Corrigé T.D.3.

Exercice 1

Question 1 Plaçons-nous en un noeud x de l'arbre ; s'il est inférieur au seuil, il n'est pas modifié. Les noeuds de son sous-arbre gauche ne sont pas modifiés : ils étaient majorés par x, ils le restent. Les noeuds de son sous-arbre droit sont éventuellement augmentés de 1 ; ils majoraient x, ils le majorent toujours. Supposons maintenant x au moins égal au seuil : il est donc incrémenté. Les noeuds de son sous-arbre gauche sont ou bien inchangés ou bien augmentés de 1 : comme ils étaient majorés par x, ils sont dans tous les cas majorés par $x+1$. Les noeuds de son sous-arbre droit le majoraient, donc étaient supérieurs au seuil, donc ont été incrémentés ; ils majorent donc $x+1$. Conclusion : $\mathcal{T}_s(a)$ possède encore la propriété des arbres binaires de recherche.

Question 2 Une seule finesse : si le noeud considéré est inférieur ou égal à s, il n'est pas nécessaire de visiter son sous-arbre gauche.

```
let rec compte s = function
 | F                 -> 0
 | N(g,x,d) when x<s -> compte s d
 | N(g,x,d) when x=s -> (compte s d) + 1
 | N(g,_,d)          -> (compte s g) + 1 + (compte s d);;
```

Question 3 La remarque faite à la question précédente s'applique encore.

```
let rec incr s = function
 | F                 -> F
 | N(g,x,d) when x<s -> N(g,x,incr s d)
 | N(g,x,d) when x=s -> N(g,x+1,incr s d)
 | N(g,x,d)          -> N(incr s g,x+1,incr s d) ;;
```

Exercice 2

Question 1 Raisonnons par récurrence ; notons $\mathcal{A}(n)$ l'assertion suivante :

" si $p(a) \geq n$, alors a contient au moins 2^n feuilles "

$\mathcal{A}(0)$ est vraie, car un arbre binaire contient au moins une feuille, or $2^0 = 1$. Supposons $\mathcal{A}(n)$ acquise et considérons un arbre binaire a vérifiant $p(a) \geq n+1$. Notons g et d ses sous-arbres gauche et droit ; alors $p(g) \geq n$ et $p(d) \geq n$; en effet, la profondeur d'une feuille dans g ou d est sa profondeur dans a (laquelle vaut au moins $n+1$) diminuée de 1. De par l'hypothèse de récurrence, chacun des deux arbres g et d contient au moins 2^n feuilles ; donc a contient au moins $2^n + 2^n$, soit 2^{n+1} feuilles. Ceci établit $\mathcal{A}(n+1)$; par récurrence, l'assertion est vraie pour tout naturel n.

Question 2
```
let rec etiquette_maximale = function
 | F x    -> x
 | N(g,d) -> max (etiquette_maximale g) (etiquette_maximale d) ;;
```

Question 3
```
let rec profondeur_minimale = function
 | F _    -> 0
 | N(g,d) -> 1 + min (profondeur_minimale g) (profondeur_minimale d) ;;
```

Exercice 3

1.
```
let distance i j = let d = ref 0. in
                      for k = 0 to q - 1 do
                        d := !d +. ponderation.(k) *. abs_float (fromages.(i).(k) -
. fromages.(j).(k))
                      done;
                      !d;;
```

2.
```
let dist = let nbfro = Array.length fromages in
               let d = Array.make_matrix nbfro  nbfro 0.
               in
               for i = 0 to nbfro - 1 do
                for j = i + 1 to nbfro - 1 do
                  let v = distance i j in
                  d.(i).(j) <- v;
                  d.(j).(i) <- v
                done
               done;
               d;;
```

3.
```
let rec dist_arbr a1 a2 = match a1, a2 with
| Fromage(i), Fromage(j) -> dist.(i).(j)
| Fromage(_), N(g2, d2, _) -> min (dist_arbr a1 g2) (dist_arbr a1 d2)
| N(g1, d1, _), _ -> min (dist_arbr g1 a2) (dist_arbr d1 a2);;
```

4. On écrit tout d'abord un certain nombre de fonctions auxiliaires :
(** distance minimale de l'arbre h aux éléments de la forêt l **)

```
let rec min_dist h l = match l with
  | [x] -> (h, x, dist_arbr h x)
  | h2::t2 -> let  a1, a2, d = min_dist h t2 in
              let dhh2 = dist_arbr h h2 in
              if dhh2 <= d then
                h, h2, dhh2
              else
                a1, a2, d;;
```

(** distance minimale entre éléments de la forêt l **)

```
let rec min_dist2 l = match l with
  | [h1; h2] -> (h1, h2, dist_arbr h1 h2)
  | h1::h2::t -> let a1, a2, d  = min_dist2 (h2::t) in
                 let a1', a2', d' = min_dist h1 (h2::t) in
                 if d' <= d then
                   a1', a2', d'
                 else
                   a1, a2, d;;
```

(** création de la foret initiale **)

```
let genliste () =
  let rec loop k =
```

```
    if k = -1 then
      []
    else
      Fromage(k) :: loop (k-1)
  in
  loop (Array.length fromages - 1);;
```

(** fonction retirant deux éléments a1 a2 de la liste l **)

```
let rec enlever_deux l a1 a2 = match l with
  | [] -> []
  | h::t -> if h = a1 || h = a2 then
              enlever_deux t a1 a2
            else
              h::(enlever_deux t a1 a2);;
```

Voici la fonction demandée

```
let dendrogramme () =
  let foret_init = genliste () in
  let rec loop foret = match foret with
    | [a] -> a
    | h1::h2::t -> let a1, a2, d = min_dist2 foret in
                   let fusion = N(a1, a2, d) in
                   loop (fusion::enlever_deux foret a1 a2)
  in
  loop foret_init;;
```