Programmation fonctionnelle

Fiche de TP 4

L3 Informatique 2020-2021

Arbres binaires

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 inférieur, habituellement appelés *gauche* et *droit*. Du point de vue de ces éléments fils, l'élément dont ils sont issus au niveau supérieur est appelé *père*. Au niveau le plus élevé il y a donc un *nœud racine*. Au niveau directement inférieur, il y a au plus deux nœuds fils. En continuant à descendre aux niveaux inférieurs, on peut en avoir quatre, puis huit, seize, *etc.* c'est-à-dire la suite des puissances de 2. Un nœud n'ayant aucun fils est appelé *feuille*, sinon il est appelé *nœud interne*. Le nombre total de niveaux, autrement dit la distance entre une feuille parmi les plus éloignées et la racine, est appelé *hauteur de l'arbre*. Le niveau d'un nœud est appelé *profondeur*.

Exercice 1. (Type arbre binaire)

Écrire le type `bintree` permettant de représenter les arbres binaires définis sur les entiers. Utiliser ce type pour construire l'arbre binaire défini en figure 1 et y en liant le nom `example_tree`.

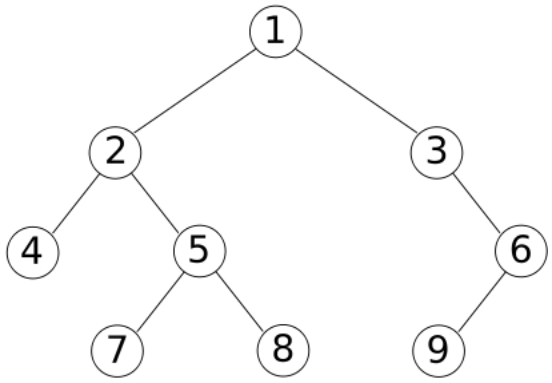

FIGURE 1 – Un arbre binaire sur les entiers.

Tous les exemples de la suite porteront sur l'arbre `example_tree`.

Exercice 2. (Compter)

1. Écrire la fonction `bintree_count_nodes` qui renvoie le nombre de nœuds dans un arbre binaire en paramètre.
2. Écrire la fonction `bintree_count_leaves` qui renvoie le nombre de feuilles dans un arbre binaire en paramètre.
3. Écrire la fonction `bintree_count_internal_nodes` qui renvoie le nombre de nœuds internes dans un arbre binaire en paramètre.
4. Écrire la fonction `bintree_count_right` qui renvoie le nombre d'arêtes orientées vers la droite dans un arbre binaire en paramètre (ce qui correspond aux nœuds internes qui ont des fils droits).
5. Écrire la fonction `bintree_count_left` qui renvoie le nombre d'arêtes orientées vers la gauche dans un arbre binaire en paramètre.(ce qui correspond aux nœuds internes qui ont des fils gauches).

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

# bintree_count_nodes example_tree;;
- : int = 9

# bintree_count_leaves;;
- : bintree -> int = <fun>

# bintree_count_leaves example_tree;;
- : int = 4

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

# bintree_count_internal_nodes example_tree;;
- : int = 5

# bintree_count_right example_tree;;
- : int = 4

# bintree_count_left example_tree;;
- : int = 4
```

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 des 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.

   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 Empty Empty;;
- : bool = true

# bintree_is_mirror (Node(1, Empty, Empty)) (Node(2, Empty, Empty));;
- : bool = true

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

# bintree_is_symmetric;;
- : bintree -> bool = <fun>

# bintree_is_symmetric Empty;;
- : bool = true

# bintree_is_symmetric (Node(1, Empty, Empty));;
- : bool = true

# bintree_is_symmetric (Node(1, Node(2, Empty, Empty), Node(2, Empty, Empty)));;
- : bool = true

# bintree_is_symmetric (Node(1, Node(2, Empty, Empty), Node(3, Empty, Empty)));;
- : bool = true

# bintree_is_symmetric (Node(1, Node(2, Empty, Empty), Empty));;
- : bool = false
```

Exercice 4. (Collecter)

1. Écrire la fonction `bintree_collect_nodes` qui renvoie la liste des entiers apparaissant dans un arbre binaire en paramètre.
2. Écrire la fonction `bintree_collect_leaves` qui renvoie la liste des entiers apparaissant dans les feuilles d'un arbre binaire en paramètre.

3. Écrire la fonction `bintree_collect_internal_nodes` qui renvoie la liste des entiers apparaissant dans les nœuds internes d'un arbre binaire en paramètre.
4. Écrire la fonction `bintree_collect_level` qui renvoie la liste des entiers apparaissant dans les nœuds à une profondeur donnée dans un arbre binaire en paramètre.
5. 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_nodes;;
- : bintree -> int list = <fun>

# bintree_collect_nodes example_tree;;
- : int list = [1; 2; 4; 5; 7; 8; 3; 6; 9]

# bintree_collect_leaves;;
- : bintree -> int list = <fun>

# bintree_collect_leaves example_tree;;
- : int list = [4; 7; 8; 9]

# bintree_collect_internal_nodes;;
- : bintree -> int list = <fun>

# bintree_collect_internal_nodes example_tree;;
- : int list = [1; 2; 5; 3; 6]

# bintree_collect_level;;
- : bintree -> int -> int list = <fun>

# bintree_collect_level example_tree 1;;
- : int list = [1]

# bintree_collect_level example_tree 2;;
- : int list = [2; 3]

# bintree_collect_level example_tree 3;;
- : int list = [4; 5; 6]

# bintree_collect_level example_tree 4;;
- : int list = [7; 8; 9]

# bintree_collect_level example_tree 5;;
- : int list = []

# bintree_collect_canopy example_tree;;
- : int list = [0; 0; 1; 0]
```

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 = [1; 2; 4; 5; 7; 8; 3; 6; 9]

# bintree_visit_post;;
- : bintree -> int list = <fun>

# bintree_visit_post example_tree;;
- : int list = [4; 7; 8; 5; 2; 9; 6; 3; 1]

# bintree_visit_in;;
- : bintree -> int list = <fun>

# bintree_visit_in example_tree;;
- : int list = [4; 2; 7; 5; 8; 1; 3; 9; 6]
```

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 Empty 1;;
- : bintree = Node (1, Empty, Empty)

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

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

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

# 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_rotate` 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 (2,
 Node (4, Node (8, Empty, Empty),
  Node (10, Node (14, Empty, Empty), Node (16, Empty, Empty))),
 Node (6, Empty, Node (12, Node (18, Empty, Empty), Empty)))

# bintree_apply;;
- : bintree -> (int -> int) -> bintree = <fun>

# bintree_apply example_tree (function x -> x + 1);;
- : bintree =
Node (2,
 Node (3, Node (5, Empty, Empty),
  Node (6, Node (8, Empty, Empty), Node (9, Empty, Empty))),
 Node (4, Empty, Node (7, Node (10, Empty, Empty), Empty)))

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

```
# bintree_rotate example_tree;;
- : bintree =
Node (1, Node (3, Node (6, Empty, Node (9, Empty, Empty)), Empty),
 Node (2, Node (5, Node (8, Empty, Empty), Node (7, Empty, Empty)),
  Node (4, Empty, Empty)))

# bintree_sum_subtree;;
- : bintree -> bintree = <fun>

# bintree_sum_subtree example_tree;;
- : bintree =
Node (45,
 Node (26, Node (4, Empty, Empty),
  Node (20, Node (7, Empty, Empty), Node (8, Empty, Empty))),
 Node (18, Empty, Node (15, Node (9, Empty, Empty), Empty)))
```