Tree Recursion

Tree Recursion

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

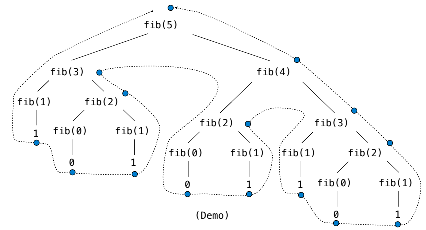
n:	0,	1,	2,	3,	4,	5,	6,	7,	8,	... ,	35
fib(n):	0,	1,	1,	2,	3,	5,	8,	13,	21,	... ,	9,227,465

```
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

A Tree-Recursive Process

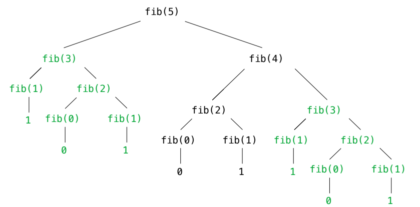
The computational process of fib evolves into a tree structure



(Demo)

Repetition in Tree-Recursive Computation

This process is highly repetitive; fib is called on the same argument multiple times



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

Total # of ways to get to (M, N)

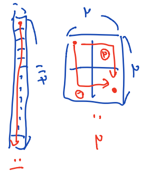
= Total # of ways to get to (M-1, N) +

Total # of ways to get to (M, N-1)

took 1 step towards M

took 1 step towards N

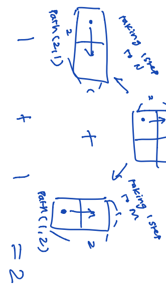
```
def path (m, n):
    """Return the number of paths from one corner of an
    M by N grid to the opposite corner.

    >>> path (2, 2)
    2
    >>> path (5, 7)
    210
    >>> path (117, 1)
    1
    >>> path (1, 157)
    1
    """
    if m==1 or n==1:
        return 1
    return path(m-1, n) + path(m, n-1)
```



Total # of ways to get to (M, N)

= Total # of ways to get to (M-1, N) +

Total # of ways to get to (M, N-1)



n = 689   k = 16

n // 10   n % 10

```
def knap(n, k):
    if n == 0:
        return k == 0
    with_last = knap(n//10, k - n % 10)
    without_last = knap(n//10, k)
    return with_last or without_last
```

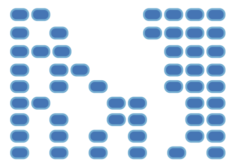
Example: Counting Partitions

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.

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

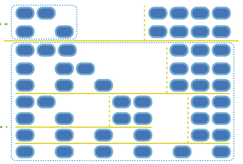


Counting Partitions

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 non-decreasing order.

count_partitions(6, 4)

- 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:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.



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.

- 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:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.

```
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
```

(Demo)

all_nums

k=3 →

0
1
10
11
100
101
110
111

h(2,0)

h(1,1)

h(3,0)

h(2,1)

```
def all_nums(k):
    def h(k, prefix):
        if k == 0:
            print(prefix)
            return
        h(k-1, prefix * 10)
        h(k-1, prefix * 10 + 1)
    h(k, 0)
```

1100
1101
1110
1111