Question 1: That’s odd...

Create a function \((\text{odd-lengths } t)\) that consumes a \(\text{BST}\) and returns a new version of \(t\) where all values that are strings of odd length are replaced with the string "odd".

\[
(\text{define-struct node (key val left right)})
\]

;; A binary search tree (BST) is either
;; * () or
;; * (make-node Nat Any BST BST)...

Question 2: Go west, young man...

Create a function \(\text{leftmost}\) that consumes a non-empty \(\text{SSTree}\) and returns the leftmost label in the tree.

;; a simple search tree (SSTree) is either
;; * () or
;; * a SNode, where keys in left are less than key, and in right greater.

Question 3: Adding additions

Create a function \(\text{count-ops}\) that consumes a \(\text{BinExp}\) and returns the number of operations in that expression.

;; an Operator is (anyof '+ '-' '* '/)
\[
(\text{define-struct binode (op arg1 arg2)})
\]

;; a binary arithmetic expression internal node (BINode)
;; is a (make-binode Operator BinExp BinExp)
;; A binary arithmetic expression (BinExp) is either:
;; a Num or
;; a BINode

Question 4: Four Wonderful Bats, Ah Ah Ah!

Create a function \((\text{l lt-count } value \ T)\) that consumes an \(\text{Num}\) and a \(\text{LLT}\) and counts the number of times that value appears in \(T\).

\[
(\text{l lt-count 7 (list 7 (list 4 (list 7 6 5) 7))}) \Rightarrow 3
\]

! Do not use flatten.

;; a leaf-labelled tree (LLT) is either
;; a Num or
;; a non-empty (listof LLT).
Write a function `swap-ops` that consumes an `AExp` and returns a new `AExp` in which the `*` and `+` operations have been swapped (i.e. all `*` become `+` and all `+` become `*`).

```scheme
;; an Operator is (anyof `+` `-` `*` `/)
(define-struct ainode (op args))
;; an arithmetic expression internal node (AINode)
;; is a (make-ainode Operator (listof AExp))
;; An arithmetic expression (AExp) is either:
;;   a Num or
;;   a AINode
```