Algorithmique des structures de données arborescentes
Feuille d'exercices 2

1 Induction structurelle

Reprendre les exercices 1.4 et 1.5 de la feuille 1.

2 Dénombrement des arbres binaires

Exercice 2.1 Le nombre de squelette d'arbres binaires de taille n est $\frac{1}{n+1} \cdot \binom{2n}{n}$

1. Énumerer les squelettes d'arbres binaires de taille $1, 2, 3$
2. Quelle est la hauteur minimale d'un arbre binaire de taille 4 ? Donner un exemple.
3. Quelle est la hauteur maximale d'un arbre binaire de taille 4 ? Donner un exemple.

Exercice 2.2

1. Énumerer les squelettes d'arbres binaires complets de taille $1, 3, 5$.
2. Quelle est la hauteur minimale d'un arbre binaire complet de taille 7 ? Donner un exemple.
3. Quelle est la hauteur maximale d'un arbre binaire complet de taille 7 ? Donner un exemple.
4. Justifier qu'il y a autant de squelettes d'arbres binaires de taille n que de squelettes d'arbres binaires complets de taille $2n + 1$.

Exercice 2.3 Soit `t` un arbre binaire non vide de hauteur `h`. Soient `g` son sous-arbre gauche et `d` sont sous-arbre droit.

1. Énoncer les conditions nécessaires et suffisantes sur les sous-arbres `g` et `d` pour que l'arbre `t` soit un arbre parfait.
2. Combien de squelettes d'arbres parfaits de hauteur `h` et d'arbres quasi-parfaits de hauteur `h` existe-t-il ?
3. Énoncer les conditions nécessaires et suffisantes sur les sous-arbres `g` et `d` pour que l'arbre `t` soit un arbre quasi-parfait.

3 Fonctions récursives et arbres binaires en Ocaml

Une ***liste*** `l` est

- soit la liste vide, notée `[]` en OCaml,
- soit constituée d'un premier élément `x` suivi du reste de la liste. Cela s'écrit `x::tail` en OCaml,

On peut donc écrire des fonctions récursives sur les listes. Par exemple la fonction qui renvoie la longueur d'une liste s'écrit de la façon suivante :

```
let rec list_length l =
  match l with
  | [] -> 0
  | x::tail -> 1 + list_length tail
```

Cette fonction est de type `'a list -> int` où `'a` est une variable de type qui représente un type quelconque, donc elle peut prendre en paramètre une liste d'entiers, une liste de réels, une liste de charactères, ... (mais tous les éléments doivent être du même type).

Exercice 2.4 Écrire une fonction `list_nb_occ` qui prend en paramètre un élément `e` et une liste `l` et qui renvoie le nombre d'occurrences de cet élément dans la liste.
Par exemple l'appel `list_nb_occ 2 [5;2;4;2;3;7;2]` doit renvoyer 3.
Attention : les éléments de la liste sont séparés par des points-virgule.

Pour l'implémentation en OCaml des arbres binaires, on utilisera le type récursif suivant :

```
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
```

Voici un exemple d'arbre binaire de type `int btree` :

```
let t = Node(7, Node(2, Node(5,Empty,Empty), Empty),
                Node(1, Node(5,Empty,Empty), Node(3,Empty,Empty)))
```

On utilisera le filtrage pour écrire des fonctions sur les arbres binaires. Par exemple la fonction qui renvoie l'étiquette de la racine de l'arbre s'écrit :

```
let btree_root t =
  match t with
| Empty -> failwith "l'arbre est vide"
| Node(x,l,r) -> x
```

Exercice 2.5 Écrire une fonction `btree_are_equal` de type `'a btree -> 'a btree -> bool` qui teste si deux arbres sont égaux.
Note. L'opérateur `=` d'OCaml répond à la question, mais on demande de le ré-écrire.

Exercice 2.6 Écrire une fonction `btree_mem` de type `'a -> 'a btree -> bool` qui teste si une valeur apparaît comme étiquette dans un arbre.

Exercice 2.7 Écrire une fonction `btree_at_depth` de type `'a btree -> int -> 'a list` qui renvoie la liste des étiquettes des nœuds se trouvant à profondeur donnée dans un arbre, lus de gauche à droite.

Exercice 2.8 Écrire une fonction `btree_is_full` de type `'a btree -> bool` testant si un arbre binaire est complet.

Exercice 2.9 Arbre binaire parfait.

1. Écrire une fonction `btree_is_perfect` de type `'a btree -> bool` testant si un arbre est parfait.
2. Évaluer la complexité de la fonction écrite.
3. Si la fonction n'est pas de complexité linéaire, en écrire une version linéaire.

Exercice 2.10 Écrire une fonction `btree_is_quasi_perfect` de type `'a btree -> bool` testant si un arbre binaire est quasi-parfait.