From Recursion to Self Reference

public int calc(int n){
    return n*calc(n-1);
}

● What is the Internet?
  ➢ A network of networks.
  Or ...
● What is recursive DNS?
  ➢ What IP is fxyzt4.com?
● What is PageRank?
  ➢ Where is it used?

PFTD, PFTW, PFYL

● Structure motivates ‘self-referential’ code
  ➢ motivation
● Basic understanding of recursion
  ➢ Principles
  ➢ Examples
● APTs with recursion, setup for Boggle/Trees

Why Linked Lists and Recursion are ...

● Node has
  ➢ Information
  ➢ Pointer to node

● Recursive method
  ➢ Does NOT call self
  ➢ Does call "clone"

● Self-referential
  ➢ You talking to me?

● What is 'this' in code?
  ➢ Self-referential

● Methods call other methods (often?)
  ➢ How to remember where to return?
  ➢ Abstraction!

Quota Exceeded: coping with storage

● You're running out of disk space
  ➢ Buy more
  ➢ Compress files
  ➢ Delete files

● How do you find your "big" files?
  ➢ What's big?
  ➢ How do you do this?
BlobCount or edge detection or ...

- How do we find images? Components? Paths?
  - Create information from data

Tools: Solving Computational Problems

- Algorithmic techniques and paradigms
  - Brute-force/exhaustive, greedy algorithms, dynamic programming, divide-and-conquer, ...
  - Transcend a particular language
  - Designing algorithms, may change when turned into code

- Programming techniques and paradigms
  - Recursion, memo-izing, compute-once/lookup, tables, ...
  - Transcend a particular language
  - Help in making code work
    - Cope with correctness and maintenance
    - Cope with performance problems

Tools: Solving Computational Problems

- Java techniques
  - java.util.*, Comparator, LinkedList, Map, Set, ...
  - These aren’t really Java-specific, but realized in Java
  - Map, Comparator, Set: C++, Python, ...
  - We learn idioms in a language and talk about abstractions

- Analysis of algorithms and code
  - Mathematical analysis, empirical analysis
  - We need a language and techniques for discussion
  - Theory and practice, real problems and in-the-limit issues

- “In theory there is no difference between theory and practice, but in practice there is.”
  (attributed to many)
Solving Problems Recursively

- **Recursion:** indispensable in programmer’s toolkit
  - Elegance can lead to better programs: easier to modify, extend, verify, more efficient, cure...
  - Sometimes recursion isn’t appropriate, when it’s bad it can be very bad—every tool requires knowledge and experience in how to use it

- The basic idea is to get help solving a problem from coworkers (clones) who work and act like you do
  - Ask clone to solve a simpler/smaller, but similar problem
  - Use clone’s result to put together your answer

- **Both:** call on the clone and use the result

Exponentiation

- **Computing** \(x^n\) means multiplying \(n\) numbers
  - Does it require \(n\) multiplies?
  - What’s the simplest value of \(n\) when computing \(x^n\)?
  - To only multiply once, what can you ask a clone?

```java
public static double power(double x, int n){
    if (n == 0){
        return 1.0;
    }
    return x * power(x, n-1);
}
```

- **Number of multiplications? Structure?**
  - Note base case: no recursion, no clones
  - Note recursive call: moves toward base case (unless …)

Faster exponentiation

- **Recursive calls made to compute** \(2^{1024}\)
  - How many multiplies on each call? Is this better?

```java
public static double power(double x, int n){
    if (n == 0) return 1.0;
    double semi = power(x, n/2);
    if (n % 2 == 0) return semi*semi;
    return x * semi * semi;
}
```

- **What about an iterative version of this function?**
  - Why might we want such a version?

Back to Recursion

- **Recursive functions have two key attributes**
  - There is a **base case**, aka exit case: no recursion!
    - See print directories, exponentiation
  - All other cases make a recursive call, with some measure (e.g., parameter value) that decreases towards the base case
    - Ensure that sequence of calls eventually reaches the base case
    - “Measure” can be tricky, but usually it’s straightforward

- **Example: structural recursion: data meets code**
  - Why is directory code inherently recursive?
  - How is this different from exponentiation?
More recursion recognition

```java
public static int sumit(int[] a, int index){
    if (index < a.length) {
        return a[index] + sumit(a,index+1);
    }
    return 0;
}
```

// original call:  int v = sumit(a,0);

- What is base case, what value is returned?
- How is progress towards base case realized?
- How is recursive call used to return a value?
- What if we sum values in a linked list?

Recursive methods sometimes use extra parameters; helper methods set this up.

Fran Allen

- IBM Fellow, Turing Award
  - Optimizing compilers
- Taught high school for two years, then Master’s degree and IBM
  - Teachers excited me to learn

I've always felt that theory without practice is maybe nice and maybe pretty, but it’s not going to influence computing as much as the practice side. But the practice has to be backed up with the ability to talk about it, reason about it, and formulate it so that it can be reproduced.

Blob Counting, Flood Fill

- Flood a region with color
  - Erase region, make transparent, ..
  - How do find the region?
- Finding regions, blobs, edges, ..
  - See blob counting code
  - What is a blob?
- Recursion helps, but necessary?
  - Performance, clarity, ...
  - Ease of development

Ideas behind blob fill code

- Ask your neighbors
  - Return blob size
  - Ensure no re-counts
  - Sum and return
- What do neighbors do?
  - Same thing!
  - Colors indicate calls
Details and Idioms in blob code

- **Method blobFill** has four parameters
  - (row,column) of where search starts
  - Character being searched for (initially * or blob)
  - Character to fill with on success (e.g., count '2' or '4')
    - Mark for visualization
    - Mark to ensure we don't search again!
- If (row,column) is part of blob, count it and ask neighbors for their counts
  - They're part of blob (if never visited before)
- Return total of yourself and neighbors
  - Key to recursion: do one thing and ask for help

Blob questions

- What changes if diagonal cells are adjacent?
  - Conceptually and in code
- How do we find blob sizes in a range?
  - Not bigger than X, but between X and Y
- How would we number blobs by size rather than by when they're found?
  - Do we have the tools to do this in existing code?
- Can we avoid recursion and do this iteratively?

SpreadingNews APT: greedy, recursive

![SpreadingNews diagram]

Ideas for coding SpreadingNews

- Assume three subordinates: 5 min, 4 min, 7 min
  - Who do we tell first? Second?
  - How long does it take us to finish?
  - Suppose subordinates take 3 min, 3 min, 3 min; then what?
- How do we found out how long it takes for sub?
  - Do we have a way of doing this?
  - Is there some way out of always asking again?
- Putting ideas into code: greedy and recursion!
  - Where is base case?
**Word Ladder APT**

- From->[words]->to
  - From hit to cog via [hot,dot,lot,dog,log]
- What words reachable from 'from'?
  - Repeat until we get to 'cog'
- Problem: reachable from 'dot'
  - Why not include 'hot'?
  - Don't re-use words
- Algorithm:
  - Find all words 1-away
  - From each n-away find (n+1)-away

**Digression: word ladders**

- How many ladders from cart to dire as shown?
  - Enqueue dare more than once?
  - Downside? Alternative?
- We want to know number of ladders that end at W.
  - What do we know initially?
  - When we put something on the queue, what do we know?
  - How do we keep track?
- Initialize and update per-word statistics

**Word Ladder: more details**

- # ladders that end at dare
  - At each word W
- Ladder length to W
  - Calculable from??
- Two maps

**Dequeue s**

- foreach W one-away
- if not-seen ???
- else ???

**Alan Kay**

- Turing award 2003
  - OO programming, Dynabook
- “The best way to predict the future is to invent it”
- “American’s have no past and no future, they live in an extended present.”

I think the main thing about doing ...any kind of programming work, is that there has to be some exquisite blend between beauty and practicality. There’s no reason to sacrifice either one of those, and people who are willing to sacrifice either one of those, I don’t think really get what computing is all about.
### Standard list processing (recursive)

- Visit all nodes once, e.g., count them
  ```java
  public int recsize(Node list) {
    if (list == null) return 0;
    return 1 + recsize(list.next);
  }
  ```

- Base case is almost always empty list: null pointer
  - Must return correct value, perform correct action
  - Recursive calls use this value/state to anchor recursion
  - Sometimes one node list also used, two "base" cases

- Recursive calls make progress towards base case
  - Almost always using `list.next` as argument

### Linked list Practice

- What is a list? Empty or not: mirrored in code
  ```java
  public Node copy(Node list) {
    if (null == list) return null;
    Node first = new Node(list.info,null);
    first.next = copy(list.next);
    return first;
  }
  ```

- How can we replace last three lines with one?
  - `return new Node(list.info,copy(list.next));`
  - When constructing a list, make sure to assign to `.next` field!

- What about iterative version? Issues? Advantages?

### Recursion with pictures

- Counting recursively
  ```java
  int recsize(Node list){
    if (list == null) return 0;
    return 1 + recsize(list.next);
  }
  ```

- System.out.println(recsize(ptr));

### Programming with Nodes/Linked Lists

- When adding or removing nodes
  - Be sure you alter `.next` field: re-assign or call new
  ```java
  list.next = new Node() OR tmp OR recursiveCall
  ```

- Using iteration: keep pointer to first AND current
  - Allow iteration over list, but must keep pointer to front
  - Sometimes call new before looping to have a Node
    - e.g., invariant add to `.next` field in loop
    - Return `temp.next` as needed

- Recursion is often simpler than iteration
  - Code mirrors structure of data!