AIT Semantic and Declarative Technologies Course

Homework P2-3: Trees

In the problem set below we use the data structure *itree* (integer tree) which is defined as follows.

A Prolog term is an *itree* if and only if:

- it is a leaf, i.e., a `leaf(`n`)` compound, where n is an integer; or
- it is a node, i.e., a `node(`t_1`,`t_2`)` compound, where the two arguments – t_1 and t_2 – are both *itrees*.

We will refer to this data structure simply as a *tree* or as a *binary tree*.

For each task write a Prolog predicate that corresponds to the specification provided as a comment. You can use built-in predicates mentioned in the lecture slides so far, e.g. `number(Term)` and `Y is Expr`.

A sample task with solution

Given a binary tree, count the number of nodes in it, i.e., calculate the number of occurrences of the compound `node(`t_1`,`t_2`)` within the tree data structure.

```
% tree_node_count(+Tree, ?N): Binary tree Tree has N nodes.

| ?- tree_node_count(leaf(1), N).
N = 0 ? ; no
| ?- tree_node_count(leaf(1), 0).
yes
| ?- tree_node_count(leaf(1), 1).
no
| ?- tree_node_count(node(leaf(1),node(leaf(2),leaf(3))), N).
N = 2 ? ; no
| ?- tree_node_count(node(leaf(1),node(leaf(2),node(leaf(4),leaf(3)))), N).
N = 3 ? ; no
```

A solution:

```
% tree_node_count(+Tree, ?N): Binary tree Tree has N nodes.
tree_node_count(leaf(_), 0).
tree_node_count(node(L, R), N):-
        tree_node_count(L, N1),
        tree_node_count(R, N2),
        N is N1 + N2 + 1.
```

1. Incrementing all leaf values of a tree

```
% increment_tree(+Tree0, ?Tree): Tree is obtained from binary tree
% Tree0 by incrementing each leaf value by 1.

| ?- increment_tree(leaf(1), Tree).
Tree = leaf(2) ? ; no
| ?- increment_tree(node(leaf(1),node(leaf(2),leaf(3))), Tree).
Tree = node(leaf(2),node(leaf(3),leaf(4))) ? ; no
```

2. Finding the rightmost leaf value

```
% tree_rightmost_value(+Tree, ?Value): Tree is a binary tree and
% the integer in its rightmost leaf is Value.

| ?- tree_rightmost_value(leaf(1), V).
V = 1 ? ; no
| ?- tree_rightmost_value(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), V).
V = 3 ? ; no
```

3. Calculating tree depth

   The depth of a tree is the maximal number of `node(...)` structures nested into each other.

```
% tree_depth(+Tree, ?D): Tree is a binary tree of depth D.

| ?- tree_depth(leaf(3), D).
D = 0 ? ; no
| ?- tree_depth(leaf(5), 1).
no
| ?- tree_depth(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), D).
D = 2 ? ; no
| ?- tree_depth(node(leaf(1),node(leaf(2),node(leaf(4),leaf(3)))), D).
D = 3 ? ; no
```

   Hint: you can use the function `max` in arithmetic expressions, e.g.

```
| ?- X is max(2,3)+1. ---> X = 4 ? ; no
```

4. Finding a leaf value in a tree

```
% tree_leaf_value(+Tree, +V): V is present as a leaf value in Tree.

| ?- tree_leaf_value(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), 3).
yes
| ?- tree_leaf_value(node(node(leaf(1),leaf(4)),node(leaf(2),leaf(3))), 5).
no
```

   Most of the time the program for the above task works also for the input-output mode `tree_leaf_value(+Tree, -V)`, i.e., if the second argument is not an integer, but a variable. In such a case the predicate should enumerate all the leaf values, as shown in the example run below. (The order of the solutions does not matter.) Make your code work for this I/O mode as well.

```
| ?- tree_leaf_value(node(node(leaf(1),leaf(4)), node(leaf(2),leaf(3))), V).
V = 1 ? ; V = 4 ? ; V = 2 ? ; V = 3 ? ; no
```

5. Obtaining the list of the leaf values of a tree

```
% tree_leaves(+Tree, ?L): L is the list of all leaf values in Tree in left-to-right order.

| ?- tree_leaves(node(node(leaf(1),leaf(4)),node(leaf(1),leaf(3))), L).
L = [1,4,1,3] ? ; no

| ?- tree_leaves(leaf(1), L).
L = [1] ? ; no

no
```

   Hint: re-use the `list_concat` predicate from an earlier practice.