Intro to Recursion

May 12, 2014
CS/IT115 Spring 2014
Idea of a procedure

- “A procedure is a pattern for the local evolution of a computational process. It specifies how each stage of the process is built upon the previous stage.” - Abelson and Sussman, SICP, p. 31

- “Procedure” here means something like “algorithm” (which in Java means “method”)

- Key is to be able to visualize simple procedures
Linear Recursion

• Factorial function:

\[ n! = n \times (n - 1) \times (n - 2) \times \ldots \times 3 \times 2 \times 1 \]

• Notice that \( n! \equiv n \times (n - 1)! \)

\[ n! = n \times [(n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1] = n \times (n-1)! \]

• So, we can compute \( n! \) by computing \((n-1)!\) and multiplying the result by \( n \).

• Start with simple rule that \( 1! \equiv 1 \)
Computing Factorial

Using rules above, we can create this simple, linearly recursive method to compute n!

```java
public static int factorial(int n) {
    if (n == 1)
        return 1;
    else
        return n * factorial(n - 1);
}
```
Factorial in Action

factorial(4)
4 * factorial(3)
4 * (3 * factorial(2))
4 * (3 * (2 * factorial(1)))
4 * (3 * (2 * 1))
4 * (3 * 2)
4 * 6
24
Deferred Operations

- This visualization reveals a shape of expansion, followed by contraction.
- The expansion occurs as the method builds up a chain of deferred operations (in this case, multiplications).
- The contraction occurs as the operations are actually performed (evaluated).
- This type of process is called a recursive process.
Recursive Processes

• Executing this method requires that the JVM (interpreter) keep track of the operations to be performed later on.

• For the factorial function, the length of the chain of deferred multiplications, and hence the amount of information needed to keep track of it, grows linearly with n.

• This method is called a linear recursive process
Tree Recursion

• Another pattern of recursive computation.
• Consider the computation of the Fibonacci numbers – each number is the sum of the preceding two:
  – 0, 1, 1, 2, 3, 5, 8, 13, 21, …
• In general, Fib(n) =
  0 if n == 0
  1 if n == 1
  Fib(n-1) + Fib(n-2) otherwise
Recursive Method for Fib

```java
public static int fib(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```
Pattern of Fibonacci Computation

- To compute fib(5), we need to compute fib(4) and fib(3)
- To compute fib(4), we need to compute fib(3) (which we already had to compute once above) and fib(2), and so on for fib(3)...fib(0)
- In general, the evolved method execution looks like a tree, with branches splitting into two at each level (except at the bottom).
- This reflects that fib calls itself twice each time it is invoked.
Execution Tree for Fib
This method is inefficient

- Notice that the entire computation of $\text{fib}(3)$ is duplicated (about half the total work).
- $\text{fib}(1)$ and $\text{fib}(0)$ get computed $\text{fib}(n+1)$ times!
- This method uses a number of steps that grows exponentially with the size of the input.
- The space (memory) required grows only linearly, b/c we only need to keep track of the nodes above us in the tree at any point.
Iterative fibonacci method

• Can use an iterative process that still makes a recursive call, but whose computation takes place iteratively (that is, it takes a number of steps linear in the size of input)

• Idea is to use pair of ints a and b, initialized to fib(1) == 1 and fib(0) == 0, and to repeatedly apply the transformations:
  - a = a + b
  - b = a
private static int fibHelper(int a, int b, int count) {
    if (count == 0)
        return b;
    return fibHelper(a + b, a, count - 1);
}

public static int fib(int n) {
    return fibHelper(1, 0, n)
}
Iterative Version of fib

- The iterative version takes far fewer steps (linear instead of exponential), which is a big difference even for small inputs.
- Not all tree-recursive methods are useless – they can help us understand and design programs. Our first version of fib is much simpler than the iterative version.