TME4 – Arbres Binaires de Recherche

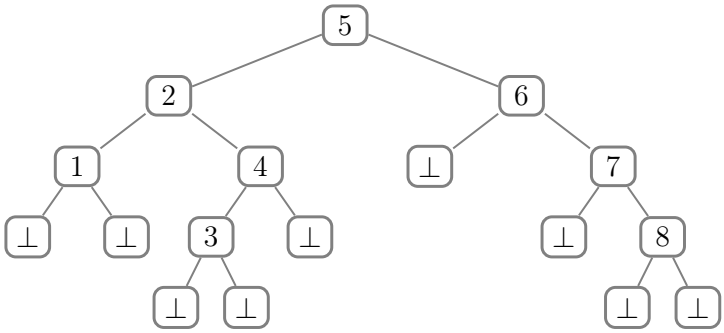
Pour représenter les arbres binaires, on définit le type :

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

Un *arbre binaire de recherche* `bt` est un arbre binaire tel que :

- `bt` est l'abre vide
- ou `bt` est de la forme `Node(x,bt1,bt2)` et
  - `x` est strictement supérieur à toutes les étiquettes de `bt1`
  - `x` est inférieur ou égal à toutes les étiquettes de `bt2`
  - `bt1` et `bt2` sont des arbres binaires de recherche

Par exemple, en utilisant ⊥ pour représenter `Empty`, la figure ci-dessous représente un arbre binaire de recherche :



Exercice 4.1 (Test d'ABR)**.**

1. Définir la fonction de signature `lt_btree (t:'a btree) (x:'a) : bool` telle que `(lt_btree t x)` est le booléen `true` si et seulement si toutes les étiquettes de `t` sont inférieures (au sens strict) à `x`. Pour tout `x`, `(lt_btree Empty x)=true`.
   Indication : Utiliser l'opérateur de comparaison polymorphe <.

```
# lt_btree (Node (2, Node (4, Empty, Empty), Empty)) 5;;
- : bool = true
# lt_btree Empty 5;;
- : bool = true
# lt_btree (Node (4, Node (6, Empty, Empty), Empty)) 5;;
- : bool = false
```

2. Définir la fonction de signature `ge_btree (t:'a btree) (x:'a) : bool` telle que `(ge_btree t x)` est le booléen `true` si et seulement si toutes les étiquettes de `t` sont supérieures ou égales à `x`. Pour tout `x`, `(ge_btree Empty x)=true`.
   Indication : Utiliser l'opérateur de comparaison polymorphe >=.

```
# ge_btree (Node (4, Node (5, Empty, Empty), Empty)) 4;;
- : bool = true
# ge_btree Empty 5;;
- : bool = true
# ge_btree (Node (4, Node (3, Empty, Empty), Empty)) 4;;
- : bool = false
```

3. Définir la fonction de signature `is_abr (bt:'a btree) : bool` qui détermine si un arbre est un arbre binaire de recherche.

```
# is_abr Empty;;
- : bool = true
# is_abr (Node (5, Node (2, Node (1, Empty, Empty),
                            Node (4, Empty, Empty)),
                    Node (6, Empty, Node (7, Empty, Empty)))) ;;
- : bool = true
# is_abr (Node (5, Node (2, Node (1, Empty, Empty),
                            Node (4, Empty, Empty)),
           Node (3, Empty, Node (7, Empty, Empty))));;
- : bool = false
# is_abr (Node (5, Node (2, Node (1, Empty, Empty),
                            Node (4, Empty, Empty)),
                    Node (7, Empty, Node (6, Empty, Empty))));;
- : bool = false
```

Exercice 4.2 (Recherche dans un ABR)**.**
Définir la fonction de signature `mem (bt: 'a btree) (x:'a) : bool` telle que `(mem x bt)` est le booléen `true` si l'élément `x` est présent dans `bt` (et `false` sinon). On fait l'hypothèse que `bt` est un arbre binaire de recherche et dans ce cas, les propriétés des arbres binaires de recherche permettent d'optimiser la recherche.

```
# mem Empty 4;;
- : bool = false
# mem (Node (5, Node (2, Node (1, Empty, Empty),
                         Node (4, Empty, Empty)),
                 Node (6, Empty, Node (7, Empty, Empty)))) 4;;
- : bool = true
# mem (Node (5, Node (2, Node (1, Empty, Empty),
                         Node (4, Empty, Empty)),
                 Node (6, Empty, Node (7, Empty, Empty)))) 8;;
- : bool = false
```

Exercice 4.3 (Construction d'ABR)**.**

1. Définir la fonction de signature `insert (bt:'a btree) (x:'a) : 'a btree` qui construit l'arbre binaire de recherche obtenu en insérant l'élément `x` dans l'arbre binaire de recherche `bt`. Ainsi, si `is_abr bt = true`, alors `is_abr (insert x bt) = true` pour tout `x`.

   ```
   # insert Empty 4;;
   - : int btree = Node (4, Empty, Empty)

   # insert (Node (5, Node (2, Node (1, Empty, Empty),
                               Node (4, Empty, Empty)),
                       Node (6, Empty, Node (7, Empty, Empty))))   3;;
   - : int btree =
   Node
    (5,
     Node (2, Node (1, Empty, Empty), Node (4, Node (3, Empty, Empty), Empty)),
     Node (6, Empty, Node (7, Empty, Empty)))
   ```

```
# insert (Node (5, Node (2, Node (1, Empty, Empty),
                            Node (4, Empty, Empty)),
                  Node (6, Empty, Node (7, Empty, Empty)))) 4;;
- : int btree =
Node
 (5,
  Node (2, Node (1, Empty, Empty), Node (4, Empty, Node (4, Empty, Empty))),
  Node (6, Empty, Node (7, Empty, Empty)))
```

2. Définir la fonction de signature `abr_of_list (l:'a list) : 'a btree` qui construit un arbre binaire de recherche contenant les éléments de la liste `l`.

```
# abr_of_list [3;1;2;5;31;2;42;18;2];;
- : int btree =
Node
 (3,
  Node (1, Empty, Node (2, Empty, Node (2, Empty, Node (2, Empty, Empty)))),
  Node
   (5, Empty, Node (31, Node (18, Empty, Empty), Node (42, Empty, Empty))))
```

Remarque. Plusieurs arbres binaires de recherche différents peuvent contenir le même ensemble d'éléments : la forme de l'arbre dépend de la fonction `abr_of_list`.

Exercice 4.4 (Tri par ABR)**.**

1. Définir la fonction de signature `list_of_abr (bt:'a btree) : 'a list` qui construit la liste contenant des étiquettes de `bt` en parcours infixe.

```
# list_of_abr (Node(3,Node (1, Empty,
                              Node (2, Empty,
                                       Node (2, Empty, Node (2, Empty, Empty)))),
                      Node(5, Empty, Node (31, Node (18, Empty, Empty),
                                               Node (42, Empty, Empty)))));;
- : int list = [1; 2; 2; 2; 3; 5; 18; 31; 42]
```

2. Définir la fonction de signature `abr_sort (l:'a list) : 'a list` qui trie la liste `l` passée en argument. Cette fonction utilise les fonctions définies précédemment et n'est pas récursive.

```
# abr_sort [12;3;5;42;1;18;21];;
- : int list = [1; 3; 5; 12; 18; 21; 42]
```