Tree Recursion

Tree-shaped processes arise whenever executing the body of a function entails making more than one call to that function.

```
```

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

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

```
```

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
```

Memoized Tree Recursion

```
Demo
```

```
fib(6)
```

```
fib(4)
```

```
fib(2)
```

```
fib(1)
```

```
fib(0)
```

```
fib(3)
```

```
fib(2)
```

```
fib(1)
```

```
fib(0)
```

```
fib(4)
```

```
fib(2)
```

```
fib(1)
```

```
fib(0)
```

```
fib(6)
```

```
fib(5)
```

```
fib(3)
```

```
fib(2)
```

```
fib(1)
```

```
fib(0)
```

```
fib(4)
```

```
fib(2)
```

```
fib(1)
```

```
fib(0)
```

```
fib(6)
```

```
fib(5)
```

```
fib(3)
```

```
fib(2)
```

```
fib(1)
```

```
fib(0)
```

```
fib(4)
```

```
fib(2)
```

```
fib(1)
```

```
fib(0)
```

```
fib(35)
```

```
35
```

```
18,454,929
```

```
Call to fib
Found in cache
```

```
Call to fib with memoization:
35
Calls to fib without memoization: 18,454,929
```
Iteration vs Memoized Tree Recursion

Iterative and memoized implementations are not the same.

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>n steps</td>
<td>n entries</td>
</tr>
</tbody>
</table>

The first Fibonacci number

```python
def fib_iter(n):
    prev, curr = 1, 0
    for _ in range(n-1):
        prev, curr = curr, prev + curr
    return curr
```

@memo
def fib(n):
    if n == 1:
        return 0
    if n == 2:
        return 1
    return fib(n-2) + fib(n-1)
```

Counting Change

How many ways are there to change a dollar?

How many ways to change $0.11 with nickels & pennies?

$0.11 can be changed with nickels & pennies by

A. Not using any more nickels; $0.11 with just pennies
B. Using at least one nickel; $0.06 with nickels & pennies

Space Consumption

Which environment frames do we need to keep during evaluation?
Each step of evaluation has a set of active environments.
Values and frames referenced by active environments are kept.
Memory used for other values & frames can be reclaimed.

Active environments:
- The environment for the current expression being evaluated
- All environments for expressions that depend upon the value of the current expression
- All environments associated with values referenced by active environments

Counting Change Recursively

```python
def count_change(a, kinds=(50, 25, 10, 5, 1)):
    # base cases
    d = kinds[0]
    return count_change(a, kinds[1:]) + count_change(a-d, kinds)
```

Fibonacci Environment Diagram

```
Fib: 1 2 3 ...
    ↓ fib
n: 3 fib(3)
    ↓
0

Fib(0): 0 1 1 ...
    ↓ fib
n: 1 fib(1)
    ↓
1

Fib(2): 1 1 2 ...
    ↓ fib
n: 2 fib(2)
    ↓
1 1

Fib(1): 0 1 1 ...
    ↓ fib
n: 1 fib(1)
    ↓
1

fib: 1 1 2 ...
    ↓ fib
n: 3 fib(3)
    ↓
1 1 2

fib(n): 1 1 2 ...
    ↓ fib
n: 3 fib(3)
    ↓
1 1 2
```

Fibonacci Environment Diagram

```
Fib: 1 2 3 ...
    ↓ fib
n: 3 fib(3)
    ↓
0

Fib(0): 0 1 1 ...
    ↓ fib
n: 1 fib(1)
    ↓
1

Fib(2): 1 1 2 ...
    ↓ fib
n: 2 fib(2)
    ↓
1 1

Fib(1): 0 1 1 ...
    ↓ fib
n: 1 fib(1)
    ↓
1

fib: 1 1 2 ...
    ↓ fib
n: 3 fib(3)
    ↓
1 1 2

fib(n): 1 1 2 ...
    ↓ fib
n: 2 fib(2)
    ↓
1 1
```

```
fib(n): 1 1 2 ...
    ↓ fib
n: 3 fib(3)
    ↓
1 1 2
```
Fibonacci Memory Consumption

- fib(6)
  - fib(4)
    - fib(2)
      - fib(1)
    - fib(3)
  - fib(5)
    - fib(4)
    - fib(3)

Assume we have reached this step

Active Environments for Returned Functions

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add1 = make_adder(1)
```

Therefore, all frames in this environment must be kept.

Associated with an environment

Has an active environment
Can be reclaimed
Hasn't yet been created