Sujet XENS (2023)

Question I.1.

```
let occurrences l = let occ=Array.make 256 0 in
    let rec aux l = match l with
        | [] -> ()
        | t::q -> occ.(t) <- occ.(t)+1; aux q
    in aux l; occ;;
```

Question I.2.

```
let nonzero_occurrences t = let i = ref 0 in
    for k=0 to Array.length t -1 do
        if t.(k)<>0 then i:= !i +1;
    done;
    let t2=Array.make !i (0,0) in
    let j= ref 0 in
    for k=0 to Array.length t -1 do
        if t.(k)<>0 then begin t.(!j) <- (k,t.(k)); j:= !j + 1 end
    done;
    t2;;
```

Question II.1. Un arbre ayant une seule feuille fixée x est unique : c'est F(x). Un arbre ayant deux feuilles $x \neq y$, peut avoir sous-arbre gauche $F(x)$ et droit $F(y)$ ou l'inverse. Il y a deux tels arbres. Un arbre ayant 3 feuilles x, y, z peut avoir un sous-arbre gauche ayant une ou deux feuilles ; si celui-ci a une feuille, il y a 3 possibilités. Pour chacune d'entre elle (disons x), il reste deux sous-arbres droits ayant feuilles les deux autres étiquettes (en l'espèce y et z). Cela nous fait $3 \times 2 = 6$ arbres de ce type possibles. Il peut aussi y avoir un sous-arbre gauche à 2 feuilles ; pour choisir ces deux feuilles il y a $\binom{3}{2} = 3$ possibilités et, pour chacune d'entre elles, 2 possibilités, le sous-arbre droit est automatiquement fixé. Il y a donc encore 6 nouveaux arbres de ce type. Au total, il y a 12 arbres à 3 feuilles fixées.
Un arbre ayant quatre feuille peut avoir un sous-arbre gauche avec 1,2 ou 3 éléments. Parmi les arbres gauches ayant un élément, il y a 4 possibilités ; un tel sous-arbre gauche fixé, il reste à fixer un sous-arbre droit à 3 feuilles ; il y a pour ceux-ci 12 possibilités. Cela nous fait $4 \times 12 = 48$ possibilités de ce type. De même, il y a 48 arbres à quatre feuilles ayant un sous-arbre gauche à 3 feuilles et sous-arbre droit à 1 feuille. Pour se donner un arbre ayant sous-arbre droit et gauche à deux feuilles chacun, il y a $\binom{4}{2}$ façons de choisir deux feuilles pour le sous-arbre gauche ; cela fait, il y a 2 sous-arbres gauches possibles ; cela fait, il y a 2 sous-arbres droits possibles. Les arbres ayant sous-arbres droit et gauche à deux feuilles sont au nombre de $\binom{4}{2} \times 2 \times 2 = 24$.
Au total, il y a donc $48 + 48 + 24 = 120$ arbres dont les feuilles sont 4 éléments distincts (comme A,B,C,D).
Question II.2.

```
let cq a q =
    let rec aux a s h = match t with
        | Feuille(x) -> s+q.(x)*h
        | N(g,d) -> aux g (aux d s (h+1)) (h+1) in aux a 0 0;;
```

La fonction "aux" sur un arbre N(g,d) effectue un appel récursif sur g et un sur d, ainsi qu'un nombre borné d'opérations de complexité bornée. La complexité $C(a)$ sur un arbre $a = N(g, d)$ de taille $|a|$ (la taille étant le nombre de noeuds internes et de feuilles) suit la relation $C(|a|) = C(|g|) + C(|d|) + O(1)$ donc, comme $|a| = |g| + |d| + 1$, $C(|a|) = O(|a|)$. Comme $|a| = |S| + |S| - 1 = O(|S|)$, la complexité est $O(|S|)$.
Question II.3.

```
let get_path i t = let tab = Array.length (Array.length t) [] in
    let rec aux t l = match t with
        | Feuille(x) -> tab.(x) <- l
        | N(g,d) -> aux g (0::l); aux d (1::l) in aux t [] in tab.(i);;
```

Comme précédemment, la complexité vérifie la relation $C(|a|) = C(|g|) + C(|d|) + O(1)$ donc la complexité est $O(|S|)$.
Question II.4.

```
let integers k =
    let rec noeud un x k =
        if un<k then Noeud(complet x un, noeud (un+1) (x+2**(un-1)))
        else Feuille(x)
    and complet x k =
        if k=0 then Feuille(x)
        else Noeud(complet x (k-1), complet (x+2**(k-1)) (k-1))
    in noeud 0 0 k;;
```

Question III.1. Montrons ce résultat par récurrence sur la longueur d'une séquence de bits.
Si cette séquence a strictement moins de bits que le code de la plus petite lettre, il n'y a pas de lettre donc pas mot pouvant être codé par cette suite. Si cette suite a autant de bits que le plus petit nombre de bits pour coder une lettre, alors cette séquence ne peut coder qu'une lettre ; cette séquence de bits correspond à un chemin dans l'arbre donc à une unique lettre.
Supposons que le résultat est vraie pour tout séquence d'au plus n bits. Considérons une séquence $s_1 \cdots s_{n+1}$ de $n+1$ bits. Soit i tel que le chemin codé par $s_1 \cdots s_i$ soit le premier à aboutir à une feuille donc à une lettre x. Comme cette lettre est une feuille, le chemin codé par $s_1 \cdots s_k$ avec $k > i$ ne reste pas dans l'arbre car la feuille correspondant à x n'a pas de fils. Tout mot codé par cette séquence commence donc par x. Comme le mot codé par $s_{i+1} \cdots s_{n+1}$ est, d'après l'hypothèse de récurrence, unique, le mot codé par $s_1 \cdots s_{n+1}$ est unique.
Ceci prouve le résultat.
Question III.2.

```
let decomp1 s t =
    let rec aux mot s noeudcourant = match s,noeudcourant with
        | [], Feuille(x) -> x::mot
        | [],_ -> failwith "code incorrect"
        | _,Feuille(x) -> aux (x::mot) s t
        | 0::s0,Noeud(g,d) -> aux mot s0 g
        | 1::s0,Noeud(g,d) -> aux mot s0 d in aux [] s t;;
```

Pour chaque bit de la séquence, un nombre borné d'opérations et un appel récursif sur une séquence de taille un de moins est effectué. La complexité est donc linéaire en la taille de la séquence de bits. C'est donc $c_q(t)$ si le mot codé est dans $\mathcal{S}^q$.
Question IV.1 Soit g' un arbre pour $(q, \mathcal{S}_g)$. Par optimalité de $t = N(g, d)$, $c_q(t) \leq c_q(t')$ où $t' = N(g', d)$. Or $c_q(t') = \sum_{\sigma\in\mathcal{S}_g} q(\sigma)l_{t'}(\sigma) + \sum_{\sigma\in\mathcal{S}_d} q(\sigma)l_{t'}(\sigma)$ et $c_q(t) = \sum_{\sigma\in\mathcal{S}_g} q(\sigma)l_t(\sigma) + \sum_{\sigma\in\mathcal{S}_d} q(\sigma)l_t(\sigma)$. Or, pour tout $\sigma \in \mathcal{S}_d$, $l_t(\sigma) = l_{t'}(\sigma)$ et, pour tout $\sigma \in \mathcal{S}_g$, $l_t(\sigma) = l_g(\sigma) + 1$ et $l_{t'}(\sigma) = l_{g'}(\sigma) + 1$ donc, comme $c_q(t) \leq c_q(t')$, $\sum_{\sigma\in\mathcal{S}_g} q(\sigma)(l_g(\sigma) + 1) \leq \sum_{\sigma\in\mathcal{S}_g} q(\sigma)(l_{g'}(\sigma) + 1)$ donc $c_q(g) = \sum_{\sigma\in\mathcal{S}_g} q(\sigma)l_g(\sigma) \leq \sum_{\sigma\in\mathcal{S}_g} q(\sigma)l_{g'}(\sigma) = c_q(g')$. Ainsi, g est optimal pour $(q, \mathcal{S}_g)$.
Question IV.2. Soit t un arbre pour $(q, \mathcal{S})$. Supposons que la feuille σ_0 soit à profondeur $k > 1$. Soit σ_1 une lettre à profondeur 1. Considérons l'arbre t' obtenu en échangeant les feuilles σ_0 et σ_1. Alors $c_q(t) - c_q(t') = q(\sigma_0)\times k + q(\sigma_1)\times 1 - q(\sigma_0) - q(\sigma_1)\times k = (k-1)(q(\sigma_0) - q(\sigma_1))$. Or $q(\sigma_0) > \sum_{\sigma\neq\sigma_0} q(\sigma) \geq q(\sigma_1)$ donc $c_q(t) > c_q(t')$ donc t n'est pas optimal. Par contraposition, si t est optimal pour $(q, \mathcal{S})$, σ_0 est à profondeur 1.
Question IV.3. Soit t' un arbre optimal pour $(q', \mathcal{S}')$ et t l'arbre obtenu en remplaçant $F(\sigma_3)$ par $N(F(\sigma_1), F(\sigma_2))$.
Soit T un arbre optimal pour $(q, \mathcal{S})$. Quitte à échanger σ_1 et σ_2, supposons que $l_t(\sigma_1) \leq l_t(\sigma_2)$; on notera $h_1 = l_T(\sigma_1)$ et $h_2 = l_T(\sigma_2)$. Supposons que σ_1 et σ_2 ne sont pas issus du même noeud. Notons a le sous-arbre voisin de $F(\sigma_2)$. Considérons T' l'arbre obtenu en échangeant $F(\sigma_2)$ et a dans T. Alors $c_q(T) - c_q(T') = \sum_{\sigma\in\mathcal{S}_a} q(\sigma)l_T(\sigma) + q(\sigma_1)h_1 - \sum_{\sigma\in\mathcal{S}_a} q(\sigma)l_{T'}(\sigma) - q(\sigma_1)h_2$ et $l_{T'}(\sigma) = l_T(\sigma) - h_2 + h_1$ donc $c_q(T) - c_q(T') = \sum_{\sigma\in\mathcal{S}_a} q(\sigma)(h_2 - h_1) - q(\sigma_1)(h_2 - h_1) = (h_2 - h_1)\big(\sum_{\sigma\in\mathcal{S}_a} q(\sigma) - q(\sigma_1)\big)$. Or $h_2 - h_1 \geq 0$ et $\sum_{\sigma\in\mathcal{S}_a} q(\sigma) - q(\sigma_1) \geq 0$. Ainsi $c_q(T) - c_q(T') \geq 0$ donc, par optimalité de T, T' est optimal. T' est un arbre optimal ayant un noeud $N(F(\sigma_1), F(\sigma_2))$ ou $N(F(\sigma_2), F(\sigma_1))$. L'échange de $N(F(\sigma_2), F(\sigma_1))$ en $N(F(\sigma_1), F(\sigma_2))$ ne modifiant pas c_q, on peut obtenir un arbre optimal ayant un noeud $N(F(\sigma_1), F(\sigma_2))$.
Considérons désormais un arbre t' de $(q', \mathcal{S}')$ optimal. Considérons T' celui obtenu en changeant $F(\sigma_3)$ en $N(F(\sigma_1), F(\sigma_2))$ dans t'. Soit T_0 un autre arbre de $(q, \mathcal{S})$. D'après le raisonnement précédent, on peut obtenir à partir de T_0 en échangeant l'arbre voisin de $F(\sigma_1)$ et $F(\sigma_2)$ un arbre T_0' tel que $c_q(T_0') \leq c_q(T_0)$ qui possède un noeud $N(F(\sigma_1), F(\sigma_2))$. Soit t_0' l'arbre obtenu en remplaçant dans T_0' $N(F(\sigma_1), F(\sigma_2))$ par $F(\sigma_3)$. Pour $\sigma \neq \sigma_1, \sigma_2$, $l_{t_0'}(\sigma) = l_{T_0'}(\sigma)$ et $q(\sigma) = q'(\sigma)$, $l_{T_0'}(\sigma_1) = l_{T_0'}(\sigma_2) = 1 + l_{t_0'}(\sigma_3)$. On a donc $c_q(T_0') = c_{q'}(t_0') + (1 + l_{t_0'}(\sigma_3))q(\sigma_1) + (1 + l_{t_0'}(\sigma_3))q(\sigma_2) = c_{q'}(t_0') + (1 + l_{t_0'}(\sigma_3))q'(\sigma_3) = c_{q'}(t_0') + q'(\sigma_3)$. Pour les mêmes raisons, $c_q(T') = c_{q'}(t') + q'(\sigma_3)$. Par optimalité de t', $c_{q'}(t') \leq c_{q'}(t_0)$ donc $c_q(T') = c_{q'}(t') + q'(\sigma_3) \leq c_{q'}(t_0) + q'(\sigma_3) = c_q(T_0') \leq c_q(T_0)$. Ceci valant pour tout T_0, T' est optimal pour $(q, \mathcal{S})$.
Question IV.4.

```
let rec insert (q,a) l = match l with
    | [] -> [(q,a)]
    | (q0,a0)::l0 when q<=q0 -> (q,a)::l
```

```
    | (q0,a0)::l0 -> (q0,a0)::(insert (q,a) l0);;

let optimal l =
    let rec initiale l = match l with
        | [] -> []
        | (s,q)::l0 -> insert (q,Feuille(s)) (initial l0)
    and aux l = match l with
        | [(q,a)] -> a
        | (q1,a1)::(q2,a2)::l0 -> aux (insert (q1+q2,Noeud(a1,a2)) l0)
    in aux (initial l);;
```

L'insertion dans une liste croissante de taille n est en $O(n)$. La fonction auxiliaire initiale a une complexité $O(n^2)$ pour une liste de taille n. Pour la liste initiale des $(\sigma, q(\sigma))$, elle aura complexité $|\mathcal{S}|^2$. L'appel à la fonction "aux" sur une liste de taille n appelle la fonction "insert" sur une liste de taille $n-1$, ce qui a une complexité $O(n-1) = O(n)$, et effectue un appel récursif sur une taille de taille $n-1$. La complexité C_n de la fonction "aux" sur une liste de taille n vérifie donc la relation $C_n = C_{n-1} + O(n)$ donc $C_n = O(n^2)$. Par conséquent, la complexité temporelle de "optimal" est $O(|\mathcal{S}|^2)$.

Question IV.5.

Supposons que $t = N(t_1, t_2)$ et $t' = N(1', t'_2)$ sont deux arbres tels que t' est l'arbre inséré dans E après l'insertion de t dans E. Si t'_1, t'_2 étaient des arbres déjà présents dans la liste croissante par valeurs de $\overline{q}$ des arbres de E, comme t_1, t_2 précédaient t'_1, t'_2, $\overline{q}(t_1) \leq \overline{q}(t_2) \leq \overline{q}(t'_1) \leq \overline{q}(t'_2)$ donc $\overline{q}(t) = \overline{q}(t_1) + \overline{q}(t_2) \leq \overline{q}(t'_1) + \overline{q}(t'_2) = \overline{q}(t')$. Sinon, t'_1 ou t'_2 est l'arbre que l'on vient d'insérer, à savoir $t = N(t_1, t_2)$. Dans ce cas, $t'_1 = t$ ou $t'_2 = t$; par positivité de $\overline{q}$, $\overline{q}(t') = \overline{q}(t'_1) + \overline{q}(t_2) \geq \overline{q}(t)$. Ainsi, les arbres t sont insérés par valeurs croissantes de $\overline{q}$ dans E.

Question IV.6. Il suffit de créer une file f_arbres dont les éléments seront les nouveaux arbres à ajouter dans E (et donc de ne pas insérer ces nouveaux arbres parmi la liste $l\_feuilles$ des feuilles initiales de E). Tant que f_arbres contient au moins deux éléments ou que $l\_feuilles$ contient au moins un élément, on choisit, entre la tête de $l\_feuilles$ et celle de f_arbres, l'arbre de plus petite valeur de $\overline{q}$ que l'on retire de sa structure puis l'on choisit l'élément suivant parmi les nouvelles têtes de f_arbres et $l\_feuilles$ de plus petite valeur de $\overline{q}$; ces deux valeurs t_1, t_2 sont fusionnés en un arbre $N(t_1, t_2)$ qui est ensuite inséré à la fin de la file f_arbres.

L'insertion se fait alors en $O(1)$. La complexité de la fonction "aux" de la fonction "optimal" vérifie cette fois la relation $C_n = C_{n-1} + O(1)$ donc $C_n = O(n)$. La complexité de la fonction "optimal" devient linéaire en $|\mathcal{S}|$.

Question V.1.

```
let canonical t1 t2 =
    let rec aux1 liste i = match liste with
        | [] -> ([],i)
        | [a] -> ([Noeud(Feuille(t2.(i)),a)],i-1)
        | a1 :: a2 :: l0 -> let l1,j= aux1 l0 i in (Noeud(a1,a2) :: l1,j)
    and aux2 liste k i =
        if k=0 then (liste,i)
        else aux2 (Feuille(t2.(i)) :: liste) (k-1) (i-1)
    and aux3 h =
        if h=Array.length t1 -1 then aux2 [] t1.(h) (Array.length t2 -1)
        else let (l0,i0)=aux3 (h+1) in let (l1,i1)=aux1 l0 i0 in aux2 l1 t1.(h) i1
     in match aux3 (Array.length t1 -1) with t::q,i -> t;;
```

Pour chaque noeud de l'arbre, il y a un nombre borné d'opérations effectué donc la complexité est $O(n)$ où $n = 2|\mathcal{S}| + 1$ est la taille de l'arbre. La complexité est donc $O(|\mathcal{S}|)$.

Question VI.1. Si $t = N(g, d)$, $c_q(t) = c_q(t) + c_q(d) + \sum_{\sigma \in \mathcal{S}} q(\sigma)$ donc $m_{i,j} = \min_{0 \leq k \leq j-1} (m_{i,k} + m_{k+1,j}) + \sum_{k=i}^{j} q(k)$.

```
let alpha_optimal q = let n=Array.length q in let a = Array.make_matrix n n Feuille(-1) in
    for i=0 to n-1 do
        a.(i).(i) <- (q.(i),Feuille(i))
    done;
    for i=0 to n-1 do
        for j=i+1 to n-1 do
            let m=ref 0 and s=ref 0 in
            for k=i to j-1 do
                if fst(a.(i).(k))+fst(a.(k+1).(j))<fst.(a.(i).(!m)).fst(a.( !m +1).(j))
                then m:=k;
                s:= !s + q.(k)
            done;
            s:= !s + q.(j);
            a.(i).(j)<-(fst.(i).(!m)+fst.(!m+1).(j)+ !s , Noeud(snd(a.(i).(!m)),a.(!m +1).(j)))
```

```
        done
    done;
    a.(0).(n-1);;
```

Question VI.2.

```
let alpha b =
    let rec arbre1 l x = match l with
        | [] -> failwith "erreur"
        | 1::q -> (Feuille(x),q, x+1)
        | 0::q -> let (a1,l0, j)=arbre1 q x in let (a2,l1,k)=arbre1 l0 j in (Noeud(a1,a2),l1,k)
    in let (a,q,x)=arbre1 b 0 in if q=[] then a else failwith "erreur";;
```

Question VII.1.
1. Il en faut au moins $c_q(t) = 15 + 4 \times 2 + 1 \times 2 = 25$.
2. Il y en a $\binom{20}{15}\binom{5}{4}\binom{1}{1} = \frac{20\times19\times18\times17\times16}{4\times3\times2} = 5 \times 19 \times 3 \times 17 \times 2^4$.
3. $15 \times 17 \leq 2^8$, $19 \leq 2^5$ donc le nombre de mots de $\mathcal{S}^q$ est majoré par $2^{5+8+4} = 2^{17}$.
Grâce à 17 bits, on peut coder 2^{17} listes différentes donc 2^{17} nombres différents.

Question IIV.2

```
let decomp2 q y k =
    let n=ref 0 in
    for s=0 to Array.length q -1 do
        n:= !n + q.(s)
    done;
    let inverse z = let x1=z/n and x2=ref z mod n and i=ref 0 in
        while (!x2 - q.(!i) )>=0 do
            x2:= !x2 -q.(!i)
        done;
        (x1*q.(!i)+ !x2,!i) in
    let s=Array.make k -1 and z=ref y in
    for i=0 to k-1 do
        let (a,b)=inverse y in
        z:=a;
        s.(k-1-i) <- b
    done;
    (!z , s);;
```

Question VII.3. On doit avoir, pour tout i, $x_i < 2^B$. Pour avoir k_i le plus petit possible, on cherchera également à avoir $2^{B-1} \leq x_i$, pour tout i.
Choisissons $k_i = \lfloor \log_2(\frac{x_i N}{\sigma_i 2^B}) \rfloor + 1$. Alors $2^{k_i-1} \leq \frac{x_i N}{\sigma_i 2^B} < 2^{k_i}$ donc $\frac{2^{B-1}}{N}\sigma_i \leq \frac{x_i}{2^{k_i}} < \frac{2^B}{N}\sigma_i$ donc $\frac{2^{B-1}}{N}\sigma_i \leq y_i < \frac{2^B}{N}\sigma_i$ donc $\frac{2^{B-1}}{N} \leq \frac{y_i}{\sigma_i} \leq \frac{2^B}{N}$ donc $\frac{2^{B-1}}{N} \leq \lfloor \frac{y_i}{\sigma_i} \rfloor \leq \frac{2^B}{N} - 1$ donc $2^{B-1} \leq \lfloor \frac{y_i}{\sigma_i} \rfloor N \leq 2^B - N$. Comme $\lfloor \frac{y_i}{\sigma_i} \rfloor N \leq x_{i+1} \leq \lfloor \frac{y_i}{\sigma_i} \rfloor N + N - 1$, $2^{B-1} \leq x_{i+1} \leq 2^B - 1 < 2^B$.
Réciproquement, connaissant y_i, on posera $l_i = \lceil \log_2(\frac{2^B}{y_i}) \rceil + 1$. Alors $2^{B-1} \leq y_i 2^{l_i} < 2^B$ donc $2^{B-l_i-1} \leq y_i < 2^{B-l_i}$ donc $2^{B-l_i-1} \leq \frac{x_i}{2^{k_i}} < 2^{B-l_i}$ donc $2^{k_i-l_i-1} \leq \frac{x_i}{2^B} < 2^{k_i-l_i}$. Comme $\frac{1}{2} \leq \frac{x_i}{2^B} < 1$, $k_i = l_i$. Ainsi, le calcul de l_i permet bien de retrouver la valeur k_i choisie.