

61A Lecture 18

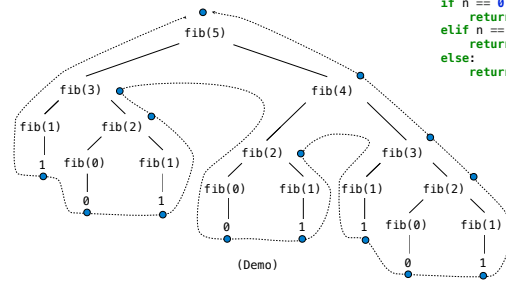
Announcements

Measuring Efficiency

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

```
def fib(n):  
    if n == 0:  
        return 0  
    elif n == 1:  
        return 1  
    else:  
        return fib(n-2) + fib(n-1)
```



<http://en.wikipedia.org/wiki/File:Fibonacci.jpg>

Memoization

Memoization

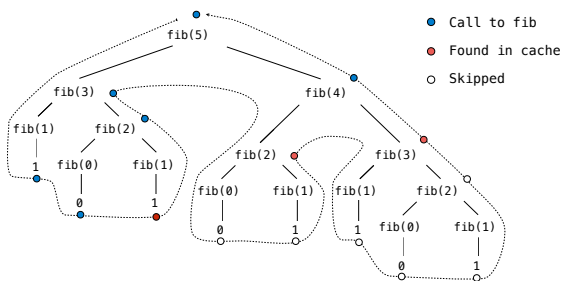
Idea: Remember the results that have been computed before

```
def memo(f):  
    cache = {}  
    def memoized(n):  
        if n not in cache:  
            cache[n] = f(n)  
        return cache[n]  
    return memoized
```

Same behavior as f, if f is a pure function

(Demo)

Memoized Tree Recursion



Tree Class

Tree Class

A Tree has an entry (any value) at its root and a list of branches

```
class Tree:
    def __init__(self, entry, branches=()):
        self.entry = entry
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        return Tree(left.entry + right.entry, (left, right))
```

(Demo)

Hailstone Trees

Hailstone Trees

Pick a positive integer n as the start

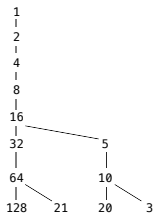
If n is even, divide it by 2

If n is odd, multiply it by 3 and add 1

Continue this process until n is 1

```
def hailstone_tree(k, n=1):
    """Return a Tree in which the paths from the
    leaves to the root are all possible hailstone
    sequences of length k ending in n."""
```

All possible n that start a
length-8 hailstone sequence



(Demo)