Recursion

Frédo Durand and Ana Bell
MIT EECS, 6.00

Warning: CS’s favorite joke topic

https://xkcd.com/754/
World of world of warcraft

Announcements

• No lecture on Thursday
  – Quiz in the evening

• Fredo’s office hours
  – Mondays 4–5pm in D424
Towers of Hanoi - intro
Towers of Hanoi

- 3 rods, N disks of increasing size

- Goal: move tower from one rod to another one
  - Only one disk can be moved at a time.
  - Each move takes the upper disk from one of the stacks and places it on top of another stack
    - i.e. a disk can only be moved if it is the top disk on a stack.
  - No disk may be placed on top of a smaller disk
Recursion
Recursion

- describing things
- solving problems
  divide & conquer
Recursion

• way of describing, defining, or specifying things.

• way of designing solutions to problems (divide and conquer)
Birthright US Citizenship

• Any one born inside the United States

• Any one born outside the United States, both of whose parents are citizens of the U.S., as long as one parent has lived in the U.S.

http://en.wikipedia.org/wiki/Birthright_citizenship_in_the_United_States
Recursion

- Base case
  - direct solution (always one or more)

- Inductive case
  - solution in terms of solution to the same problem on simpler inputs
Recursion

- Base case
  - tells us directly what the answer is

- Inductive case
  that defines the answer in terms of the answer to the same question on some other input
  - typically to a simpler version of the same problem
Factorial
Recursive factorial

\[ N! = N \times (N-1)! \]

1! = 1

def fact(N):
    if N == 1:
        return 1  # base case
    else:
        return N \times \text{fact}(N-1)  # recursive call
Recursive factorial

def fact(N):
    if N==1:
        return 1
    return N * fact(N-1)

print fact(5)
Online Python tutor, scope

```python
1  def fact(N):
2       if N==1:
3           return 1
4       return N * fact(N-1)
5
6  print fact(5)
```

Frames

- Global frame
  - fact

Objects

- function fact(N)
  - fact
    - N = 5
  - fact
    - N = 4
  - fact
    - N = 3
  - fact
    - N = 2
  - fact
    - N = 1
Non-recursive version

def fact(N):
    f = 1
    for i in xrange(2,N+1):
        f = f*i
    return f

print fact(5)
Recursion vs. non-recursion

- recursive: simplest
  - rarely the most efficient
  
  programmer time is most precious

From computer perspective
Recursion vs. non-recursion

• Disclaimer:
  Some of the early examples we study are too simple to show benefits

• Recursion:
  – often simpler to implement
  – rarely the most efficient
  – can always be done non-recursively

  – but programmer productivity is paramount
Recursion
archetype
Simple Recursion
Simple Recursion

def recursive_function(x, other param):
    if x is simple_enough:
        return simple_answer
    else:
        return some_operation(
            recursive_function(
                smaller(x) , other param) )
Base case
Base case

• There must always be a base case
• Otherwise, infinite recursion
I WANT TO BUILD A PERFECT HO-SCALE (1/87) MODEL TRAIN LAYOUT OF MY TOWN.
IN YOUR BASEMENT? BAD IDEA. NEVER MAKE A LAYOUT OF THE AREA YOU'RE IN.

WHY NOT?
BECAUSE IT'D INCLUDE A LITTLE 10" REPLICA OF YOUR HOUSE.

SO? THAT'D BE COOL! I'D MAKE TINY REPLICA OF MY ROOMS, MY FURNITURE—
AND YOUR TRAIN LAYOUT?

MY GOD.

Yeah, it's the second rule of model train layouts: No nesting.

...What's the first rule?

'Do not talk about model train layouts.' That rule was actually voted in by our friends and families.

Philistines.

http://xkcd.com/878/
Recursion

• Often good idea to start with the simplest: base case
  – For one thing, you won’t forget it
  – Also less intimidating

• Then figure out the recursion
Recursion on lists
Recursive product

- Given input list of numbers, compute product of all elements
  - Yes, we don’t need recursion, but it’s a good exercise

\[ [1, 2, 3] \]

base case: list w/ 1 element
Slicing lists
List slicing

- $L[\text{begin:end}]$ returns a COPY of the sublist between element begin and end-1
List slicing

- \( L[\text{begin:end}] \) returns a COPY of the sublist between element begin and end-1
- Recall \( L1 = L[:] \) copies the full list
- Similarly \( L1 = L[1:] \) copies the sublist from the second element to the end
- \( L1 = L[:\text{end}] \) copies the sublist from the beginning to end-1
Recursive product
Recursive product

\[ L = [1, 3, 4, 3, 6, 11] \]

```python
def product(L):
    if len(L) == 0:
        return 1
    return L[0]*product(L[1:])

print(product(L))
```
Binary search
Find element in ordered list

\[ [1, 3, 5, 6, 8] \]

is 4 in list?

base case: 0 or 1 element
def binary_search(L, x):
    print L for debugging
    if len(L) == 1:
        return x == L[0]
    mid = len(L) / 2
    if L[mid] <= x:
        return binary_search(L[mid:], x)
    else:
        return binary_search(L[:mid], x)
def binary_search(L, x):
    print L for debugging
    if len(L) == 1:
        return x == L[0]
    mid = len(L) / 2
    if L[mid] <= x:
        return binary_search(L[mid:], x)
    else:
        return binary_search(L[:mid], x)
Recursive structures
Family tree
Tournament

Winner

Winner of Semi

Winner of Semi
Koch snowflake

http://en.wikipedia.org/wiki/Koch_snowflake
Koch snowflake

http://en.wikipedia.org/wiki/Koch_snowflake
See also

Fibonacci &
double-recursion
Mating rabbits
Mating rabbits

• You begin with one male rabbit and one female rabbit. These rabbits have just been born.

• A rabbit will reach sexual maturity after one month.

• The gestation period of a rabbit is one month.

• Once it has reached sexual maturity, a female rabbit will give birth every month.

• A female rabbit will always give birth to one male rabbit and one female rabbit.

• Rabbits never die.
Example
Abstraction
Abstraction

• All the rabbits from last month are still alive
• The rabbits from the month before give birth to one pair of rabbits
• Recurrence:
  \[ F(N) = F(N-1) + F(N-2) \]
• More complex base case:
  – both 0 and 1 must be specified
Recursive Fibonacci
Recursive Fibonacci

def fib(N):
    if N == 0:
        return 0
    if N == 1:
        return 1
    return fib(N-1) + fib(N-2)
Merging ordered lists
Merge

- Given two ordered lists
- return merged ordered list

\[
\begin{array}{c}
\left[1, 2, 4, 7\right] \\
\left[3, 5, 7\right]
\end{array}
\rightarrow \left[1, 2, 3, 4, 5, 7\right]
\]

Base case: 1 empty list
Merge
def merge(L1, L2):
    if len(L1) == 0: return L2
    if len(L2) == 0: return L1
    if L1[0] < L2[0]:
        return [L1[0]] + merge(L1[1:], L2)
    else:
        return [L2[0]] + merge(L1, L2[1:])

print merge(L1, L2)
Warning

• Computer scientists love recursion Jokes

  e.g. Linux means: Linux is not UniX

Towers of Hanoi - recursion
Recursive towers of Hanoi
Recursive towers of Hanoi

• Base case:
  – move smallest ring from A to B
    (no constraint, always possible directly)

• Induction.
  goal: move N rings from rod A to B
    – move N–1 smaller rings to rod C
    – move big ring to rod B
    – move N–1 smaller rod from C to B