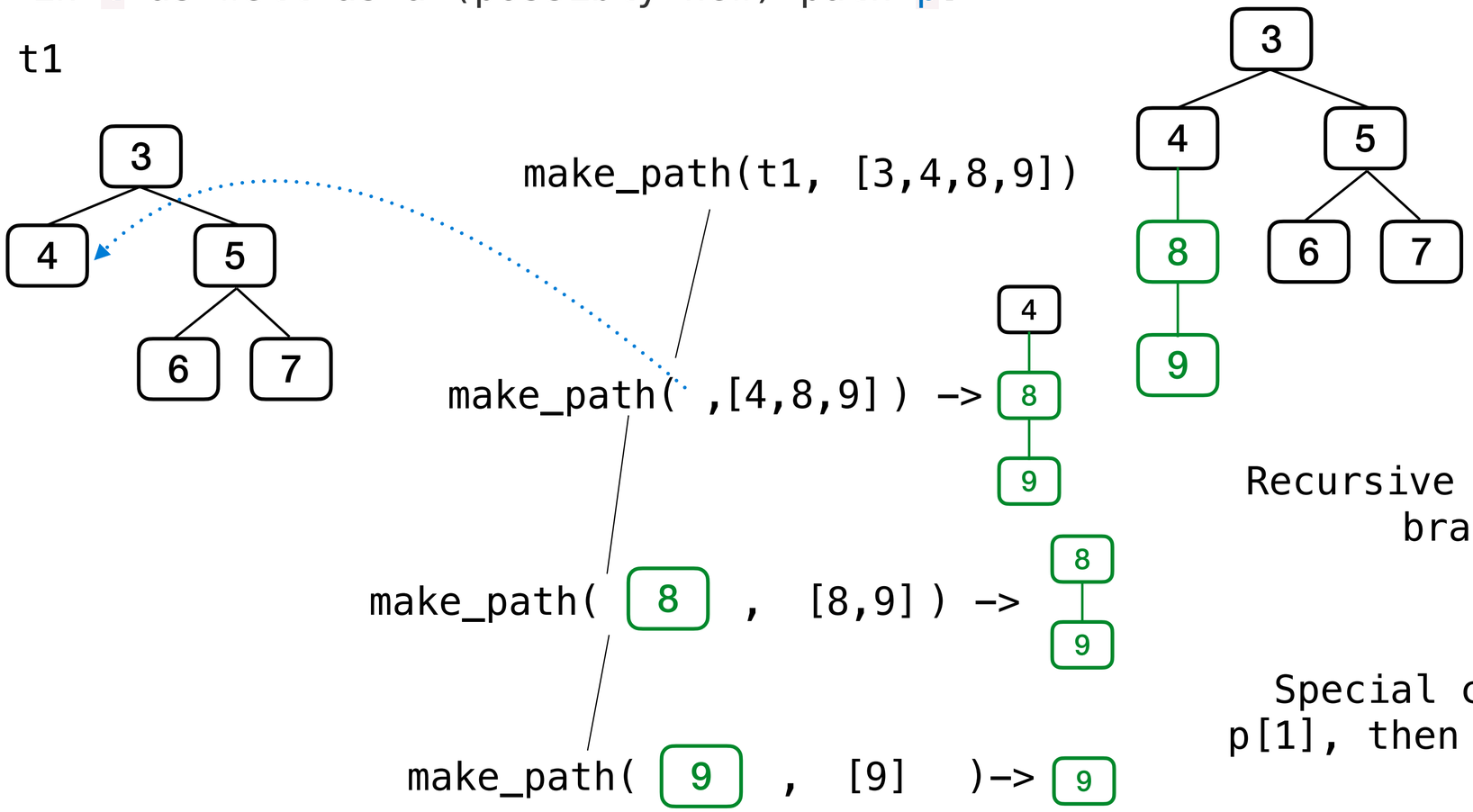
Generators

Announcements

Tree Practice

Example: Make Path

A list describes a path if it contains labels along a path from the root of a tree. Implement `make_path`, which takes a tree `t` with unique labels and a list `p` that starts with the root label of `t`. It returns the tree `u` with the fewest nodes that contains all the paths in `t` as well as a (possibly new) path `p`.



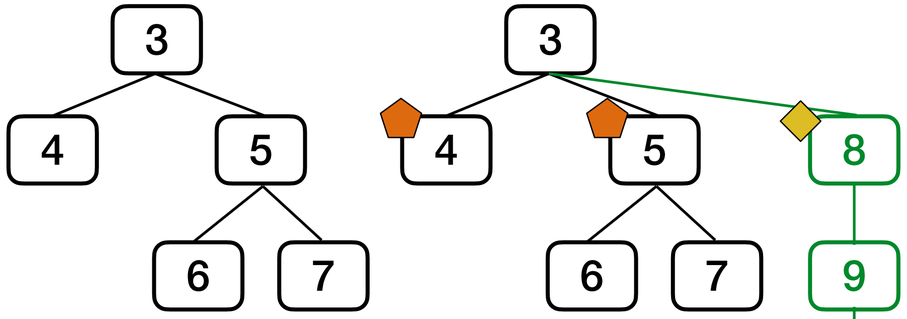
Recursive idea: `make_path(b, p[1:])` is a branch of the tree returned by `make_path(t, p)`

Special case: if no branch starts with p[1], then a leaf labeled p[1] needs to be added

Example: Make Path

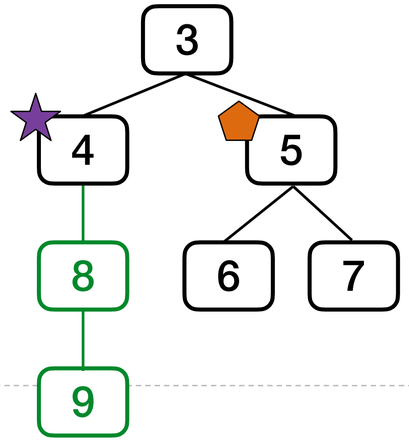
A list describes a path if it contains labels along a path from the root of a tree. Implement `make_path`, which takes a tree `t` with unique labels and a list `p` that starts with the root label of `t`. It returns the tree `u` with the fewest nodes that contains all the paths in `t` as well as a (possibly new) path `p`.

t1



make_path(t1, [3,8,9,1])

make_path(t1, [3,4,8,9])




```
def make_path(t, p):
    "Return a tree like t also containing path p."
    assert p[0] == label(t), 'Impossible'
    if len(p) == 1:
        return t
    new_branches = []
    found_p1 = False
    for b in branches(t):
        if label(b) == p[1]:
            new_branches.append(make_path(b, p[1:]))
            found_p1 = True
        else:
            new_branches.append(b)
    if not found_p1:
        new_branches.append(make_path(tree(p[1]), p[1:]))
    return tree(label(t), new_branches)
```

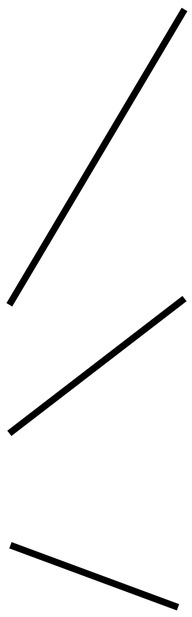
Recursive idea: `make_path(b, p[1:])` is a branch of the tree returned by `make_path(t, p)`

Special case: if no branch starts with p[1], then a leaf labeled p[1] needs to be added

Min Practice

Match the description to the code

```
w = {...}  # a dict with unique keys and values

m = {v: k for k, v in w.items()}

Which expression evaluates to?

1. The key that has the smallest value in w

2. The value that has the smallest key in w

3. The smallest absolute difference between
   a key and its value
```



```
min(w.keys(), key=lambda k: w[k])

min(w.keys(), key=lambda k: m[k])

min(w.values(), key=lambda v: w[v])

min(w.values(), key=lambda v: m[v])

min(w.keys(), key=lambda k: abs(k - w[k]))

min(w.keys(), key=lambda k: abs(k - m[k]))

min(map(lambda k: abs(k - w[k]), w.keys()))

min(map(lambda k: abs(k - m[k]), w.keys()))
```

pollev.com/cs61a

Generators

Generators and Generator Functions

```
>>> def plus_minus(x):
...     yield x
...     yield -x

>>> t = plus_minus(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object plus_minus ...>
```

A *generator function* is a function that **yield**s values instead of **return**ing them

A normal function **return**s once; a *generator function* can **yield** multiple times

A *generator* is an iterator created automatically by calling a *generator function*

When a *generator function* is called, it returns a *generator* that iterates over its yields

(Demo)

Spring 2023 Midterm 2 Question 5(b) Revisited

Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **park**, a **generator function** that yields all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n.

```
def park(n):
    """Yield the ways to park cars and motorcycles in n adjacent spots.

    >>> sorted(park(1))
    ['%', '.']
    >>> sorted(park(2))
    ['%%', '%.', '.%', '..', '<>']
    >>> len(list(park(4)))  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
```

More Tree Practice

Spring 2023 Midterm 2 Question 4(a)

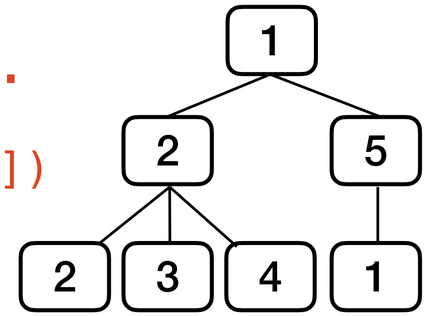
Implement exclude, which takes a tree t and a value x. It returns a tree containing the root node of t as well as each non-root node of t with a label not equal to x. The parent of a node in the result is its nearest ancestor node that is not excluded.

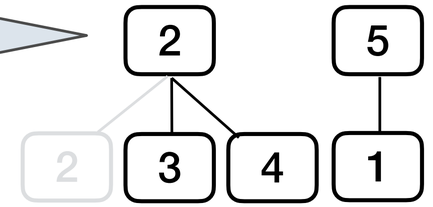
```
def exclude(t, x):
    """Return a tree with the non-root nodes of tree t labeled anything but x.

    >>> t = tree(1, [tree(2, [tree(2), tree(3), tree(4)]), tree(5, [tree(1)])])
    >>> exclude(t, 2)
    [1, [3], [4], [5, [1]]]
    >>> exclude(t, 1)  # The root node cannot be excluded
    [1, [2, [2], [3], [4]], [5]]
    """
    filtered_branches = map(lambda y: ____(____, x), branches(t))
    bs = []
    for b in filtered_branches:
        if ____________:
            bs.____(____)
        else:
            bs.append(b)
    return tree(label(t), bs)
```

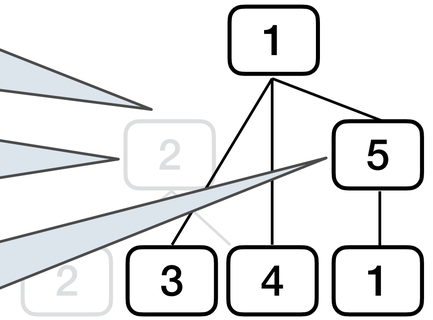
label(b) == x

extend

branches(b)







What will the recursive call on each branch return?

What should we do with those return values?

Branch has label x? Take its branches

Otherwise we're cool with the branch as-is

37% of stud… got this r…

24% got it right

30% got it right; 1 of 4 options