Discussion 03

Recursion, Tree Recursion :)

Aditya Balasubramanian
aditbala [at] berkeley [dot] edu

Slides available at `teaching.aditbala.com`

Announcements 

- Homework 1 due today (6/30)
- Lab 2 due today (6/30)
- Hog Checkpoint due tommorow (7/1)
- The tuition refund deadline to drop any session C summer course is July 1st.
  - There's also still lots of room in CS 10, if you're looking for a different pace than 61A—you can enroll in the class, and then reach out to cs10@berkeley.edu and they'll help catch you up.

Agenda

- Mini Lecture - Recursion
- `Q2` walkthrough
- `Q1`
- `Q3` / `Q4` / `Q5`
- Mini Lecture - Tree Recursion
- `Q7`
- `Q8`

Recursion

An Exciting Metaphor

- The Problem
  - Want to find how many dolls are inside this doll
- What we need to do
  - Create a function that we can repeat until there is no more dolls to count
- Ideas?

Solution

- Remove one layer at a time and one to total, stop when there is no more dolls and add one

Dish Washing Example

INTERACTIVE !

- 20 dishes to wash on a Sunday morning

Recursion

- What is a recursive function?
  - A function that calls itself
  - Returns a function call of itself, not the object (different than HOF)
- Recursive Leap of Faith
  - The idea that the recursive function will work no matter what/how many test cases are passed in

Solution in Formal Terms of Recursion

- Base Case
  - smallest problem with guranteed answer, or smallest input
  - **A doll with no other dolls inside of it**
  - **One dish left**
- Recursive Call
  - A method of reducing the current problem into a smaller problem
  - **Removing each layer one by one and adding it to a total**
  - **Making a clone to wash dishes**

Recursion vs Iteration

- Recursion
  - Make problem smaller
  - Variables get reset
  - Many frames will open
    - Lot of memory taken up
  - Better for some problems
    - Recursive Data Structures (Trees, Linked Lists)
- Iteration
  - Loops happen in one frame
  - Easy to visualize
  - No additional function calls

Q2: Recursion Environment Diagram

Draw an environment diagram for the following code:

```
def rec(x, y):
    if y > 0:
        return x * rec(x, y - 1)
    return 1

rec(3, 2)
```

Q1: Recursive Multiplication (7 min)

Write a function that takes two numbers m and n (only positive) and returns their product. Use recursion, not `mul` or `*`

```
def multiply(m, n):
    """ Takes two positive integers and returns
    their product using recursion.
    >>> multiply(5, 3)
    15
    """
    "*** YOUR CODE HERE ***"
```

Q3: Find the Bug (3 min)

Find the bug with this recursive function.

```
def skip_mul(n):
    """Return the product of n * (n - 2) * (n - 4) * ...
    >>> skip_mul(5) # 5 * 3 * 1
    15
    >>> skip_mul(8) # 8 * 6 * 4 * 2
    384
    """
    if n == 2:
        return 2
    else:
        return n * skip_mul(n - 2)
```

Choose your own adventure !!!

`Q3` , `Q4` , `Q5`

Tree Recursion

Tree Recursion

- What is Tree Recursion?
  - Recursion, but with more recursive calls
  - Can break down the problem in more than one way
  - With all of the options drawn out, looks like a tree of recursive calls
- When and Why?
  - Useful when the original problem can be broken down in multiple ways
  - Accumulate all sub-problems with multiple recursive calls

Recursive Fibonacci

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

- Need to look at `fib(n - 1)` and `fib(n-1)`
- All steps of recursion present
  - Base Case
  - Recursive Calls
  - Applying to solve problem

Q7: Count Stair Ways

How many different ways are there to go up a flight of stairs with `n = 1` step? How about `n = 2` steps? Try writing out some other examples and see if you notice any patterns.

```
def count_stair_ways(n):
    """Returns the number of ways to climb up a flight of
    n stairs, moving either 1 step or 2 steps at a time.
    >>> count_stair_ways(4)
    5
    """
    "*** YOUR CODE HERE ***"
```

Q8: Count K

Consider a special version of the `count_stair_ways` problem, where instead of taking `1` or `2` steps, we are able to take up to and including `k` steps at a time. Write a function `count_k` that figures out the number of paths for this scenario. Assume `n` and `k` are positive.

```
def count_k(n, k):
    """
    >>> count_k(3, 3) # 3, 2 + 1, 1 + 2, 1 + 1 + 1
    4
    >>> count_k(4, 4)
    8
    >>> count_k(10, 3)
    274
    >>> count_k(300, 1) # Only one step at a time
    1
    """
```

Thank you!

Attendance Form -> https://tinyurl.com/adit-disc03

Anon Feedback -> https://tinyurl.com/adit-anon

Study Groups -> https://tinyurl.com/adit-study