Announcements

• Project 1 is due Thursday 9/19 @ 11:59pm

• Midterm 1 is on Monday 9/23 from 7pm to 9pm
  • 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  • HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  • Extra office hours over the weekend
  • Includes topics up to and including this lecture
  • Fill out the form on the website if you cannot attend

• Homework 3 is due in two weeks: Tuesday 10/1 @ 11:59pm
  • It contains lots of recursion problems, for practice!

• Optional Hog strategy contest ends Thursday 10/3 @ 11:59pm
Hog Contest Rules

• Up to two people submit one entry; Max of one entry per person.
• Your score is the number of entries against which you win more than 50% of the time.
• All strategies must be deterministic, pure functions of the current player scores! Non-deterministic strategies will be disqualified.
• One more special rule: Ham Hijinks. Choose -1 to swap the 4-sided and 6-sided dice.
• To enter: submit proj1contest with a file hog.py that defines a final_strategy function by Thursday 10/3 @ 11:59pm
• All winning entries will receive 2 points of extra credit
• The real prize: honor and glory

Fall 2011 Winners
Keegan Mann,
Yan Duan & Ziming Li,
Brian Prike & Zhenghao Qian,
Parker Schuh & Robert Chatham

Fall 2012 Winners
Chenyang Yuan,
Joseph Hui

Fall 2013 Winners
YOUR NAME COULD BE HERE...
FOREVER!

http://inst.eecs.berkeley.edu/~cs61a/fa13/proj/hog_contest/hog_contest.html
Order of Recursive Calls
The Cascade Function

```python
def cascade(n):
    if n < 10:
        print(n)
    else:
        print(n)
cascade(n//10)
cascade(123)
```

Program output:
```
123
12
1
12
```

(Demo)

- Each `cascade` frame is from a different call to `cascade`.
- Until the Return value appears, that call has not completed.
- Any statement can appear before or after the recursive call.

Example: [http://goo.gl/090qzK](http://goo.gl/090qzK)
Two Definitions of Cascade

(Demo)

```python
def cascade(n):
    if n < 10:
        print(n)
        print(n)
    else:
        print(n)
        cascade(n//10)
        print(n)

def cascade(n):
    print(n)
    if n >= 10:
        cascade(n//10)
        print(n)
```

- If two implementations are equally clear, then shorter is usually better.
- In this case, the longer implementation is more clear (at least to me).
- When learning to write recursive functions, put the base cases first.
- Both are recursive functions, even though only the first has typical structure.
Tree Recursion
Tree Recursion

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

\[
\begin{align*}
n & : 1, 2, 3, 4, 5, 6, 7, 8, 9, \ldots, 35 \\
fib(n) & : 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots, 5,702,887 \\
\end{align*}
\]

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

A Tree-Recursive Process

The computational process of \texttt{fib} evolves into a tree structure

![Diagram showing the tree-recursive process of the Fibonacci sequence]
Repetition in Tree-Recursive Computation

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

We can speed up this computation dramatically in a few weeks by remembering results.
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.

\[
\text{partition}(6, 4)
\]

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

\[
\text{partition}(6, 4)
\]

- 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:
  - \( \text{partition}(2, 4) \)
  - \( \text{partition}(6, 3) \)
- 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:
  - partition(2, 4)
  - partition(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
```

(Demo)

Winning Hog
How to Win at Hog

What is the chance that I'll score at least $k$ points rolling $n$ six-sided dice?

\[
\text{Number of ways to score at least } k
\]
\[
\text{Number of possible rolls}
\]

The number of possible rolls is $\text{pow}(6, n)$.

The number of ways to score at least $k$ in $n$ rolls can be computed using tree recursion!

Sum over each possible dice outcome $d$ that does not \text{pig out}: the number of ways to score at least $k - d$ points using $n - 1$ rolls.

\text{Base case:} The number of ways to score at least 0 is $\text{pow}(5, n)$.

\text{Base case:} The number of ways to score positive points in 0 rolls is 0.