# **CIS 61 :: Lab 06 - Trees**

### Q1: Acorn Finder

The squirrels on campus need your help! There are a lot of trees on campus and the squirrels would like to know which ones contain acorns. Define the function acorn\_finder, which takes in a tree and returns True if the tree contains a node with the value 'acorn' and False otherwise.

|  |
| --- |
| **def** **acorn\_finder**(t):  """Returns True if t contains a node with the value 'acorn' and False otherwise.  >>> scrat = tree('acorn')  >>> acorn\_finder(scrat)  True  >>> sproul=tree('roots',[tree('branch1',[tree('leaf'),tree('acorn')]), tree('branch2')])  >>> acorn\_finder(sproul)  True  >>> numbers = tree(1, [tree(2), tree(3, [tree(4), tree(5)]), tree(6, [tree(7)])])  >>> acorn\_finder(numbers)  False  """  "\*\*\* YOUR CODE HERE \*\*\*" |

### Q2: Pruning Leaves

Define a function prune\_leaves that given a tree t and a tuple of values vals, produces a version of t with all its leaves that are in vals removed. Do not attempt to try to remove non-leaf nodes and do not remove leaves that do not match any of the items in vals. Return None if pruning the tree results in there being no nodes left in the tree.

|  |
| --- |
| **def** **prune\_leaves**(t, vals):  """Return a modified copy of t with all leaves that have a label that appears  in vals removed. Return None if the entire tree is pruned away.  >>> t = tree(2)  >>> print(prune\_leaves(t, (1, 2)))  None  >>> numbers = tree(1, [tree(2), tree(3, [tree(4), tree(5)]), tree(6,[tree(7)])])  >>> print\_tree(numbers)  1  2  3  4  5  6  7  >>> print\_tree(prune\_leaves(numbers, (3, 4, 6, 7)))  1  2  3  5  6  """  "\*\*\* YOUR CODE HERE \*\*\*" |

### Q3: Sprout leaves

Define a function sprout\_leaves that takes in a tree, t, and a list of values, vals. It produces a new tree that is identical to t, but where each old leaf node has new branches, one for each value in vals.

For example, say we have the tree t = tree(1, [tree(2), tree(3, [tree(4)])]):

1

/ \

2 3

|

4

If we call sprout\_leaves(t, [5, 6]), the result is the following tree:

1

/ \

2 3

/ \ |

5 6 4

/ \

5 6

|  |
| --- |
| **def** **sprout\_leaves**(t, vals):  """Sprout new leaves containing the data in vals at each leaf in  the original tree t and return the resulting tree.  >>> t1 = tree(1, [tree(2), tree(3)])  >>> print\_tree(t1)  1  2  3  >>> new1 = sprout\_leaves(t1, [4, 5])  >>> print\_tree(new1)  1  2  4  5  3  4  5  >>> t2 = tree(1, [tree(2, [tree(3)])])  >>> print\_tree(t2)  1  2  3  >>> new2 = sprout\_leaves(t2, [6, 1, 2])  >>> print\_tree(new2)  1  2  3  6  1  2  """ |

### Q4: Height of a Tree

Write a function that returns the height of a tree. Recall that the height of a tree is the length of the longest path from the root to a leaf.

|  |
| --- |
| def height(t):  """Return the height of a tree.  >>> t = tree(3, [tree(5, [tree(1)]), tree(2)])  >>> height(t)  2  """ |

### Q5: Double Tree

Write a function that takes in a tree and doubles every value. It should return a new tree. You can assume that every item is a number.

|  |
| --- |
| def double\_tree(t):  """Return a tree with the square of every element in t  >>> numbers = tree(1,  [tree(2,  [tree(3),  tree(4)]),  tree(5,  [tree(6,  [tree(7)]),  tree(8)])])  >>> print\_tree(double\_tree(numbers))  2  4  6  8  10  12  14  16  """ |

### (Optional) Q6: Add trees

Define the function add\_trees, which takes in two trees and returns a new tree where each corresponding node from the first tree is added with the node from the second tree. If a node at any particular position is present in one tree but not the other, it should be present in the new tree as well.

*Hint*: You may want to use the built-in zip function to iterate over multiple sequences at once.

*Note*: If you feel that this one's a lot harder than the previous tree problems, that's totally fine! This is a pretty difficult problem, but you can do it! Talk about it with other students, and come back to it if you need to.

|  |
| --- |
| **def** **add\_trees**(t1, t2):  """  >>> numbers = tree(1,  ... [tree(2,  ... [tree(3),  ... tree(4)]),  ... tree(5,  ... [tree(6,  ... [tree(7)]),  ... tree(8)])])  >>> print\_tree(add\_trees(numbers, numbers))  2  4  6  8  10  12  14  16  >>> print\_tree(add\_trees(tree(2), tree(3, [tree(4), tree(5)])))  5  4  5  >>> print\_tree(add\_trees(tree(2, [tree(3)]), tree(2, [tree(3), tree(4)])))  4  6  4  >>> print\_tree(add\_trees(tree(2, [tree(3, [tree(4), tree(5)])]), \  tree(2, [tree(3, [tree(4)]), tree(5)])))  4  6  8  5  5  """  "\*\*\* YOUR CODE HERE \*\*\* |