TP 5 Programmation fonctionnelle

Exercice 1. Écrire la fonction qui teste si un arbre binaire est un un arbre de recherche binaire.

Solution.

```
type t = Node of t * int * t | Nil

let rec for_all p = function
  | Nil -> true
  | Node(l,x,r) -> p x && for_all p l && for_all p r

let rec is_search_tree = function
  | Nil -> true
  | Node(l,x,r) ->
      for_all (fun y -> y <= x) l && for_all (fun y -> y >= x) r &&
      is_search_tree l && is_search_tree r
```

□

Exercice 2. Écrire la fonction qui, étant donnée 2 valeurs, détermine l'ancêtre commun le plus petit dans un arbre de recherche binaire. On suppose que les 2 valeurs apparaissent dans l'arbre.

Solution.

```
let rec path_to x = function
  | Nil -> []
  | Node(_,y,_) when y = x -> []
  | Node(l,y,r) -> if y >= x then y :: path_to x l else y :: path_to x r

let rec common_prefix l1 l2 =
  match l1, l2 with
  | x::xs, y::ys when x = y -> x :: common_prefix xs ys
  | _ -> []

let find_min x y t =
  let px = path_to x t in
  let py = path_to y t in
  let prefix = common_prefix px py in
  List.fold_left min max_int prefix
```

□

Exercice 3. Écrire la fonction qui détermine le k-eme plus grand élément dans un arbre de recherche binaire.

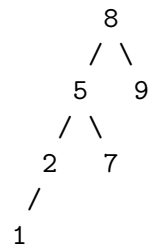
Solution.

```
let rec revvisit = function
 | Nil -> []
 | Node(l,x,r) -> revvisit r @ [x] @ revvisit l

 let rec nth n l =
   match l with
   | x :: xs -> if n = 0 then Some x else nth (n-1) xs
   | [] -> None

 let k_larger k t =
   nth k (revvisit t)
```

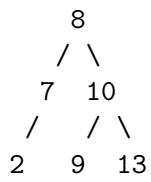
□

Exercice 4. Étant donné un arbre de recherche binaire contenant des valeurs entières positives supérieures à 0, vérifier si l'arbre de recherche binaire contient ou non une impasse. Une impasse est un nœud après que nous ne pouvons insérer aucun élément. Suggestion : vérifier s'il existe un nœud feuille avec une valeur x tel que x+1 et x-1 existent dans l'arbre de recherche binaire. Attention, pour x = 1, nous ne pouvons pas insérer 0 car l'arbre ne contient que des entiers positifs.

Exemples :



L'arbre contient un impasse car après le noeud 1 nous ne pouvons plus insérer d'élément.



L'arbre contient un impasse car après le noeud 9 nous ne pouvons plus insérer d'élément.

Solution.

```
let rec invisit = function
 | Nil -> []
 | Node(l,x,r) -> invisit l @ [x] @ invisit r
```

```
let rec mem n = function
  | [] -> false
  | x :: xs -> x = n || mem n xs

let rec exists p = function
  | [] -> false
  | x :: xs -> p x || exists p xs

let impasse t =
  let all_nodes = invisit t in
  exists (fun x ->
    let p = path_to x t in
    x = 1 || (mem (x+1) p && mem (x-1) p)) all_nodes
```

□

Exercice 5. Étant donné deux arbres de recherche binaires, recherchez les nœuds communs entre eux.

Solution.

```
let rec scan_sorted l1 l2 =
  match l1, l2 with
  | [], _ -> []
  | _, [] -> []
  | x::xs, y::ys when x = y -> x :: scan_sorted xs ys
  | x::xs, y::ys when x < y -> scan_sorted xs (y::ys)
  | x::xs, y::ys -> scan_sorted (x::xs) ys

let common t1 t2 =
  let l1 = invisit t1 in
  let l2 = invisit t2 in
  scan_sorted l1 l2
```

□