61A Lecture 18
Announcements
Measuring Efficiency
Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:
Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```
Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

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)
```

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

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)
```

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)
```

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

\[
\begin{align*}
\text{def } & \quad \text{fib(n):} \\
\text{if } & \quad n == 0: \\
\quad & \quad \text{return 0} \\
\text{elif } & \quad n == 1: \\
\quad & \quad \text{return 1} \\
\text{else:} \\
\quad & \quad \text{return fib(n-2) + fib(n-1)}
\end{align*}
\]

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

[Diagram of recursive computation of the Fibonacci sequence with code snippet]
Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

[Diagram of recursive computation of Fibonacci sequence]

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

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)
```

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)
```

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)
```

Our first example of tree recursion:

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

![Fibonacci.jpg](http://en.wikipedia.org/wiki/File:Fibonacci.jpg)
Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

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)
```

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)
```

[Diagram of recursive computation of Fibonacci sequence]

Recursive Computation of the Fibonacci Sequence

Our first example of tree recursion:

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

![Fibonacci Tree](http://en.wikipedia.org/wiki/File:Fibonacci.jpg)
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)
```

Memoization
Memoization

**Idea:** Remember the results that have been computed before
Memoization

**Idea:** Remember the results that have been computed before

```python
def memo(f):
```

```python
def memo(f):
```
Memoization

**Idea:** Remember the results that have been computed before

```python
def memo(f):
    cache = {}
Memoization

**Idea:** Remember the results that have been computed before

```python
def memo(f):
    cache = {}
    def memoized(n):
```
Memoization

**Idea:** Remember the results that have been computed before

```python
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
```
Memoization

**Idea:** Remember the results that have been computed before

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

**Idea:** Remember the results that have been computed before

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

**Idea:** Remember the results that have been computed before

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

**Idea:** Remember the results that have been computed before

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

Keys are arguments that map to return values
Memoization

Idea: Remember the results that have been computed before

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

Keys are arguments that map to return values

Same behavior as f, if f is a pure function
Memoization

**Idea:** Remember the results that have been computed before

```python
def memo(f):
    cache = {}

def memoized(n):
    if n not in cache:
        cache[n] = f(n)
    return cache[n]

return memoized
```

- **Keys are arguments that map to return values**
- **Same behavior as f, if f is a pure function**

(Demo)
Memoized Tree Recursion

fib(5)
  /      /
fib(3)   fib(4)
  /  
fib(1) fib(2)
    / 
   1 fib(0) fib(1)

fib(3)
  /  
fib(1) fib(2)
    / 
   1 fib(0) fib(1)

fib(4)
  /  
fib(2) fib(3)
    / 
   0 fib(0) fib(1)

fib(2)
  /  
fib(0) fib(1)
    / 
   0 1
Memoized Tree Recursion

Call to fib

fib(5)

fib(3)
  fib(1)
    1
  fib(2)
    fib(0)
      0
    fib(1)
      1

fib(4)
  fib(2)
    fib(0)
      0
    fib(1)
      1

fib(3)
  fib(1)
    fib(0)
      0
    fib(1)
      1

fib(2)
  fib(0)
    0
  fib(1)
    1

fib(1)
  fib(0)
    0
  fib(1)
    1

fib(0)
  1
Memoized Tree Recursion

Call to fib

Found in cache
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

Call to fib
Found in cache
Skipped
Memoized Tree Recursion

Call to fib
Found in cache
Skipped

1
0
1
fib(0)
fib(1)
fib(2)
fib(3)
fib(4)
fib(5)

0
1
fib(0)
fib(1)
fib(2)
fib(3)
fib(4)
fib(5)

0
1
fib(0)
fib(1)
fib(2)
fib(3)
fib(4)
fib(5)

0
1
fib(0)
fib(1)
fib(2)
fib(3)
fib(4)
fib(5)
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped

```
Call to fib(5)
  Call to fib(3)
    Call to fib(1)
    fib(0) = 0
    fib(1) = 1
    Call to fib(2)
      Call to fib(0)
      fib(0) = 0
      fib(1) = 1
  fib(2) = 1
  Call to fib(4)
    Call to fib(2)
      Call to fib(0)
      fib(0) = 0
      fib(1) = 1
    fib(1) = 1
    Call to fib(3)
      Call to fib(1)
      fib(0) = 0
      fib(1) = 1
  fib(3) = 1
```

```
1
Call to fib(1)
fib(0) = 0
fib(1) = 1

1
Call to fib(2)
fib(0) = 0
fib(1) = 1

1
```

```
1
```

```
1
```

```
1
```

```
1
```

```
1
```
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

Call to fib

Found in cache

Skipped
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

- Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped

fib(5)
  fib(3)
    fib(1)
      1
    fib(0)
      0
  fib(2)
    fib(1)
      1
    fib(0)
      0

fib(4)
  fib(2)
    fib(1)
      1
    fib(0)
      0
  fib(3)
    fib(1)
      1
    fib(0)
      0

fib(3)
  fib(1)
    fib(0)
      0
    fib(1)
      1
  fib(2)
    fib(0)
      0
    fib(1)
      1

fib(4)
  fib(2)
    fib(1)
      1
    fib(0)
      0
  fib(3)
    fib(1)
      1
    fib(0)
      0

fib(5)
  fib(3)
    fib(1)
      1
    fib(0)
      0
  fib(2)
    fib(1)
      1
    fib(0)
      0
Memoized Tree Recursion

- Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped
Memoized Tree Recursion

- **Call to `fib`**
- **Found in cache**
- **Skipped**
Memoized Tree Recursion

- Blue circle: Call to \texttt{fib}
- Red circle: Found in cache
- White circle: Skipped

```
Call to \texttt{fib}
\textbf{fib}(5)
- \textbf{fib}(3)
  - \textbf{fib}(1)
    - \textbf{fib}(0): 0
    - \textbf{fib}(1): 1
  - \textbf{fib}(2)
    - \textbf{fib}(0): 0
    - \textbf{fib}(1): 1
- \textbf{fib}(4)
  - \textbf{fib}(2)
    - \textbf{fib}(0): 0
    - \textbf{fib}(1): 1
  - \textbf{fib}(3)
    - \textbf{fib}(0): 1
    - \textbf{fib}(1): 1
```
Memoized Tree Recursion

Call to fib
- Found in cache
- Skipped
Tree Class
Tree Class

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

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

class Tree:
Tree Class

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

class Tree:
    def __init__(self, entry, branches=[]):
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
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)
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)

Built-in `isinstance` function: returns True if `branch` has a class that is or inherits from `Tree`
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)
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):
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)
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)

Built-in `isinstance` function: returns True if `branch` has a class that is or inherits from `Tree`
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))
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
Hailstone Trees

Pick a positive integer \( n \) as the start
Hailstone Trees

Pick a positive integer \( n \) as the start

If \( n \) is even, divide it by 2
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
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
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
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
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

1
2
4
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

\[ 1 \]
\[ 2 \]
\[ 4 \]
\[ 8 \]
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

1
2
4
8
16
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
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
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

1

2

4

8

16

32

64

128
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

1
  | 2
  | 4
  | 8
  | 16
  | 32
  | 64
  | 128
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
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
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
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
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

All possible $n$ that start a length-8 hailstone sequence

1 2 4 8 16 32 64 128 21 20 3
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

```python
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
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

All possible n that start a length-8 hailstone sequence

```python
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."""
```

(Demo)