PoPL Homework 1

Due: Wednesday Jan 18, 2023 11:59pm

Difficulty: Medium

The following Tree datatype implements a binary tree with a number in each node and leaf:

```
(define-type Tree
  (leaf [val : Number])
  (node [val : Number]
        [left : Tree]
        [right : Tree]))
```

Part 1 — Sum

Implement a *sum* function that takes a tree and returns the sum of the numbers in the tree.

Example: (sum (node 5 (leaf 6) (leaf 7))) should produce 18.

Part 2 — Negate

Implement the function *negate*, which takes a tree and returns a tree that has the same shape, but with all the numbers negated.

Example: (negate (node 5 (leaf 6) (leaf 7))) should produce (node -5 (leaf -6) (leaf -7)).

Part 3 — Contains?

Implement the function *contains?*, which takes a tree and a number and returns #t if the number is in the tree, #f otherwise.

Example: (contains? (node 5 (leaf 6) (leaf 7)) 6) should produce #t.

The second argument to the contains? function is “along for the ride.”

Part 4 — Big Leaves?

Implement the function big-leaves?, which takes a tree and returns #t if every leaf is bigger than the sum of numbers in the path of nodes from the root that reaches the leaf.

Examples: (big-leaves? (node 5 (leaf 6) (leaf 7))) should produce #t, while (big-leaves? (node 5 (node 2 (leaf 8) (leaf 6)) (leaf 7))) should produce #f (since 6 is smaller than 5 plus 2).

The big-leaves? function should be a thin wrapper on another function, perhaps bigger-leaves?, that accumulates a sum of node values.

Part 5 — Positive Trees?

Implement the function *positive-trees?*, which takes a list of trees and returns #t if every member of the list is a positive tree, where a positive tree is one whose numbers sum to a positive value.

Hint 1: This function takes a list, not a tree, so don't try to use the template for a tree function.

Hint 2: Use your sum function as a helper.

Hint 3: (positive-trees? empty) should produce #t, because there's no tree in the empty list whose numbers sum to 0 or less.

More examples:

```
  (test (positive-trees? (cons (leaf 6)
                               empty))
        #t)

  (test (positive-trees? (cons (leaf -6)
                               empty))
        #f)

  (test (positive-trees? (cons (node 1 (leaf 6) (leaf -6))
                               empty))
        #t)

  (test (positive-trees? (cons (node 1 (leaf 6) (leaf -6))
                               (cons (node 0 (leaf 0) (leaf 1))
                                     empty)))
        #t)

  (test (positive-trees? (cons (node -1 (leaf 6) (leaf -6))
                               (cons (node 0 (leaf 0) (leaf 1))
                                     empty)))
        #f)
```

Part 6 -Transmutation

Implement a *positive-thinking* function, which takes a Tree as argument and produces a new tree which removes all negative leaves from the original tree. If the resulting tree would have no nodes, return false.

Hint: the total number of nodes (leaf and interior) in the resulting tree will be less than or equal to the total number of nodes in the original, minus the number of negative leaves in the original.