4. Completá la implementación del tipo Árbol Binario dada en el teórico, donde utilizamos la siguiente representación:

implement Tree **of** T **where**

type Node **of** T = **tuple**
left: **pointer to** (Node **of** T)
value: T
right: **pointer to** (Node **of** T)
end tuple

type Tree **of** T= **pointer to** (Node **of** T)

```
implement Tree of T where

type Node of T = tuple
                   left: pointer to (Node of T)
                   value: T
                   right: pointer to (Node of T)
                 end tuple

type Tree of T = pointer to (Node of T)

type Direction = enum
                   Left
                   Right
                 end enum

type Path = List of Direction

constructors
    fun empty_tree() ret t: Tree of T
        t := null
    end fun

    fun node(tl: Tree of T, e: T, tr: Tree of T) ret t: Tree of T
        alloc(t)
        t→left := null
        t→value := e
        t→right := null
    end fun

operations
    fun is_empty_tree(t: Tree of T) ret b: bool
        b := (t = null)
    end fun

    {- PRE: not is_empty_tree(t) -}
    fun root(t: Tree of T) ret e: T
        e := t→value
    end fun

    {- PRE: not is_empty_tree(t) -}
    fun left(t: Tree of T) ret tl: Tree of T
        tl := t→left
    end fun
```

```
{- PRE: not is_empty_tree(t) -}
fun right(t: Tree of T) ret tr: Tree of T
    tr := t→right
end fun

fun height(t: Tree of T) ret n: nat
    if is_empty_tree(t) then
        n := 0
    else
        n := 1 + (height(t→left) `max` height(t→right))
    fi
end fun

fun is_path(t: Tree of T, p: Path) ret b: bool
    var aux: Path
    var d: Direction

    if is_empty_tree(t) then
        b := false
    else
        if is_empty_list(p) then
            b := true
        else
            aux := copy_list(p)
            d := head(aux)
            tail(aux)

            if d = Left then
                b := is_path(t→left,aux)
            else if d = Right then
                b := is_path(t→right,aux)
            fi

            destroy_list(aux)
        fi
    fi
end fun

fun subtree_at(t: Tree of T, p: Path) ret t0: Tree of T
    var d: Direction
    var aux: Path

    if is_empty_tree(t) then
        t0 := t
    else
        d := head(p)
        aux := copy_list(p)
        tail(aux)

        if d = Left then
            t0 := subtree_at(t→left,aux)
        else if d = Right then
            t0 := subtree_at(t→right,aux)
        fi

        destroy_list(aux)
    fi
end fun
```

```
    {-  PRE: is_path(t,p) -}
    fun elem_at(t: Tree of T, p: Path) ret e: T
        e := root(subtree_at(t,p))
    end fun

end implement
```