Exercise 2.28.

Write a procedure `fringe` that takes as argument a tree (represented as a list) and returns a list whose elements are all the leaves of the tree arranged in left-to-right order. For example,

```
(define x (list (list 1 2) (list 3 4)))

(fringe x)
(1 2 3 4)

(fringe (list x x))
(1 2 3 4 1 2 3 4)
```

Answer.

To implement `fringe`, do the following:

- `Fringe` of an empty list is the empty list itself.
- `Fringe` of a tree `t` is the `fringe` of the `car` of `t` combines `fringe` of the `cdr` of `t`.
- `Fringe` of a leaf is simply a list contains that leaf.

Remember that we've seen in section 2.2.1 that `append` is a procedure which combines two list into a single one. Therefore, the procedure `finge` can be expressed as:

```
(define (fringe t)
  (cond ((null? t) nil)
        ((not (pair? (car t))) t)
        (else
          (append (fringe (car t))
                  (fringe (cdr t))))))
```