœuds noirs sont les nœuds lourds). Pour $a \in A'$, $t_2(a)$ est la distance qui sépare un sommet de son plus proche ancêtre léger.



a	0	1	2	3	4	5	6	7
$t_1[a]$	0	2	1	3	1	1	2	1
$t_2[a]$	0	0	2	0	1	3	0	1
$\lambda(a)$	$\langle\,\rangle$	$\langle 1,1,2,0\rangle$	$\langle 1,2\rangle$	$\langle 3,0\rangle$	$\langle 1,1\rangle$	$\langle 1,3\rangle$	$\langle 2,0\rangle$	$\langle 2,0,1,1\rangle$

Question 15. La démarche est voisine de celle suivie pour construire le tableau des cimes : on effectue un parcours en profondeur à l'aide d'une fonction récursive **aux**. Cette dernière agit sur une liste de fils dont **a** est le père et **t1** et **t2** les valeurs des tableaux t_1 et t_2 correspondant à la tête de cette liste.

```
let etiquettes f c =
  let n = vect_length f in
  let lambda = make_vect n [] in
  let rec aux a t1 t2 = function
    | []   -> ()
    | x::q -> lambda.(x) <- lambda.(c.(x)) @ [t1; t2] ;
              aux x 1 (t2+1) f.(x) ; aux a (t1+1) 0 q
  in aux 0 1 1 f.(0) ; lambda ;;
```

Question 16. Cette question repose sur la constatation suivante ; si a est un sommet et b la cime de a alors :

- si $t_2[a] = 0$, a est léger donc b est son père et a est le $t_1[a]$-ieme élément de la liste des fils de b ;
- si $t_2[a] > 0$, a est lourd et a est le $t_2[a]$-ieme descendant gauche de b.

Observons maintenant l'étiquette de a : $\lambda(a) = \langle x_1, x_2, \ldots, x_p\rangle$.

- Si $\lambda(a) = \langle\,\rangle$, alors $a = 0$.
- Sinon, l'étiquette $\langle x_1, \ldots, x_{p-2}\rangle$ est l'étiquette de la cime b de a et les valeurs $t_1[a] = x_{p-1}$ et $t_2[a] = x_p$ permettent à l'aide de la constatation ci-dessus de déterminer a.

Rédigeons tout d'abord deux fonctions permettant d'extraire le i^e élément d'une liste et le i^e descendant gauche d'un sommet :

```
let rec ieme_frere i = function
  | a::_ when i = 1 -> a
  | _::q            -> ieme_frere (i-1) q
  | _               -> failwith "ieme_frere" ;;

let ieme_descendant f i a =
  let rec aux i = function
    | t::q when i = 1 -> t
    | t::q            -> aux (i-1) f.(t)
    | []              -> failwith "ieme_descendant"
  in aux i f.(a) ;;
```

On peut ensuite mettre en œuvre la démarche récursive décrite ci-dessus :

```
  let trouve f =
  let rec aux b = function
    | []       -> b
    | t1::0::q -> let a = ieme_frere t1 f.(b) in aux a q
    | _::t2::q -> let a = ieme_descendant f t2 b in aux a q
    | _        -> failwith "trouve"
  in aux 0 ;;
```

Question 17.

a) Soit a un sommet, et $a_1 = a, a_2, \ldots, a_p = 0$ tel que a_{k+1} soit la cime de a_k. Parmi ces nœuds se trouvent α nœuds légers et $p - \alpha$ nœuds lourds. Or si a_k est un nœud lourd sa cime a_{k+1} est un nœud léger, donc $p - \alpha \leqslant \alpha$, soit $p \leqslant 2\alpha$.
Sachant que la longueur de $\lambda(a)$ est égale à $2p$ on a $|\lambda(a)| \leqslant 4\alpha$, et puisque chaque nœud de la liste $a_1, \ldots, a_p$ est un ancêtre de a, on a bien montré que la longueur de $\lambda(a)$ est inférieure ou égale à quatre fois le nombre d'ancêtres légers de a.

b) Considérons le plus long préfixe de longueur paire commun à u et v et posons $u = w \circ u'$, $v = w \circ v'$. Le sommet s vérifiant $\lambda(s) = w$ est un ancêtre commun à a et b.
Si $u' = \langle\,\rangle$ ou $v' = \langle\,\rangle$ c'est que $s = a$ ou $s = b$ et l'un de ces deux sommets est un ancêtre de l'autre, donc $\mathrm{ppac}(a, b) = s$.
Sinon, on peut poser $u = \langle t_1, t_2\rangle \circ u''$ et $v = \langle t'_1, t'_2\rangle \circ v''$.
Si $t_2 = t'_2 = 0$ alors $t_1 \neq t'_1$ et a et b appartiennent à la descendance de deux fils légers distincts de s. On a donc $\mathrm{ppac}(a, b) = s$.
Si $t_2 = 0$ et $t'_2 \neq 0$ ou $t'_2 = 0$ et $t_2 \neq 0$, a et b appartiennent à la descendance de deux fils distincts de s (car l'un est léger et l'autre lourd). On a donc $\mathrm{ppac}(a, b) = s$.
Si $t_2 \neq 0$ et $t'_2 \neq 0$ alors $t_1 = t'_1 = 1$: la recherche se poursuit dans le fils lourd de s. Supposons par exemple $t_2 < t'_2$. Alors le nœud s' étiqueté par $w \circ \langle 1, t_2\rangle$ est un ancêtre de celui étiqueté par $w \circ \langle 1, t'_2\rangle$ et s' est le plus petit ancêtre commun à a et b.
En conclusion, $\mathrm{ppac}(a, b) = \begin{cases} s' \text{ avec } \lambda(s') = w \circ \langle 1, t_2\rangle & \text{si } u = w \circ \langle 1, t_2\rangle \circ u'' \text{ et } v = w \circ \langle 1, t'_2\rangle \circ v'' \text{ avec } t_2 < t'_2 \\ s \text{ avec } \lambda(s) = w & \text{dans les autres cas} \end{cases}$

♦ ♦
♦