Sujet X 2010 : corrigé

Question 1

```
let remplir_taille =
   let rec aux_arbre a fils = match (a,fils) with
      | Noeud(i,[]) -> taille.(i) <- 1; 1
      | Noeud(i, a0::q) -> let s = aux_arbre a0 + aux_fils q in taille.(i) <- s; s
   and aux_fils fils = match fils with
      | [] -> 0
      | a0::q -> aux_arbre a0 + aux_fils q
   in let s=aux_arbre A in ();;
```

Question 2

```
let appartient i j = (j<=i) && (i<j+taille.(j));;
```

Question 3

```
let ppac1 i j =
   if appartient i j then j
   else begin
      let k = ref j in
      while not(appartient !k i) do
         let l= ref !k in
         while not(appartient !k !l) do
            l:=!l-1
         done;
         k:= !l
      done;
      !k end;;
```

Question 4

On montre ce résultat par récurrence (forte) sur taille[i] que le tout eulérien à partir de i contient $2 \times taille[i] + 1$ éléments.

Si i a taille 1, i n'a pas de fils, alors la seule opération à effectuer est l'ajout i à la fin de résultat, résultat n'a donc qu'une valeur, à savoir i, et $1 = 2 \times taille[i] - 1$.

Supposons que i a $n+1$, $n \geq 1$. Soient $j_1, \cdots, j_p$ les fils de i, qui ont tous taille au plus n. Par définition d'un tour eulérien, le tour eulérien est constitué des tours eulériens en j_k auxquels on ajoute l'élément i pour tout k, et, enfin, de l'élément i. Par hypothèse de récurrence, le tour eulérien à partir de j_k est constitué de $2 \times taille[j_k] - 1$ éléments donc le tour eulérien à partir de j_k auquel on ajoute i est constitué de $2 \times taille[j_k] - 1 + 1 = 2 \times taille[j_k]$ éléments. Le tour eulérien à partir de i est donc constitué de tous ces tour eulériens et de i ; il contient donc $\sum_{k=1}^{p} \big(2 \times taille[j_k]\big) + 1$ éléments, à savoir $2 \sum_{k=1}^{p} taille[j_k] + 1$. Or $taille[i] = 1 + \sum_{k=1}^{p} taille[j_k]$. Le tour eulérien à partir de i prend donc $2 \times (taille[i] - 1) + 1 = 2 \times taille[i] - 1$ éléments.

En particulier, le tour eulérien de `A` étant le tour eulérien à partir de 0, sa racine, il contient $2 \times taille[0] - 1 = 2n - 1$ éléments.

Question 5

```
let remplir_euler =
```

```
   let rec aux_arbre a k = match a with
      | Noeud( i, []) -> index.(i) <- k; euler.(k) <- i; k-1
      | Noeud( i, a0::q) ->
      index.(i) <- k; euler.(k) <- i; aux_fils q (aux_arbre a0 (k-1))
   and aux_fils l k = match l with
      | [] -> k
      | a0::q -> aux_fils q (aux_arbre a0 k)
   in let rec_aux_arbre a m in ();;
```

Question 6

Pour chaque noeud k, k est marqué une première fois puis une fois de plus à la fin de chaque parcours de fils. Il est donc parcouru $p+1$ fois où p est son nombre de fils. Notons $k_1, \cdots, k_{p+1}$ les indices de ces marquages, p étant le nombre de fils de k. Les éléments marqué entre k_i et k_{i+1} sont les noeuds du i-ème fils de k. Ce sont donc des descendants de k donc des éléments plus petits que k. Les éléments correspondant aux indices entre k_1 et k_{p+1} sont donc tous inférieurs à k.

Supposons que $k = PPAC(i,j)$ et que $k \neq i$, $k \neq j$. Alors si i dans le m-ème fils de k, toutes les occurrences de i seront comprises entre k_m et k_{m+1} ; si j est le q-ème fils de k, ses occurrences seront entre k_q et k_{q+1}. De plus, comme k est le plus petit ancêtre commun, $m \neq q$. Supposons, quitte à changer i et j, $m < q$. Alors index[i]$<k_{m+1} \leq k_q<$index[j]. Par conséquent, euler[k_{m+1}]=k est bien le plus petit élément du tableau euler entre index[i] et index[j]. PPAC(i,j) est donc bien le minimum du tableau euler entre les indices index[i] et index[j].

Lorsque $k = i$ ou $k = j$, on peut supposer $k = i$ quitte à échanger i et j, toutes les occurrences de j sont entre k_q et k_{q+1}. Ainsi, $k_q < index[i] < k_{q+1}$. On a donc $k_1 < index[i] < k_{p+1}$. Tous les éléments du tableau euler entre k_1 et k_{p+1} sont au plus égaux à $k = i$. En particulier, si $k_r = index[i]$, les éléments du tableau euler pour les indices compris entre $index[i] = k_r$ et $index[j]$ sont tous au plus égaux à k ; comme $k = euler[k_r]$, k est bien le minimum du tableau euler entre index[i] et index[j]. PPAC(i,j) est à nouveau le minimum de euler entre les indices index[i] et index[j].

Question 7

```
let rec log2 n = if n=1 then 0 else 1+log2 (n/2);;
```

Question 8

```
let remplir_M =
   for i=0 to k do
      let deuxPuissanceJMoinsUn= ref 1 in
      M.(i).(0) <- euler.(i);
      for j=1 to log2(m-i) do
         M.(i).(j) <- min (M.(i).(j-1)) (M.(i+ !deuxPuissanceJMoinsUn).(j-1));
         deuxPuissanceJMoinsUn:=2* !deuxPuissanceJMoinsUn
         done
      done
   done;;
```

Question 9

```
let rec puissance2 k = if k=0 then 1 else
   let p = puissance (k/2) in if k mod 2 =0 then p*p else 2*p*p;;
let minimum i j = let k= log2(j+1-i) in min (M.(i).(k) M.(j-1-puissance2 k).(k));;
```

Question 10

```
let ppac2 i j = minimum index.(i) index.(j);;
```

Question 11

```
let rec bit_fort n = if n=1 then 0 else bit_fort (decalage_droite n 1);;
```

Question 12

```
let t=Array.make 256 0;;
let deuxPuis= ref 1 in
```

```
for p=0 to 7 do
   for k=deuxPuis to 2*deuxPuis-1 do
      t.(k) <- p;
      deuxPuis:=2* !deuxPuis
   done
done;;

let rec bit_fort n =
   let m=decalage_droite n 15 in
   if m=0 then let p=decalage_droite n 7 in
   if p=0 then t.(n) else 7+t.(p)
   else let p=decalage_droite p 7 in
   if p=0 then 15+t.(m) else 22+t.(p);;
```

Quesiton 13

```
let remplir_B =
   let k=bit_fort (n-1) in
   let p0=decalage_gauche 1 k in
   let rec aux a m p = match a with
      | Noeud(i, []) -> ()
      | Noeud(i, [ag ;ad]) -> B.(i) <- m; B.(m) <- i;
      let p1= decalage_droite p 1 in aux ag (ou_excl_bits m (ou_excl_bits p1 p)) p1;
      aux ad (ou_excl_bits m p1) p1
    in aux a p0 p0;;
```

Question 14
x est le premier bit différent. k est donc la hauteur du PPAC de i et j. B(a) est donc la troncature à l'ordre k de $B(i)$ suivie de 1, c'est-à-dire que, si $B(i) = x_d \cdots x_k \cdots x_0$, alors $B(a) = x_d \cdots x_k 10 \cdots 0$.
Question 15

```
let ppac3 i j =
   if appartient i j then j
   else if appartient j i then i
   else let k=bit_fort (ou_excl B.(i) B.(j)) in
   ou_excl (decalage_gauche (decalage_droite B.(i) k) k) (decalage_gauche 1 (k-1));;
```

Question 16
Dans le pire des cas, c'est-à-dire celui où $m = i$ ou $m = j$, en notant C_n la complexité de l'algorithme, on a la relation de récurrence : $C_n = C_{n-1} + \Theta(n)$ car l'algorithme va s'appeler sur un tableau de taille un de moins après une recherche de minimum en $\Theta(n)$. La complexité est donc $\Theta(n^2)$.
Question 17

```
let rec construire_liste T i =
   if i=0 then [Noeud(T.(0),[])]
   else let l=construire_liste (i-1) in
   let rec insere i l acc = match l with
      | [] -> [Noeud(T.(i), acc)]
      | Noeud(u,_)::_ when T.(i)>=u -> (Noeud(T.(i), acc))::l
      | Noeud(u, l0)::q -> insere i q [Noeud(u, l0@acc)]
   in insere i l [];;

let construire_A T =
   let rec liste_vers_arbre l acc = match l with
      | [] -> let [x]=acc in x
      | Noeud(vk,l)::q -> liste_vers_arbre q [Noeud(vk, l@acc)]
   in liste_vers_arbre (construire_liste T (Array.length T -1));;
```

Question 18
Soit C_n la complexité de la fonction "liste vers arbre" pour une liste de taille n. Alors, l'opération de concaténation

de liste étant en $O(1)$ (puisque pour une liste de taille, sauf éventuellement la première), on a $C_n = C_{n-1} + O(1)$ donc ce coût est linéaire.
Notons j_i l'indice noté j dans l'énoncé après l'insertion de T[i]. La profondeur de la branche droite après insertion de T[i] est donc $j_i - 1$.
Lors de l'insertion de T[i], lorsque $j_i < j_{i-1}$ (cas noté "j<k" par l'énoncé), on insère T[i] après avoir parcouru $j_{i-1} - j_i$ noeuds de la branche droite. On a donc une complexité pour cette insertion de $O(j_{i-1} - j_i)$.
Dans l'autre cas, $j_i = j_{i-1} + 1$ et on ne parcourt que le premier noeud de la branche droite. La complexité est $O(1) = O(j_i - j_{i-1}) = O(|j_{i-1} - j_i|)$.
La complexité totale de l'insertion de chaque T[i] est donc $O(\sum_{i=1}^{n-1} |j_{i-1} - j_i|)$. Notons J_+ l'ensemble des indices i tels que $j_{i-1} > j_i$ et J_- son complémentaire. Pour $i \in J_-$, $|j_{i-1} - j_i| = j_i - j_{i-1} = 2(j_i - j_{i-1}) + (j_{i-1} - j_i)$. On a donc

$$\sum_{i=1}^{n-1} |j_{i-1} - j_i| = \sum_{j \in J_+} (j_{i-1} - j_i) + \sum_{j \in J_-} (j_i - j_{i-1}) = \sum_{j \in J_+} (j_{i-1} - j_i) + \sum_{j \in J_-} (j_{i-1} - j_i) + 2 \times \sum_{j \in J_-} (j_i - j_{i-1})$$

$$= \sum_{j=1}^{n-1} (j_{i-1} - j_i) + 2 \sum_{j \in J_-} 1 = 1 - j_{n-1} + 2|J_-|.$$

Or $j_{n-1} \geq 0$ et $|J_-| \leq n$. On a donc $\sum_{i=1}^{n-1} |j_{i-1} - j_i| = O(n)$.
En conclusion la complexité de l'insertion de tous les éléments T[i] est en $O(n)$; la complexité de "construire_liste" est donc $O(n)$.
La complexité totale de "construireA" est donc $O(n)$.