

Les arbres binaires

On rappelle le type `tree[A]` des arbres binaires génériques, où une valeur est associée à chaque nœud, vu en cours :

```
type tree[A] = Empty[A] | Node[A]

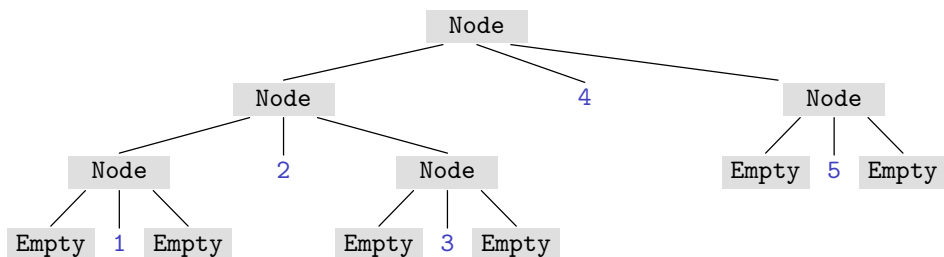
@dataclass
class Empty[A]:
  pass

@dataclass
class Node[A]:
  left: tree[A]
  value: A
  right: tree[A]
```

Voici un exemple d'arbre de type `tree[int]` contenant 5 nœuds :

```
Node(
  Node(
    Node(
      Empty(),
      1,
      Empty()),
    2,
    Node(
      Empty(),
      3,
      Empty())),
  4,
  Node(
    Empty(),
    5,
    Empty()))
```

Cet arbre peut aussi être représenté sous forme de diagramme :



1. Définir un arbre de type `tree[Optional[str]]` contenant au moins 5 nœuds et au moins 3 chaînes de caractères.
2. Ecrire une fonction `mk_leaf[A](v : A) -> tree[A]` qui construit un arbre réduit à un unique nœud (une feuille).
3. Ecrire une fonction `traversal[A](t: tree[A]) -> list[A]` qui construit la liste des valeurs associée aux nœuds d'un arbre parcouru dans l'ordre infix (de gauche à droite).
4. Ecrire une fonction `tmap[A, B](f: Callable[[A], B], t: tree[A]) -> tree[B]` telle que `tmap(f, t)` renvoie l'arbre obtenu en appliquant la fonction `f` aux valeurs contenues dans les nœuds de l'arbre `t`.
5. Vérifier sur quelques exemples que `map(f, traversal(t)) = traversal(tmap(f, t))` (où la fonction `map` est celle déjà implantée pour les listes).

On rappelle maintenant le type `ltree[A]` des arbres binaires génériques où une valeur est associé à chaque feuille, aussi vu en cours :

```

type ltree[A] = Leaf[A] | LNode[A]

@dataclass
class Leaf[A]:
    value: A

@dataclass
class LNode[A]:
    left: ltree[A]
    right: ltree[A]

```

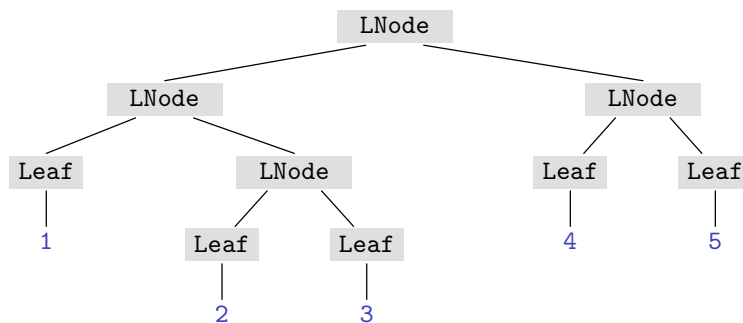
Voici un exemple d'arbre de type `ltree[int]` contenant 4 noeuds et 5 feuilles:

```

LNode(
  LNode(
    Leaf(1),
    LNode(
      Leaf(2),
      Leaf(3))),
  LNode(
    Leaf(4),
    Leaf(5)))

```

Cet arbre peut aussi être représenté sous forme de diagramme :



1. Ecrire une fonction `ltraversal[A](t: ltree[A]) -> list[A]` qui construit la liste des feuilles d'un arbre parcouru de gauche à droite.
2. Ecrire une fonction `lmap[A, B](f: Callable[[A], B], t: ltree[A]) -> ltree[B]` telle que `lmap(f, t)` renvoie l'arbre obtenu en appliquant la fonction `f` à chaque feuille de `t`.
3. Ecrire une fonction `convert[A](t: ltree[A]) -> tree[Optional[A]]` qui convertit un arbre de type `ltree[A]` en arbre de type `tree[Optional[A]]`. On utilisera la fonction `mk_leaf` pour construire les feuilles.
4. Verifier sur des exemples ce que calcule la fonction `traversal` o `convert`. Comment peut-on utiliser cette fonction pour redéfinir `ltraversal` ?
5. Ecrire une fonction `display[A](t: ltree[A], depth: int = 0) -> str` qui affiche un arbre de type `ltree[A]` sur plusieurs lignes et correctement indenté comme dans l'exemple ci-dessus (le paramètre `depth` correspond à la profondeur d'indentation).