Programmation fonctionnelle

Fiche de TP 4

L3 Informatique 2021-2022

Arbres binaires

Rappel : un arbre binaire est un objet récursif défini comme étant

— soit une feuille ;

— soit un nœud interne portant un élément et à partir duquel sont attachés deux arbres binaires.

De manière équivalente, un *arbre binaire* est une structure de données qui peut se représenter sous la forme d'une hiérarchie dont chaque élément est appelé *nœud*. Dans un arbre binaire, chaque élément possède au plus deux éléments fils au niveau supérieur, habituellement appelés *gauche* et *droit*. Du point de vue de ces éléments fils, l'élément dont ils sont issus au niveau inférieur est appelé *père*. Au niveau le plus bas il y a un nœud unique, appelé *racine*. Au niveau directement supérieur, il y a au plus deux nœuds fils. En continuant à explorer les niveaux supérieurs, on peut avoir quatre, puis huit, seize, trente-deux, *etc.* nœuds par niveau. Un nœud n'ayant aucun fils est appelé *feuille*, sinon il est appelé *nœud interne*. La *profondeur* d'un nœud est la distance de ce nœud par rapport à la racine. Ainsi, la profondeur de la racine est 0, celle de ses éventuels fils est 1, *etc.* La *hauteur* d'un arbre binaire est la plus grande profondeur parmi tous ses nœuds. Finalement, chaque nœud interne d'un arbre binaire peut porter un élément ; on parlera dans ce cas d'*arbre binaire sur les entiers* pour désigner un arbre binaire dont chaque nœud interne porte un entier.

Exercice 1. (Type arbre binaire)

Écrire le type `bintree` permettant de représenter les arbres binaires définis sur les entiers. Se baser pour cela sur la définition récursive des arbres binaires, rappelée tout au début de la fiche.

Utiliser ce type pour construire l'arbre binaire défini en figure 1. Lier le nom `example_tree` à cet arbre binaire.

Tous les exemples de la suite porteront sur l'arbre `example_tree` (sauf les fonctions qui portent sur les arbres binaires de recherche qui elles travaillent sur les arbres binaires dont l'étiquetage respecte la condition arbre binaire de recherche).

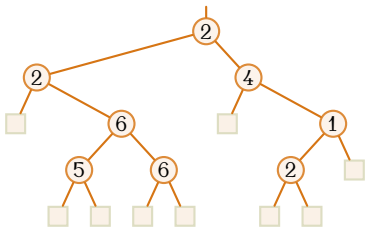

FIGURE 1 – Un arbre binaire sur les entiers. Les nœuds internes sont représentés par des cercles et les feuilles par des carrés. Chaque nœud interne porte comme élément l'entier qui y figure à l'intérieur.

Exercice 2. (Compter)

1. Écrire la fonction `bintree_count_leaves` qui renvoie le nombre de feuilles dans un arbre binaire en paramètre.
2. Écrire la fonction `bintree_count_internal_nodes` qui renvoie le nombre de nœuds internes dans un arbre binaire en paramètre.
3. Écrire la fonction `bintree_count_nodes` qui renvoie le nombre de nœuds dans un arbre binaire en paramètre.
4. Écrire la fonction `bintree_count_right` qui renvoie le nombre d'arêtes orientées vers la droite connectant deux nœuds internes dans un arbre binaire en paramètre.
5. Écrire la fonction `bintree_count_left` qui renvoie le nombre d'arêtes orientées vers la gauche connectant deux nœuds internes dans un arbre binaire en paramètre.

```
# bintree_count_leaves;;
- : bintree -> int = <fun>
# bintree_count_leaves example_tree;;
- : int = 9

# bintree_count_internal_nodes;;
- : bintree -> int = <fun>
# bintree_count_internal_nodes example_tree;;
- : int = 8

# bintree_count_nodes;;
- : bintree -> int = <fun>
# bintree_count_nodes example_tree;;
- : int = 17

# bintree_count_right;;
- : bintree -> int = <fun>
# bintree_count_right example_tree;;
- : int = 4

# bintree_count_left;;
- : bintree -> int = <fun>
# bintree_count_left example_tree;;
- : int = 3
```

Exercice 3. (Propriétés)

1. Écrire la fonction `bintree_height` qui renvoie la hauteur d'un arbre binaire en paramètre.
2. Écrire la fonction `bintree_is_mirror` qui teste si un arbre binaire en paramètre est l'image miroir d'un autre arbre binaire en paramètre. Nous nous intéressons ici à la structure des deux arbres binaires et pas aux valeurs portées par leurs nœuds.

3. Un arbre binaire est *symétrique* s'il est possible de tracer une ligne verticale passant par la racine telle que le sous-arbre droit est l'image miroir du sous-arbre gauche. En d'autres termes, `t` est symétrique si `t` est sa propre image miroir.

   Écrire la fonction `bintree_is_symmetric` qui teste si un arbre binaire en paramètre est symétrique.

```
# bintree_height;;
- : bintree -> int = <fun>
# bintree_height example_tree;;
- : int = 4

# bintree_is_mirror;;
- : bintree -> bintree -> bool = <fun>
# bintree_is_mirror Leaf Leaf;;
- : bool = true

# bintree_is_mirror (Node(1, Leaf, Leaf)) (Node(2, Leaf, Leaf));;
- : bool = true
# bintree_is_mirror (Node(1, Leaf, Node(3, Leaf, Leaf))) (Node(2, Node(4, Leaf, Leaf), Leaf));;
- : bool = true

# bintree_is_symmetric;;
- : bintree -> bool = <fun>
# bintree_is_symmetric Leaf;;
- : bool = true
# bintree_is_symmetric (Node(1, Leaf, Leaf));;
- : bool = true
# bintree_is_symmetric (Node(1, Node(2, Leaf, Leaf), Node(2, Leaf, Leaf)));;
- : bool = true
# bintree_is_symmetric (Node(1, Node(2, Leaf, Leaf), Node(3, Leaf, Leaf)));;
- : bool = true
# bintree_is_symmetric (Node(1, Node(2, Leaf, Leaf), Leaf));;
- : bool = false
```

Exercice 4. (Collecter)

1. Écrire la fonction `bintree_collect_values` qui renvoie la liste des entiers portés par les nœuds internes dans un arbre binaire en paramètre, dans un ordre quelconque.
2. Écrire la fonction `bintree_collect_level` qui renvoie la liste des entiers portés par les nœuds internes à une profondeur donnée dans un arbre binaire en paramètre, dans un ordre quelconque.
3. Écrire la fonction `bintree_collect_canopy` qui renvoie la canopée d'un arbre binaire. La *canopée* d'un arbre binaire est la liste de `0` et de `1` dont l'entrée en i^{e} position renseigne sur l'orientation de la i^{e} feuille de l'arbre, où `0` (resp. `1`) exprime que la feuille est orientée vers la gauche (resp. droite).

```
# bintree_collect_values;;
- : bintree -> int list = <fun>
# bintree_collect_nodes example_tree;;
- : int list = [2; 2; 6; 5; 6; 4; 1; 2]

# bintree_collect_level;;
- : bintree -> int -> int list = <fun>
# bintree_collect_level example_tree 0;;
- : int list = [2]
# bintree_collect_level example_tree 1;;
- : int list = [2; 4]
# bintree_collect_level example_tree 2;;
- : int list = [6; 1]
# bintree_collect_level example_tree 3;;
- : int list = [5; 6; 2]
# bintree_collect_level example_tree 4;;
- : int list = []

# bintree_collect_canopy;;
- : bintree -> int list = <fun>
# bintree_collect_canopy example_tree;;
- : int list = [0; 0; 1; 0; 1; 0; 0; 1; 1]
```

Exercice 5. (Visiter)

1. Écrire la fonction `bintree_pre` qui renvoie la liste des entiers rencontrés dans un arbre binaire en paramètre lors du parcours préfixe gauche droite.
2. Écrire la fonction `bintree_post` qui renvoie la liste des entiers rencontrés dans un arbre binaire en paramètre lors du parcours suffixe gauche droite.
3. Écrire la fonction `bintree_in` qui renvoie la liste des entiers rencontrés dans un arbre binaire en paramètre lors du parcours infixe gauche droite.

```
# bintree_visit_pre;;
- : bintree -> int list = <fun>
# bintree_visit_pre example_tree;;
- : int list = [2; 2; 6; 5; 6; 4; 1; 2]

# bintree_visit_post;;
- : bintree -> int list = <fun>
# bintree_visit_post example_tree;;
- : int list = [5; 6; 6; 2; 2; 1; 4; 2]

# bintree_visit_in;;
- : bintree -> int list = <fun>
# bintree_visit_in example_tree;;
- : int list = [2; 5; 6; 6; 2; 4; 2; 1]
```

Exercice 6. (Rechercher)

Un *arbre binaire de recherche* est un arbre binaire tel que pour chaque nœud, les entiers apparaissant dans son sous-arbre gauche lui sont inférieurs et les entiers apparaissant dans son sous-arbre droit lui sont strictement supérieurs.

1. Écrire la fonction `bintree_insert` qui renvoie l'arbre binaire de recherche obtenu en insérant un entier dans un arbre binaire de recherche en paramètres.
2. Écrire la fonction `bintree_search` qui teste si un entier donné est dans un arbre binaire de recherche en paramètres.

```
# bintree_insert;;
- : bintree -> int -> bintree = <fun>

# bintree_insert Leaf 1;;
- : bintree = Node (1, Leaf, Leaf)

```

```
# bintree_insert (bintree_insert Leaf 1) 2;;
- : bintree = Node (1, Leaf, Node (2, Leaf, Leaf))

# bintree_insert (bintree_insert (bintree_insert Leaf 1) 2) 3;;
- : bintree = Node (1, Leaf, Node (2, Leaf, Node (3, Leaf, Leaf)))

# bintree_insert (bintree_insert (bintree_insert Leaf 1) 2) 0;;
- : bintree = Node (1, Node (0, Leaf, Leaf), Node (2, Leaf, Leaf))

# bintree_search;;
- : bintree -> int -> bool = <fun>
```

Exercice 7. (Modifier)

1. Écrire la fonction `bintree_double` qui renvoie un nouvel arbre binaire dans lequel tous les entiers d'un arbre binaire en paramètre ont été multipliés par 2.
2. Écrire la fonction `bintree_apply` paramétrée par une fonction de type `int -> int` et un arbre binaire et qui renvoie un nouvel arbre binaire dans lequel tous les entiers sont obtenus en appliquant cette fonction aux entiers des nœuds de l'arbre.

   Réécrire la fonction `bintree_double` en utilisant la fonction `bintree_apply`.
3. Écrire la fonction `bintree_mirror` qui renvoie un nouvel arbre binaire dans lequel les sous-arbres droits et gauches sont échangés à partir d'un arbre binaire en paramètre.
4. Écrire la fonction `bintree_sum_subtree` paramétrée par un arbre binaire et qui renvoie un nouvel arbre binaire dans lequel chaque nœud porte comme valeur la somme de tous les entiers apparaissant dans le sous-arbre enraciné en ce nœud de l'arbre en paramètre.

```
# bintree_double;;
- : bintree -> bintree = <fun>
# bintree_double example_tree;;
- : bintree =
Node (4,
   Node (4, Leaf, Node (12, Node (10, Leaf, Leaf), Node (12, Leaf, Leaf))),
   Node (8, Leaf, Node (2, Node (4, Leaf, Leaf), Leaf)))

# bintree_apply;;
- : bintree -> (int -> int) -> bintree = <fun>
# bintree_apply example_tree (function x -> x + 1);;
- : bintree =
Node (3,
   Node (3, Leaf, Node (7, Node (6, Leaf, Leaf), Node (7, Leaf, Leaf))),
   Node (5, Leaf, Node (2, Node (3, Leaf, Leaf), Leaf)))

# bintree_rotate;;
- : bintree -> bintree = <fun>
# bintree_rotate example_tree;;
- : bintree =
```

```
Node (2,
   Node (4, Node (1, Leaf, Node (2, Leaf, Leaf)), Leaf),
   Node (2, Node (6, Node (6, Leaf, Leaf), Node (5, Leaf, Leaf)), Leaf))

# bintree_sum_subtree;;
- : bintree -> bintree = <fun>
# bintree_sum_subtree example_tree;;
- : bintree =
Node (28,
   Node (19, Leaf, Node (17, Node (5, Leaf, Leaf), Node (6, Leaf, Leaf))),
   Node (7, Leaf, Node (3, Node (2, Leaf, Leaf), Leaf)))
```