

Mardi 25 Juin 2002
Durée 2h – Documents autorisés

2nd session

Deux points sont réservés à la présentation (commentaires, etc).
Les fonctions que vous proposez doivent être accompagnées de leur profil.

Exercice

Cet exercice repose sur l'algorithme de parcours en *largeur* d'un arbre binaire.

```
type 'a ab = V | Nd of 'a ab * 'a * 'a ab;;
```

```
(* type 'a ab = V | Nd of 'a ab * 'a * 'a ab *)
```

```
let a =  
  Nd  
  (Nd (Nd (Nd (V, 6, V), 3, Nd (V, 7, V)), 8,  
      Nd (Nd (V, 9, V), 5, Nd (V, 10, V))),  
   4, Nd (Nd (V, 1, V), 2, Nd (Nd (V, 20, V), 0, Nd (V, 13, V))));;
```

```
let b =  
  Nd  
  (Nd (Nd (Nd (V, 6, V), 3, Nd (V, 6, V)), 8,  
      Nd (Nd (V, 6, V), 3, Nd (V, 6, V))),  
   4, Nd (Nd (V, 3, V), 8, Nd (Nd (V, 6, V), 3, Nd (V, 6, V))));;
```

Q1. Ecrire une fonction `liste_niveau(A,n)` qui renvoie la liste des éléments des noeuds du niveau n de l'arbre binaire A .

```
let rec liste_niveau a n =  
  match (a,n) with  
  | (V, _) -> []  
  | (Nd(g,r,d), 0) -> [r]  
  | (Nd(g,_,d), x) -> (liste_niveau g (x-1)) @ (liste_niveau d (x-1));;
```

```
(*  
# liste_niveau a 2;;  
- : int list = [3; 5; 1; 0]  
# liste_niveau a 0;;  
- : int list = [4]  
# liste_niveau a 1;;  
- : int list = [8; 2]  
# liste_niveau a 3;;  
- : int list = [6; 7; 9; 10; 20; 13]  
*)
```

Q2. Ecrire une fonction `largeur(A)` qui renvoie une liste dont les éléments sont le noeuds de l'arbre binaire A parcouru en largeur.

```

let max (x,y) = if x<y then y else x;;

let rec hauteur a =
  match a with
  | V -> 0
  | Nd(g, _, d) -> 1 + max(hauteur g, hauteur d);;

let rec larg a n =
  match n with
  | -1 -> []
  | x -> (larg a (x-1)) @ (liste_niveau a x);;

let largeur a =
  larg a (hauteur a);;

(*
# largeur a;;
- : int list = [4; 8; 2; 3; 5; 1; 0; 6; 7; 9; 10; 20; 13]
*)

```

Un arbre binaire est dit nivelé si et seulement si tous les éléments de même niveau sont égaux.

Q3. Tester si un arbre binaire est nivelé.

```

let tous_egaux l =
  match l with
  | [] -> true
  | x::l' -> List.fold_right (fun a b -> a==x && b) l' true;;

let rec niveaux a n =
  match n with
  | -1 -> []
  | x -> (liste_niveau a x) :: (niveaux a (x-1));;

let arbre_nivele a =
  List.fold_right (fun x y -> (tous_egaux x) && y) (niveaux a (hauteur a)) true;;

```

Q4. Construire un arbre binaire nivelé de hauteur H où l'information des noeuds d'un niveau N est l'entier N lui même.

```

let rec cons_arbre_niv h n =
  match h with
  | 0 -> Nd(V, n, V)
  | x -> Nd(cons_arbre_niv (x-1) (n+1), n, cons_arbre_niv (x-1) (n+1));;

let cons_arbre_nivele h = cons_arbre_niv h 0;;

(*
# cons_arbre_nivele 3;;
- : int ab =
Nd
(Nd (Nd (Nd (V, 3, V), 2, Nd (V, 3, V)), 1,
Nd (Nd (V, 3, V), 2, Nd (V, 3, V))),
0,
Nd (Nd (Nd (V, 3, V), 2, Nd (V, 3, V)), 1,
Nd (Nd (V, 3, V), 2, Nd (V, 3, V))))
*)

```