TP 4 -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*, le nœud initial étant appelé *racine*. 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 deux. Un nœud n'ayant aucun fils est appelé *feuille*, sinon il est appelé *nœud interne*. Le nombre de niveaux total, autrement dit la distance entre la feuille la plus éloignée et la racine, est appelé *hauteur de l'arbre*. Le niveau d'un nœud est appelé *profondeur*.

Exercice 1. Un type pour les arbres binaires

Écrire le type `bintree` permettant de représenter les arbres binaires définis sur les entiers. Utiliser ce type pour définir l'arbre binaire défini en Fig .1. (Tous les exemples dans la suite seront donnés sur cet arbre que nous noterons `example_tree`.)

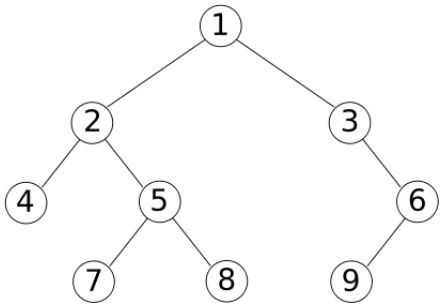

FIGURE 1 –

Exercice 2. Compter

1. Écrire la fonction `bintree_count_nodes` qui calcule le nombre de nœuds dans un arbre binaire.
2. Écrire la fonction `bintree_count_leaves` qui calcule le nombre de feuilles dans un arbre binaire.
3. Écrire la fonction `bintree_count_internal_nodes` qui calcule le nombre de nœuds internes dans un arbre binaire.

```
# 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
```

Exercice 3. Propriétés

1. Écrire la fonction `bintree_height` qui retourne la hauteur d'un arbre binaire.

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

Un arbre binaire est *symétrique* s'il est possible de tracer une ligne verticale passant par la racine telle que le sous-arbre gauche est l'image miroir du sous-arbre gauche.

2. Écrire la fonction `bintree_is_mirror` qui retourne vrai si un arbre binaire est l'image mirroir d'un autre arbre binaire. Nous nous intéressons ici à la structure des deux arbres binaires et pas aux valeurs des nœuds.
3. Écrire la fonction `bintree_is_symmetric` qui retourne vrai si un arbre binaire est symétrique.

```
# 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 retourne la liste des entiers stockés dans un arbre binaire.
2. Écrire la fonction `bintree_collect_leaves` qui retourne la liste des entiers stockés dans les feuilles d'un arbre binaire.

3. Écrire la fonction `bintree_collect_internal_nodes` qui retourne la liste des entiers stockés dans les nœuds internes d'un arbre binaire.
4. Écrire la fonction `bintree_collect_level` qui retourne la liste des entiers stockés dans les nœuds à hauteur `h` dans un arbre binaire.

```
# 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 = []
```

Exercice 5. Visiter

1. Écrire la fonction `bintree_pre` qui retourne la liste des entiers rencontrés dans un arbre binaire lors du parcours préfixe.
2. Écrire la fonction `bintree_post` qui retourne la liste des entiers rencontrés dans un arbre binaire lors du parcours postfixe.
3. Écrire la fonction `bintree_in` qui retourne la liste des entiers rencontrés dans un arbre binaire lors du parcours infixe.

```
# 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 stockés dans son sous-arbre gauche sont inférieurs ou égaux, et les entiers stockés dans son sous-arbre droit sont supérieurs.

1. Écrire la fonction `bintree_insert` qui insert un entier dans un arbre binaire de recherche.
2. Écrire la fonction `bintree_search` qui retourne vrai si un entier donné est dans un arbre binaire de recherche.

```
# 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 retourne un nouvel arbre binaire dans lequel tous les entiers ont été multiplié par deux.
2. Écrire la fonction `bintree_apply` qui prend en argument une fonction de type `int -> int` et retourne un nouvel arbre binanire tous les entiers sont obtenus en appliquant cette fonction à l'entier stocké dansle nœud. Réécricre la fonction `bintree_double` en utilisant la fonction `bintree_apply`.
3. Écrire la fonction `bintree_rotate` qui échange les sous-arbres gauche et droit en tout nœud.
4. Écrire la fonction `bintree_sum_subtree` qui retourne un nouvel arbre binaire dans lequel chaque entier est la somme de tous les entiers stockées dans le sous-arbre enraciné en ce nœud.

```
# 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)))
```