61A Lecture 20

Friday, October 17
Announcements

• Guerrilla Section 4 on Sunday 10/19: Object-oriented programming and recursive data
  • Meet in 271 Soda: Vanguard section from 12–2pm; Main section from 2:30–4:30pm
• Homework 6 is due Monday 10/20 @ 11:59pm
  • Homework party on Monday 10/20 6pm–8pm in 2050 VLSB
• Project 3 is due Thursday 10/23 @ 11:59pm
• Midterm 2 is on Monday 10/27 7pm–9pm
  • Class Conflict? Fill out the conflict form at the top of http://cs61a.org
  • Review session on Saturday 10/25 3pm–6pm in 2050 VLSB
• CSUA and Hackers@Berkeley are holding a hack-a-thon on Saturday for 61A students
  • 10am – 11pm in Wozniak Lounge
Introducing Cohorts

Each of you has been randomly placed in the cohort of a patron computer scientist.

00: Ada Lovelace
Wrote first program

10: George Boole
Invented boolean logic

01: Haskell Curry
Math for functional programming

11: Grace Hopper
Wrote first compiler
Measuring Efficiency
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](http://en.wikipedia.org/wiki/File:Fibonacci.jpg)
Memoization
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

Call to fib
- Found in cache
- Skipped
Linked List Class
Linked Lists as Objects

Linked list idea: Pairs are sufficient to represent sequences of arbitrary length

1 2 3 4

Data abstraction (old way):

```python
>>> s = link(1, link(2, link(3, link(4, empty))))
>>> len_link(s)
4
>>> getitem_link(s, 2)
3
>>> s
[1, [2, [3, [4, 'empty']]]]
```

Link class (new way):

```python
>>> s = Link(1, Link(2, Link(3, Link(4))))
>>> len(s)
4
>>> s[2]
3
>>> s
Link(1, Link(2, Link(3, Link(4))))
```
Linked List Class

Linked list class: pairs are two-attribute objects

class Link:
    empty = ()
    # Some zero length sequence

def __init__(self, first, rest=empty):
    self.first = first
    self.rest = rest

def __getitem__(self, i):
    if i == 0:
        return self.first
    else:
        return self.rest[i-1]

def __len__(self):
    return 1 + len(self.rest)

More special method names:

__getitem__ Element selection []
__len__ Built-in len function

Methods can be recursive too!

(Demo)
Tree Class
**Tree Class**

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

```python
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)
Example: 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 starting $n$ that give an 8-number-long hailstone sequence

(Demo)