Tree Recursion

Order of Recursive Calls

Two Definitions of Cascade

(Demo)

Two Definitions of Cascade

(Demo)

Example: Inverse Cascade

Inverse Cascade

Write a function that prints an inverse cascade:

```
1   def inverse_cascade(n):
12   grow(n)
123  print(n)
1234 shrink(n)
123  def f_then_g(f, g, n):
1  if n:
1     f(n)
1     g(n)

grow = lambda n: f_then_g(  )
shrink = lambda n: f_then_g(  )
```

Tree Recursion

Announcements

The Cascade Function

The Cascade Function

(Demo)
### Tree Recursion

Tree-shaped processes arise whenever executing the body of a recursive function makes more than one recursive call.

- \( n = 0, 1, 2, 3, 4, 5, 6, 7, 8, \ldots \)
- \( \text{fib}(n) = 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots \)

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

- A Tree-Recursive Process
  - The computational process of \( \text{fib} \) evolves into a tree structure

#### Repetition in Tree-Recursive Computation

This process is highly repetitive; \( \text{fib} \) is called on the same argument multiple times.

- \( \text{fib}(5) \)
  - \( \text{fib}(3) \)
  - \( \text{fib}(1) \)
  - \( \text{fib}(2) \)
  - \( \text{fib}(0) \)

(We will speed up this computation dramatically in a few weeks by remembering results)

### Counting Partitions

- The number of partitions of a positive integer \( n \), using parts up to size \( m \), is the number of ways in which \( n \) can be expressed as the sum of positive integer parts up to \( m \) in increasing order.

#### Example: Counting Partitions

- \( \text{count_partitions}(6, 4) \)
  - \( 2 + 4 = 6 \)
  - \( 1 + 1 + 4 = 6 \)
  - \( 3 + 3 = 6 \)
  - \( 1 + 2 + 3 = 6 \)
  - \( 1 + 1 + 1 + 3 = 6 \)
  - \( 2 + 2 + 2 = 6 \)
  - \( 1 + 1 + 2 + 2 = 6 \)
  - \( 1 + 1 + 1 + 1 + 2 = 6 \)
  - \( 1 + 1 + 1 + 1 + 1 + 1 = 6 \)

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don’t use any 4
- Solve two simpler problems:
  - \( \text{count_partitions}(2, 4) \)
  - \( \text{count_partitions}(6, 3) \)
- Tree recursion often involves exploring different choices.

```python
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m
```

(We will speed up this computation dramatically in a few weeks by remembering results)