Recursion revisited

Csc 115 Fall 2002

Exercise:

- Consider for a few mins: Given a string, how would you generate all possible permutations of it?

Recursive algorithms and data structures

- A method (algorithm) or class that is partially defined in terms of itself is called recursive – i.e. it calls itself
- Recursion is a powerful algorithm design and programming tool that can lead to elegant and efficient algorithms and data structures
- Any algorithm that can be solved with recursion can also be solved without recursion

Recursive algorithms and data structures

- A recursive algorithm consists of
  - ✍ a base case
  - ✍ a recursive call (with smaller or simpler arguments)
- Very important to ensure that the recursion terminates… that the base case is always reached for any input

Recursive methods and classes

- A recursive method is a method that directly or indirectly calls itself
- Simplest direct and indirect recursive methods; note that both examples result in infinite recursion since there is no base case

  void a() { a(); } // direct recursion

  void a() { b(); } void b() { a(); } // indirect
- Shortest and simplest direct and indirect recursive classes

  class X { X x; } // direct recursion

  class X { Y y; } **class** Y { X x; } // indirect

Recursive and Iterative algorithms

```
public static void writeStars(int n)
// iterative function that produces an output line
  // of exactly n stars
{
  for (int i = 0; i < n; i++)
    System.out.print("*");
  System.out.println();
}

public static void writeStars2(int n)
 // recursive function that produces an output
  // line of exactly n stars
{
  if (n <= 0)
    System.out.println();
  else {
    System.out.print("*");
    writeStars2(n - 1);
  }
}
```

Recursion vs. Iteration

Just because we can use recursion doesn't mean we should,
    requires overhead of extra method calls

HOWEVER....

some algorithms are easy to express recursively,
    hard to express iteratively
in those cases, recursion is a very useful tool
use when it's natural to define the big problem in terms of
    smaller versions of the same problem

trade-off: efficiency vs. simplicity
note: recursion adds constant factors –
    doesn't change basic complexity

Interactive examples and exercises in Eclipse….

1) *Stars.java*

2) *Reverse.java*

Example: Factorials

For positive integers n, n! ("n factorial") is

1 * 2 * 3 * ... * n

Recursive definition (recurrence relation):

1! = 1

if n > 1, n! = n * (n-1)!

To compute 4!:

```
4! = 4 * 3!
   = 4 * 3 * 2!
   = 4 * 3 * 2 * 1!
   = 4 * 3 * 2 * 1
   = 24
```

What does this do?

```
public int mystery(int n) {

  if (n == 1) {    if (n <= 1)
    return(1);
  }
  else {
    return n + mystery(n - 2);
  }
}
```

Mystery.java

Debugging recursive methods

- Draw a stack trace and execute it by hand
- Use lots of println statements
- Use a debugger
- Try boundary cases when testing

Fibonacci sequence

1 1 2 3 5 8 ……

Each number in the sequence is the addition of the previous two numbers – how do we solve this iteratively and recursively?

Example: Fibonacci

- Iteration:

```
f0=1;
f1=1;
for (i=2; i<=n; i++) {
        f=f1+f1;
        f0=f1;
        f1=f;
}
```

- Recursion

```
int F(int n) {
if (n==0) return 1;
    else if (n==1) return 1;
    else return F(n-1)+F(n-2);
}
```

Example: Permutations

- Given a string s, print out all possible permutations of the letters in s: void permute(String s)

- Simple idea: For each possible first letter in s, calculate the permutations of the remaining letters

- For this example, we need to create a helper method that is more general than permute(s):
  - permute(string prefix, string s)

- This example makes use of both recursion and iteration (similar to the treemap exercise earlier in the term)

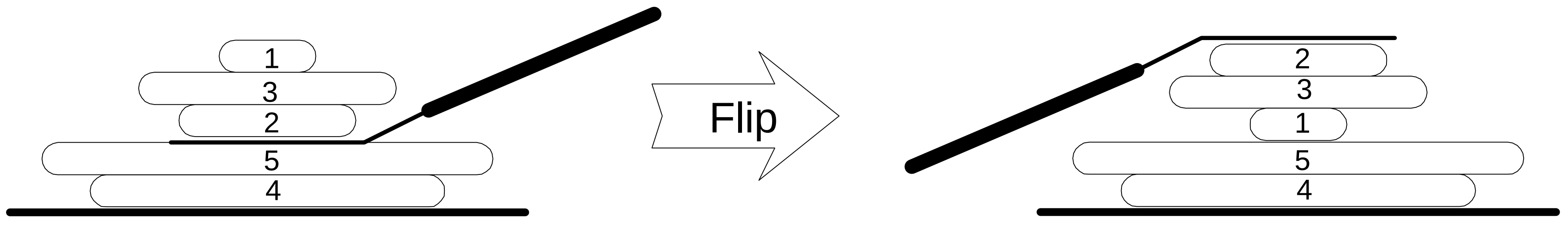
Interactive examples and exercises in Eclipse….

Permute.java

Swedish pancake problem

- Class exercise -- **Pancake sort**
- You are a cook in the Swedish Restaurant – Pannkakslandet -, and you have a stack of *n* pancakes to serve to the customers. However, since all the pancakes have different diameters ranging from 1 to *n*, and are stacked up on the plate in a random order, the stack doesn't look very tidy. Of course you want to present the plate to the customers with the pancakes stacked in a tidy pyramid, so that the maple syrup can flow evenly over the edges of each pancake.
- You have only one kind of operation that you can perform: that is to stick the spatula into the stack at some position, and to flip the top substack of pancakes upside-down. For example, if there are 5 pancakes ordered as (1,3,2,5,4) from the top down, you might insert the spatula beneath the pancake of size 2 and flip the top three pancakes, resulting in order (2,3,1,5,4)

Pancake problem



Analyze our solution?

- What is the runtime complexity of our solution in the worst case? Best case?

Recurrence relations recap.

Analysis of pancake sort?

```
T(1) = O(1)
T(n) = n + T(n-1) + b, n > 1
```

this is a "recursive" definition of a mathematical function f (defined for all n >= 1)

this recurrence relation has a "closed form" solution:

$$f(n) ? \frac{n?n ? 1?}{2}$$

But not all recurrence relations have such easy solutions!

Our approach here is very informal – don’t prove these recurrence relations until csc 225

Tower of Hanoi

The Tower of Hanoi (sometimes referred to as the Tower of Brahma or the End of the World Puzzle) was invented by the French mathematician, Edouard Lucas, in 1883. He was inspired by a legend that tells of a Hindu temple where the pyramid puzzle might have been used for the mental discipline of young priests. Legend says that at the beginning of time the priests in the temple were given a stack of 64 gold disks, each one a little smaller than the one beneath it. Their assignment was to transfer the 64 disks from one of the three poles to another, with one important proviso–a large disk could never be placed on top of a smaller one. The priests worked very efficiently, day and night. When they finished their work, the myth said, the temple would crumble into dust and the world would vanish.

Temple Pura Ulu Danau, Bali

Towers of Hanoi

Move disks from one post to another.
Rules:

- move one disk at a time
- never have a disk on top of a smaller disk

- Some applets:
  - ✍ http://www.mazeworks.com/hanoi/
  - ✍ http://www.simtools.com/ToH.html

Towers of Hanoi program

- This program solves the Towers of Hanoi puzzle.
- You specify how many disks to put initially on tower A and the program will tell you a series of moves to get them all to tower B.
- E.g. Number of disks? 3

```
Number of disks? 3
Move disk from A to B
Move disk from A to C
Move disk from B to C
Move disk from A to B
Move disk from C to A
Move disk from C to B
Move disk from A to B
```

Hanoi Algorithm

To move n disks from disk 1 to disk 3:

- move the top n-1 disks from post 1 to post 2
- move the last (largest) disk from post 1 to post 3
- move the n-1 disks from post 2 to post 3

Base case: if n = 1, just move the disk from post 1 to post 3.

Must generalize our goal: move n disks from any post to any other post.
There will always be an extra post to use for temporary storage.

Code in Eclipse

Tower.java

Analysis of this program

let $T(n)$ = time to execute tower with numdisks = n.
$T(1)$ = some constant a

if $n > 1$: call tower twice with numdisks = n-1, plus
a comparison and some printing (constant time)

so $T(n) = 2T(n-1) + b$ when $n > 1$, for some constant b

$T(1) = a$
$T(n) = 2T(n-1) + b, n > 1$

Unroll:

$T(n) = 2T(n-1) + b$
$= 2[2T(n-2) + b] + b = 4T(n-2) + 3b$
$= 4[2T(n-3) + b] + 3b = 8T(n-3) + 7b$
$= 16T(n-4) + 15b$ *anyone see a pattern yet?*

Analysis (2)

after k steps:

$T(n) = 2^k T(n-k) + (2^k-1)b$

after n-1 steps:

$T(n) = 2^{n-1}T(1) + (2^{n-1}-1)b = 2^{n-1}(a+b) - b = O(2^n)$

So our Tower of Hanoi problem has *exponential* complexity

Note $2^{64} = 18446744073709551616$

Summary

- Recursion is fun!
- http://javaboutique.internet.com/particleTree/