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

fib(n): 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots, 9,227,465

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 fib evolves into a tree structure.

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.)
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.

\[
\begin{align*}
2 + 4 &= 6 \\
1 + 1 + 4 &= 6 \\
3 + 3 &= 6 \\
1 + 2 + 3 &= 6 \\
2 + 2 + 2 &= 6 \\
1 + 1 + 3 + 2 &= 6 \\
1 + 1 + 1 + 4 &= 6 \\
1 + 1 + 1 + 3 + 1 &= 6
\end{align*}
\]

count_partitions(6, 4)

Example: Counting Partitions

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_last = count_partitions(n//10, k - n % 10)
        without_last = count_partitions(n//10, k)
        return with_last or without_last

(result)

```python
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        with_last = count_partitions(n//10, k - n % 10)
        without_last = count_partitions(n//10, k)
        return with_last or without_last

def all_nums(k):
    if k == 0:
        return
    all_nums(k-1)
    k(k-1)
    k(k-1) + 10
```

```text
# 1 + 1 + 3 + 2 = 6
# 1 + 1 + 1 + 4 = 6
# 1 + 1 + 1 + 3 + 1 = 6
# 2 + 2 + 2 = 6
# 1 + 1 + 1 + 1 + 2 = 6
# 1 + 1 + 1 + 1 + 1 + 1 = 6
```