COMP 250

Lecture 10

recursive algorithms

Sept. 28, 2016
Example 1: Factorial (iterative)

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

```plaintext
factorial( n ){  // assume n >= 1
    result = 1
    for (k = 2; k <= n; k++)
        result = result * k
    return result
}
```
Factorial (recursive)

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

```java
factorial(n) {
    // assume n >= 1
    if (n == 1)
        return 1;
    else
        return factorial(n - 1) * n;
}
```
Claim: the recursive factorial algorithm is correct.

Proof by mathematical induction:

Base case: \texttt{factorial(1)} returns 1.

Induction step:

\begin{verbatim}
for k >= 1
    if \texttt{factorial(k)} returns k!
    then \texttt{factorial(k+1)} returns (k+1)!
\end{verbatim}
Example 2: Fibonacci

\[0, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots\]

\[Fib(0) = 0\]
\[Fib(1) = 1\]
\[Fib(n + 2) = Fib(n + 1) + Fib(n), \text{ for } n \geq 0.\]
Fibonacci (iterative)

Fib(n){
    if ((n == 0) | (n == 1))
        return n
    else{
        fib0 = 0
        fib1 = 1
        for k = 2 to n{
            fib2 = fib1 + fib0   // Fib(n+2)
            fib0 = fib1         // Fib(n) in next pass
            fib1 = fib2         // Fib(n+1) in next pass
        }
        return fib2
    }
}
Fibonacci (recursive)

Fib(n) {  // assume n > 0
    if ((n == 0) || (n == 1))
        return n
    else
        return Fib(n-1) + Fib(n-2)
}

This is much simpler to express than the iterative version.
Claim: the recursive Fibonacci algorithm is correct.

Proof:

Base case: \[ \text{Fib}(0) \text{ returns } 0. \quad \text{Fib}(1) \text{ returns } 1. \]

Induction step:

\[
\text{for } k > 1 \\
\quad \text{if } \text{Fib}(k-1) \& \text{Fib}(k) \text{ returns the Fibonacci numbers } k-1 \& k \\
\quad \text{then } \text{Fib}(k+1) \text{ returns Fibonacci number } k+1.
\]
However, the recursive Fibonacci algorithm is very inefficient. It computes the same quantity many times, for example:
Example 3: Reversing a list

input (a b c d e f g h)

output (h g f e d c b a)

Idea of recursion:

a (b c d e f g h)

(h g f e d c b) a
Example 3: Reversing a list (recursive)

reverse(list) {
    // assume n > 0
    if list.size == 1 // base case
        return list
    else{
        firstElement = list.removeFirst()
        list = reverse(list) // list has only n-1 elements
        return list.addLast(firstElement)
    }
}
Example 4: Sorting a list (recursive)

// this is similar to selection sort

sort(list) {
    if list.size == 1 // base case
        return list
    else{
        minElement = list.removeMin()
        list = sort(list) // has n-1 elements
        return list.addFirst(minElement)
    }
}
Example 5: Tower of Hanoi

Problem: Move \( n \) disks from start tower to finish tower such that:

- move one disk at a time

- smaller disk on top of bigger disk (but you can’t put a bigger disk onto a smaller disk)
Example: \( n = 1 \)

Example: \( n = 2 \)
Example: \( n = 2 \)

1. Move from A to C
2. Move from A to B
3. Move from C to B
Q: How to move 5 disks from tower 1 to tower 2?

A: Think recursively.
Example: $n = 5$

1. Move 4 disks from A to C

2. Move 1 disk from A to B

3. Move 4 disks from C to B
tower(n, start, finish, other) \{ // e.g. tower(5, A, B, C)

    if n > 0 {

        tower( n-1, start, other, finish)

        move from start to finish

        tower( n-1, other, finish, start)

    }

}
Example: \( n = 5 \) \( \text{tower}( 5, A, B, C ) \)

tower( 4, A, C, B )

tower( 1, A, B, C )

tower( 4, C, B, A )
Claim: the tower() algorithm is correct, namely it moves the blocks from start to finish without breaking the two rules (one at a time, and can’t put bigger one onto smaller one).

Proof: (sketch)

Base case: \( \text{tower}(0, *, *, *) \) is correct.

Induction step:

for any \( k > 0 \), if \( \text{tower}(k, *, *, *) \) is correct then \( \text{tower}(k + 1, *, *, *) \) is correct.
How many moves does tower(1, ...) make?

Answer: 1
How many moves does tower(2, ...) make?

Answer: $1 + 2$
How many moves does tower(3, ...) make?

Answer: \[ 1 + 2 + 4 = 2^0 + 2^1 + 2^2 \]
How many **moves** does tower(n, ...) make?

\[
tower( n, \text{ start}, \text{ finish}, \text{ other} )
\]

\[
tower( n - 1, \text{ start}, \text{ other}, \text{ finish} ) \quad \text{move} \quad tower( n - 1, \text{ other}, \text{ finish}, \text{ start} )
\]

\[
tower( n - 1, \text{ start}, \text{ other}, \text{ finish} ) \quad \text{move} \quad tower( n - 1, \text{ other}, \text{ finish}, \text{ start} )
\]

Answer: \[1 + 2 + 4 + \ldots + 2^{n-1} = 2^n - 1\]