Solving Tree Problems

Implement `bigs`, which takes a tree instance containing integer labels. It returns the number of nodes in t whose labels are larger than any labels of their ancestor nodes.

```python
def bigs(t):
    # The root label is always larger than all of its ancestors
    if t:
        return sum((bigs(a) for a, x in t, if node.label > max_ancestors)
    return 0

# Some initial value for the largest ancestor so far...
max_ancestor = -1
```

Recursive Accumulation

Tree-Structured Data

A tree can contain other trees:

```
S = (N1, N2, N3, [N4], [N5, [N6]])
```

From Problem Analysis to Data Definitions

Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

Signature, Purpose Statement, Header

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

Functional Examples

Work through examples that illustrate the function’s purpose.

Function Template

Transcribe the data definitions into an outline of the function.

Function Definition

Fill in the gaps in the function template, exploit the purpose statement and the examples.

Testing

Articulate the examples as tests and ensure that the function passes all, being sure to document any assumptions you also implement examples in that they help others read and understand the definition when the code itself or the name of the function calls on subtrees

Designing Functions

How to Design Programs

From Problem Analysis to Data Definitions

Identify the information that must be represented and how it is represented in the chosen programming language. Formulate data definitions and illustrate them with examples.

Signature, Purpose Statement, Header

State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes. Define a stub that lives up to the signature.

Functional Examples

Work through examples that illustrate the function’s purpose.

Function Template

Transcribe the data definitions into an outline of the function.

Function Definition

Fill in the gaps in the function template, exploit the purpose statement and the examples.

Testing

Articulate the examples as tests and ensure that the function passes all, being sure to document any assumptions you also implement examples in that they help others read and understand the definition when the code itself or the name of the function calls on subtrees.
Designing a Function

Implement `smalls`, which takes a Tree instance `t` containing integer labels. It returns the non-leaf nodes in `t` whose labels are smaller than any labels of their descendant nodes.

```
def smalls(t):
    """Return the non-leaf nodes in t that are smaller than all their descendants."
    return []
```

```
# Example
a = Tree(1, [Tree(2, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(6)])])])

sorted([t.label for t in smalls(a)])
```

**result** = []

```
def process(t):
    process(t)
    return result
```

**Signature**: `Tree -> List of Trees`

**if** `t` is leaf():

**else**:

    smallest = ______________________________________
    if ________________________________________________:
        _______________________________________________
        return min(smallest, t.label)
    else:
        return process(...) + [t]

**Signature**: `Tree -> number`

"""Find smallest label in t & maybe add t to result"""

```
t.label < smallest
```

```
result.append(    )
t
```

Expression Trees

Interpreter Analysis

How many times does `scheme_eval` get called when evaluating the following expressions?

```
(define x (+ 1 2))
(define (f y) (+ x y))
(f (if (> 3 2) 4 5))
```

```
1 2 3 4 5
```

```
2 4 5
```

```
0 2
```

```
0 2
```

```
0 2 4 5
```

```
0 2 4 5
```