Computer Systems Modeling and Verification (USEEN1)

Part I: Binary Trees

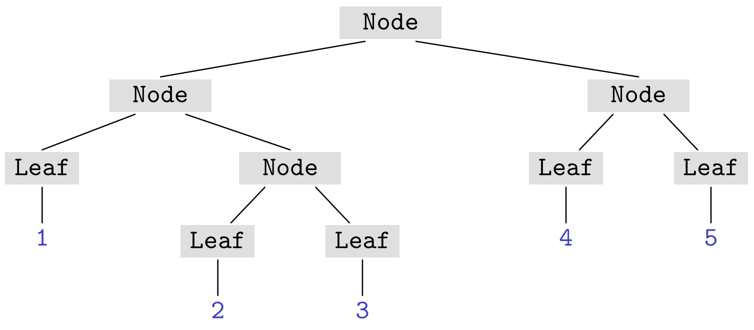
Recall the type `tree[A]` of generic binary trees where each leaf contains a value (seen in class):

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

@dataclass
class Leaf[A]:
    value: A

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

Here is an example of tree of type `tree[int]` containing 4 nodes and 5 leaves:

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

The tree could also be displayed as the following drawing:



Exercises.

1. Define the type `tree[int]` given above and check that the given example is well typed. Test your code by printing this tree (using the standard print function).

2. Define a function `display_inline[A](t: tree[A]) -> str` which displays a tree as a string on a single line (similarly to what the print function does).

3. Define a function `display[A](t: tree[A], depth: int = 0) -> str` which displays a tree as a string as in the example above (using the `depth` parameter as indentation level).

4. Define a function `traversal[A](t: tree[A]) -> list[A]` which builds the list of leaves of a tree traversed from left to right.

5. Define a function `tmap[A, B](f: Callable[[A], B], t: tree[A]) -> tree[B]` such that `tmap(f, t)` returns the tree obtained by applying the function `f` to the values contained in the leaves of tree `t`.

6. Check on some example that `list(map(f, traversal(t))) = traversal(tmap(f, t))` (where `map` is the predefined function that returns an iterable).