101. Dados un árbol y un elemento cualquiera, determinar si el elemento pertenece al árbol

Escriba una función **`tree-member?`** que devuelve verdadero cuando el elemento es una de las etiquetas de los nodos del árbol, y falso en otro caso.

Tenemos dos opciones con el mismo enunciado:

a) el árbol es binario
b) b) el árbol es *n*-ario.

En ambos casos debemos tomar una representación para el árbol, y luego definir funciones de acceso si las necesitamos.

El caso binario

Un árbol binario se puede representar como una terna:

```
(etiqueta arbol-izquierdo arbol-derecho)
```

Cuando un nodo carece de un descendiente, podemos utilizar una lista vacía en lugar de ese descendiente.

El elemento que buscamos ocupa el lugar que hemos denominado etiqueta.

Entonces un elemento buscado sólo puede alojarse en las posiciones donde hay etiquetas. De todo el árbol, la etiqueta del nodo actual es la accesible.

Eso nos da una forma de resolver el problema: el elemento buscado sólo puede esconderse en la etiqueta del nodo actual, o en alguno de los subárboles. Éstos son los únicos casos en que la función devolverá verdadero.

Vamos a comenzar a codificar esa idea, y usaremos el nombre **`bintree-member?`** para la función:

```
(define bintree-member?
  (lambda (elem node)
    (or (equal? elem (car node))
        (bintree-member? elem (cadr node))
        (bintree-member? elem (caddr node)))))
```

Cada vez que vamos por un subárbol está la posibilidad que sea una lista vacía, que representa la ausencia de descendientes. Si **`node`** referencia una lista vacía van a dar error

las operaciones **car** y **cdr** sobre **node**. Debemos protegernos de ese caso con una guarda, un condicional que no deja pasar la ejecución de ese punto:

```
(define bintree-member?
  (lambda (elem node)
    (if (null? node) #f
      (or (equal? elem (car node))
          (bintree-member? elem (cadr node))
          (bintree-member? elem (caddr node))))))
```

Podemos reescribirlo usando AND y OR pues se trata sólo de expresiones booleanas:

```
(define bintree-member?
  (lambda (elem node)
    (and (not (null? node))
         (or (equal? elem (car node))
             (bintree-member? elem (cadr node))
             (bintree-member? elem (caddr node))))))
```

Se puede apreciar que el código es la evaluación de expresiones.

Una versión del código más imperativa hubiera utilizado formas **if** en lugar de las formas **and** y **or**:

```
(define bintree-member?
  (lambda (elem node)
    (if (null? node) #f
      (if (equal? elem (car node)) #t
          (if (bintree-member? elem (cadr node)) #t
              (bintree-member? elem (caddr node)))))))
```

El caso *n*-ario

Cambia la representación.

```
(etiqueta arbol1 arbol2 arbol3 … arboln)
```

Con esta representación el **car** es la **etiqueta**, y el **cdr** es un “bosque”, una lista de árboles, que puede ser vacía si no hay descendientes.

Entonces el elemento buscado puede coincidir con la etiqueta del nodo actual, o pertenecer a alguno de los subárboles. Un elemento no pertenece a una lista vacía de subárboles.

La expresión siguiente devuelve una lista con los subárboles donde existe **elem**:

```
(filter (lambda (subtree) (tree-member? elem subtree))
        (cdr node))
```

Si la lista devuelta está vacía es que **elem** no está en ninguno de los subárboles.

Aplicamos eso a nuestro programa:

```
(define tree-member?
  (lambda (elem node)
    (and (not (null? node))
         (or (equal? elem (car node))
             (not (null?
                    (filter
                      (lambda (subtree) (tree-member? elem subtree))
                      (cdr node))))))))
```

En lugar de **filter** podemos usar una función auxiliar, que evalúe la pertenencia a los subárboles de un nodo dado:

```
(define forest-member?
 (lambda (elem forest)
   (if (null? forest) #f
     (let ((node (car forest))
           (rest (cdr forest)))
      (or (tree-member? elem node)
          (forest-member? elem rest))))))

(define tree-member?
  (lambda (elem node)
    (if (null? node) #f
      (or (equal? elem (car node))
          (forest-member? elem (cdr node))))))
```

¿Cuál es la diferencia con **filter**?

- **filter** evalúa la función en todos los elementos de su lista argumento; pues **filter** es una función
- **forest-member?** usa la forma **or** (no es una función **or**) la cual solamente evalúa su argumento hasta que encuentra el primer valor verdadero.

Entonces, en principio, **forest-member?** se comportará mucho mejor (menos evaluaciones) si hay una etiqueta coincidente con el elemento buscado en la parte más cercana del árbol. Otra forma de codificarlo sería usando la función **fold**, pero queda para otro ejercicio más tarde.