Midterm Examples

Announcements

Trees

From Discussion 5 (Updated to be about the Tree class)

For a Tree instance `t`:

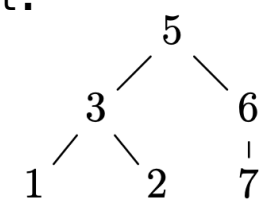
- Its root label can be any value, and `t.label` evaluates to it.
- Its branches are trees, and `t.branches` evaluates to a list of branches.
- It is a leaf if it has no branches, and `t.is_leaf()` returns `True`.
- An identical tree can be constructed with `Tree(t.label, t.branches)`.
- You can call functions that take trees as arguments, such as `height(t)`.
- That's how you work with trees. No `t == x` or `t[0]` or `x in t` or `list(t)`, etc.
- To modify a Tree instance `t`, you can:
  - Change its label: `t.label = ...`
  - Change its branches: `t.branches = ...` or `t.branches.append(...)`
  - Modify one of its branches: `t.branches[0].label = ...`

Fall 2017 CS 61A Midterm 2 Q5(a)

Students received 49% of the points on average.

26% of students answered the question correctly.

Definition. A pile (of leaves) for a tree t with no repeated leaf labels is a dictionary in which the label for each leaf of t is a key, and its value is the path from that leaf to the root. Each path from a node to the root is either an empty tuple, if the node is the root, or a two-element tuple containing the label of the node's parent and the rest of the path (i.e., the path to the root from the node's parent).




```
def pile(t):
    """Return a dict that contains every path from a leaf to the root of tree t.

    >>> pile(Tree(5, [Tree(3, [Tree(1), Tree(2)]), Tree(6, [Tree(7)])]))
    {1: (3, (5, ())), 2: (3, (5, ())), 7: (6, (5, ()))}
    """
    p = {}
    def gather(u, path):
        if u.is_leaf():
            p[u.label] = path
        for b in u.branches:
            gather(b, (u.label, path))
    gather(t, ())
    return p
```

Base case: Put a leaf label in p

Recursive call: Build a longer path

Start at the top

all paths have ()

u has a label that can be added to the path

Fall 2017 CS 61A Midterm 2 Q5(b)

Students received 53% of the points on average.

24% of students answered the question correctly.

Implement Path, a class whose __init__ method takes a Tree t and a leaf_label. Assume all leaf labels of t are unique. When a Path is printed, labels in the path from the root to the leaf of t with label leaf_label are displayed, separated by dashes.

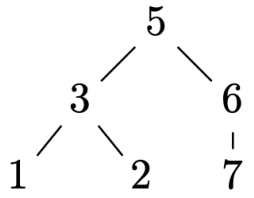
```
class Path:
    """A path through a tree from the root to a leaf, identified by its leaf label.

    >>> a = Tree(5, [Tree(3, [Tree(1), Tree(2)]), Tree(6, [Tree(7)])])
    >>> print(Path(a, 7), Path(a, 2))
    5-6-7 5-3-2
    """
    def __init__(self, t, leaf_label):
        self.pile, self.end = pile(t), leaf_label

    def __str__(self):
        path, s = ____self.pile[self.end]____ , ____str(self.end)____
        while path:
            path, s = ____path[1]____ , ____str(path[0]) + '-' + s____
        return s
```

`{..., 7: (6, (5, ()))}`

`(5, ())`

`()`

Build s from this



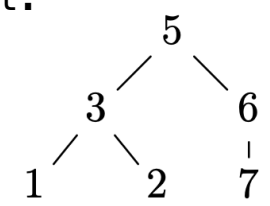
path is a nested tuple

s is a string

Fall 2017 CS 61A Midterm 2 Q5(a) Revisited

Definition. A pile (of leaves) for a tree t with no repeated leaf labels is a dictionary in which the label for each leaf of t is a key, and its value is the path from that leaf to the root. ~~Each path from a node to the root is either an empty tuple, if the node is the root, or a two-element tuple containing the label of the node's parent and the rest of the path (i.e., the path to the root from the node's parent)~~. ***Represent the path as a list of labels.***

```
def pile(t):
    """Return a dict that contains every path from a leaf to the root of tree t.

    >>> pile(Tree(5, [Tree(3, [Tree(1), Tree(2)]), Tree(6, [Tree(7)])]))
    {1: [5, 3, 1], 2: [5, 3, 2], 7: [5, 6, 7]}
    """
    p = {}
    def gather(u, path):
        if u.is_leaf():
            p[u.label] = path + [u.label]
        for b in u.branches:
            gather(b, (u.label, path)) path + [u.label])
    gather(t, ()) [])
    return p
```



OR

```
p[u.label] = path

gather(b, path + [b.label])

gather(t, [t.label])
```

Recursion

From Discussion 4 (With Some Extra Tips)

Don't start trying to write code right away. Instead, start by describing the recursive case in words. Some examples:

- In fib from lecture, the recursive case is to add together the previous two Fibonacci numbers.
- In count_partitions from lecture, the recursive case is to partition **n-m using parts up to size m** and to partition **n using parts up to size m-1.**

How to get the recursive description right?

Use abstraction: Pick an example, then figure out what a recursive call will do for you on that example, not by reading the code, but by reading the docstring.

Implement a choice: Most tree recursion problems involve making a sequence of choices (e.g., use a partition of size m or don't). The recursive case implements one of those choices; recursion implements the rest.

How to get the base cases right?

Once you know what the recursive case is, find all the simple cases it leads to.

Fall 2017 CS 61A Midterm 2 Q4(c)

Students received 71% of the points on average.

41% of students answered the question correctly.

Implement **ways**, which takes two values **start** and **end**, a non-negative integer **k**, and a list of one-argument functions **actions**. It returns the number of ways of choosing functions **f1**, **f2**, ..., **fj** from **actions** such that **f1**(**f2**(...(**fj**(**start**)))) equals **end** and **j** ≤ **k**. The same action function can be chosen multiple times. If a sequence of actions reaches end, then no further actions can be applied (see first example below).

```
def ways(start, end, k, actions):
    """Return the number of ways of reaching end from start by taking up to k actions.

    >>> ways(-1, 1, 5, [abs, lambda x: x+2])   # abs(-1) or -1+2, but not abs(abs(-1))
    2
    >>> ways(1, 10, 5, [lambda x: x+1, lambda x: x+4])  # 1+1+4+4, 1+4+4+1, or 1+4+1+4
    3
    >>> ways(1, 20, 5, [lambda x: x+1, lambda x: x+4])
    0
    >>> ways([3], [2, 3, 2, 3], 4, [lambda x: [2]+x, lambda x: 2*x, lambda x: x[:-1]])
    3
    """
    if start == end:
        return 1
    elif k == 0:
        return 0
    return sum([ways(f(start), end, k - 1, actions) for f in actions])
```

# of ways starting with that action

Choose an action

Fall 2016 CS 61A Midterm 2 Q7(a)

Students received 39% of the points on average.

4% of students answered the question correctly.

Implement sums, which takes two positive integers n and k. It returns a list of lists containing all the ways that a list of k positive integers can sum to n (in any order).

```
def sums(n, k):
    """Return the ways in which K positive integers can sum to N.

    >>> sums(2, 2)
    [[1, 1]]
    >>> sums(2, 3)
    []
    >>> sums(4, 2)
    [[3, 1], [2, 2], [1, 3]]
    >>> sums(5, 3)
    [[3, 1, 1], [2, 2, 1], [1, 3, 1], [2, 1, 2], [1, 2, 2], [1, 1, 3]]
    """
    if ____________________________ k == 1 ____________________________:
        return ____________________ [[n]] ____________________
    y = []
    for x in ____________________ range(1, n) ____________________:
        y.extend([_____ s + [x] _____ for s in sums(_____ n - x _____, _____ k - 1 _____)])
    return y
```

Choose the first number

Build the list

Choose a way

Ways of completing the list

Fall 2016 CS 61A Midterm 2 Q7(b)

Students received 17% of the points on average.

1% of students answered the question correctly.

Note: Nowadays, this question would have been labeled an A+ question and worth 0 points.

Implement sums, which takes two positive integers n and k. It returns a list of lists containing all the ways that a list of k positive integers can sum to n (in any order).

```
f = lambda x, y: (x and [_____[x] + z_____ for z in y] + f(_____x - 1_____, _____y_____)) or []


def sums(n, k):
    """Return the ways in which K positive integers can sum to N."""

    g = lambda w: (w and f(_____n, g(w-1)_____)) or [[]]

    return [v for v in g(k) if sum(v) == n]
```

x: a max number
y: a list of lists of numbers
Put each number up to x at the front of each list in y

Lists of k positive integers