Recursion subproblems

CS125 Spring 2007

Arthur Kantor
LOOK! A NEW SNOW GOON!

THAT'S NOT THE ONE I MADE!

THE ORIGINAL SNOW GOON MUST BE MAKING HIS OWN SNOW GOONS!

OH NO!

I'LL BET HE'S MAKING AN ARMY? IN A FEW DAYS, HE COULD BUILD A HUNDRED SNOW GOONS! IF EACH OF THEM BUILT ANOTHER HUNDRED, AND THEN THOSE ALL BUILT A HUNDRED MORE, WHY . . .

...THAT WOULD BE PRETTY COOL, IF THEY WEREN'T OUT TO KILL ME.

I VOTE WE MAKE TRACKS FOR FLORIDA.
Applying recursion

• To apply recursion to a problem you have to:
  – Find a sub-problem which is self-similar to the large problem
  – Make the size of the problem a parameter to your recursive function (let's call it `recFunc(size)`)  
  – Solve the base case of the problem and explicitly write the answer into `recFunc()`  
  – Assume that `recFunc(size-1)` returns the correct answer to a sub-problem of size-1.
  – Use the output of `recFunc(size-1)` to write the solution to problem `recFunc(size)`
# Finding a sub-problem

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible sub-problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>takes an integer as an argument</td>
<td>takes a smaller integer as an argument</td>
</tr>
<tr>
<td>E.g. factorial(n)</td>
<td></td>
</tr>
<tr>
<td>takes an array as an argument</td>
<td>takes a sub-array as an argument</td>
</tr>
<tr>
<td>E.g. prod(list)</td>
<td></td>
</tr>
<tr>
<td>draw a picture</td>
<td>draw a smaller picture</td>
</tr>
<tr>
<td>counting paths through some network</td>
<td>count paths through a smaller network</td>
</tr>
</tbody>
</table>
Example: Pow

- int Pow(int base, int exp) returns base raised to the power exp: base^{exp}
  - Pow(2,4) returns 16
Pow(int base, int exp)

- Subproblem:
Pow(int base, int exp)

• Subproblem:
  – Easier to solve if the exp is smaller
• Size-of-problem parameter:
  – exp (already a parameter of Pow)
• Base case:
Pow(int base, int exp)

• Subproblem:
  – Easier to solve if the exp is smaller

• Size-of-problem parameter:
  – exp (already a parameter of Pow)

• Base case:
  – Pow(base,0) returns 1 for any value of base

• General case:
Pow(int base, int exp)

• Subproblem:
  – Easier to solve if the exp is smaller

• Size-of-problem parameter:
  – exp (already a parameter of Pow)

• Base case:
  – Pow(base,0) returns 1 for any value of base

• General case:
  – Pow(base,exp)=base*Pow(base,exp-1)
public static int pow(int base, int exp)
{
    if (exp == 0)
        return 1;
    else // exp > 0
        return base * pow(base, exp - 1);
}
Example: Palindrome

- A palindrome is a word or a phrase that reads the same forwards and backwards
  - “racecar”, “Step on no pets”
  - “Kay, a red nude, peeped under a yak”
- (ignore the spaces and punctuation)
boolean palindrome(char[] str)

• The function should return true if str is a palindrome and false otherwise

• Subproblem:
boolean palindrome(char[] str)

• Subproblem:
  – Consider the string without the first and last letters: str[1]…str[str.length-2]

• Size-of-problem parameters
boolean palindrome(char[] str, int start, int stop)

• **Subproblem:**
  – Consider the string without the first and last letters: str[1]…str[str.length-2]

• **Size-of-problem parameters**
  – Start position and stop position
  – Alternately, create a new char[] array without the first and last characters

• **Base case(s):**
boolean palindrome(char[] str, int start, int stop)

• Subproblem:
  – Consider the string without the first and last letters: \text{str}[1]...\text{str}[\text{str}.length-2]

• Size-of-problem parameters
  – Start position and stop position

• Base case(s)
  – Return \text{true} if \text{stop-start}<2.

• General case:
boolean palindrome(char[] str, int start, int stop)

• Subproblem:
  – Consider the string without the first and last letters:
    str[1]…str[str.length-2]

• Size-of-problem parameters
  – Start position and stop position

• Base case(s)
  – Return true if stop–start<2.

• General case:
  – Return true if
    str[start]==str[stop] && palindrome(str, start+1, stop–1)
public boolean palindrome(char[] str, int start, int stop)
{
    if (stop-start<2)
        return true;
    else
        return str[start]==str[stop] &&
               palindrome(str, start+1, stop-1);
}
Example: Tower of Hanoi

- The goal is to move all N discs from the left peg to the right one.
- Only one disc may be moved at a time.
- A disc can be placed either on an empty peg or on top of a larger disc.
Example: Tower of Hanoi

• The goal is to move all N discs from the left peg to the right one.
• Only one disc may be moved at a time.
• A disc can be placed either on an empty peg or on top of a larger disc. (Demo)
A recursive solution to Tower of Hanoi

• Problem
  – move N rings from peg i to peg j
• Subproblem
A recursive solution to Tower of Hanoi

• Problem
  – move N rings from peg i to peg j

• Subproblem
  – Moving a shorter tower of N-1 rings is easier and seems helpful

• Size-of-problem parameters
A recursive solution to Tower of Hanoi

• Problem
  – move N rings from peg i to peg j

• Subproblem
  – Moving a shorter tower of N-1 rings is easier and seems helpful

• Size-of-problem parameters
  – Number of rings

• Base case
A recursive solution to Tower of Hanoi

• Problem
  – move N rings from peg i to peg j

• Subproblem
  – Moving a shorter tower of N-1 rings is easier and seems helpful

• Size-of-problem parameters
  – Number of rings

• Base case
  – One ring (N=1) is trivial -- just move it where you need it

• General case
Tower of Hanoi – general case

- Assume you can move N-1 rings from peg i to peg j
Tower of Hanoi – general case

- Assume you can move N-1 rings from peg i to peg j
- Move top N-1 rings to the peg that is not i and not j (call it k)
Tower of Hanoi – general case

- Assume you can move N-1 rings from peg i to peg j
- Move top N-1 rings to the peg that is not i and not j (call it k)
- Move Nth ring to peg j
Tower of Hanoi – general case

• Assume you can move N-1 rings from peg i to peg j
• Move top N-1 rings to the peg that is not i and not j (call it k)
• Move Nth ring to peg j
• Move the N-1 rings from k to j
Tower of Hanoi – general case

- Assume you can move N-1 rings from peg i to peg j
- Move top N-1 rings to the peg that is not i and not j (call it k)
- Move Nth ring to peg j
- Move the N-1 rings from k to j
- You’ve now moved N rings from peg i to peg j
Tower of Hanoi – divide and conquer

- The problem seems daunting at first
- By assuming that a similar sub-problem is solved, we find a way to solve it one step further
- That’s all that is required to solve a problem of any size
Tower of Hanoi

- No java code given!
- We did not talk about how the rings and the pegs are represented
  - Are they arrays? Objects? Something else?
- But the algorithm description is precise enough so that you *could* implement it

Think about how you would implement it in Java