Tree ADT: exam-level questions

> If you need help reviewing Tree ADT, take a look at these resources:
>
> - Albert's and Robert's slides (https://docs.google.com/presentation/d/1BXO9nB7I-iyzDuoBbvWqo4vUbJqBBhhVlxzOVv9aFAA/edit)

Each question has a "Toggle Solution" button -- click it to reveal that question's solution.

Tree ADT

Recall that we learned two different tree representations in this class:

- Abstract data type (defined in terms of functions). This is what this worksheet covers.
- Object-oriented programming (defined in termes of classes and methods). This will be covered in another review session.

The tree abstract data type is defined in terms of these four functions:

```
def tree(root, subtrees=[]):
    return [root] + list(subtrees)

def root(t):
    return t[0]

def subtrees(t):
    return t[1:]

def is_leaf(t):
    return not subtrees(t)
```

Since this is an ADT, we don't care so much about the implementation of the constructors and selectors: as long as we know what they do, we can use them.

Remember, trees are **recursively defined** (trees are constructed using smaller trees). For most questions involving the tree ADT, you can break down the thought process into three steps:

1. **Base case**: Usually, this is if the tree is a leaf (use the `is_leaf` function)
2. **Recursive call**: Consider what a recursive call on a single branch will do. What information does it give you?

3. **Recursive case**: Make recursive calls on each branch (using a `for` loop or a list comprehension) and combine that in some way with the root for your final answer.

Question 1

Implement a function `contains`, which takes a tree `t` and an element `e`. `contains` will return True if `t` contains the element `e`, and False otherwise.

```
def contains(t, e):
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 2

Implement a function `all_paths`, which takes a tree `t`. `all_paths` will return a list of paths from the root to each leaf. For example, consider the following tree:

```
  5
 / \
 3  6
/ \
2  1
```

Calling `all_paths` on this tree would return

```
[[5, 3, 2],
 [5, 3, 1],
 [5, 6]    ]
```

The list contains three paths (each path is itself a list).

```
def all_paths(t):
    "*** YOUR CODE HERE ***"
```

Toggle Solution

Question 3

Implement a function `max_tree`, which takes a tree `t`. It returns a new tree with the exact same structure as `t`; at each node in the new tree, the entry is the largest number that is contained in that node's subtrees or the corresponding node in `t`. For example, consider this tree:

```
  5
 / \
 3  6
/ \
2  4
```

Calling `max_tree` will return the following tree:

```
  6
 / \
 4  6
/ \
2  4
```

For example, the largest number that occurs at the root or below it is 6 (i.e. `max(5, 3, 2, 4, 6) = 6`), so the root of the new tree is 6.

```
def max_tree(t):
    "*** YOUR CODE HERE ***"
```

Toggle Solution