CS159

Nathan Sprague

April 8, 2015

Recursive Definitions

Merriam Websters definition of Ancestor:

Ancestor

One from whom a person is descended [...]

Here is a recursive version:

Ancestor

One's parent.

or

The parent of one's ancestor.

Recursively Defined Functions

Classic example is the factorial function:

$n!$

if $n = 0$ then $n! = 1$ *(basis or initial conditions)*
if $n > 0$ then $n! = n \times (n - 1)!$

Recursive Methods / Recursive Programming

A recursive method is a method that includes a call to itself. It is often straightforward to compute recursively defined functions using recursive methods:

```
int factorial(int n)
{
    int value;

    if  (n == 0)
        value = 1;
    else
        value = n * factorial(n - 1);

    return value;
}
```

Activation Records

Every method call results in an activation record which contains:

- Local variables and their values.
- The location (in the caller) of the call.

Tracing Recursive Methods...

Recursion is Not Always the Best Approach

```
int factorial(int n)
{
    int value = 1;

    for (int i=2; i <= n; i++)
    {
        value *= i;
    }

    return value;
}
```

Recursive Problem Solving

Recursion is often a good idea when a problem can be solved by breaking it into one or more smaller problems of the same form. The process is:

- Figure out how to solve the easy case, i.e. the base case.
- Figure out how to move the hard case toward the easy case.

Recursion Pseudocode

Nearly every recursive method ends up looking like the following:

```

recursiveMethod(input)
{
    if (input represents a base case)
    {
        handle the base case directly.
    }
    else
    {
        call recursiveMethod one or more times
        passing it only part of the input.
    }
}
```

Recursion Pseudocode Variations

Sometimes there is nothing to do for the base case. We just want to stop:

```
recursiveMethod(input)
{
     if (input is NOT the base case)
     {
          call recursiveMethod one or more times
          passing it only part of the input.
     }

     // No else statement.  Nothing to do for the base case.
}
```

Example: Binary Search

Searching for a particular value in a sorted array.
(More than one base case.)

```
public static int binarySearch(int[] array, int first, int last, int value)
{
    int mid;

    if (first > last) // Easy/Base case.  No elements left. Failure!
    {
        return - 1;
    }

    mid = (first + last) / 2;

    if (array[mid] == value) // Another base case.  Success!
    {
         return mid;
    }
    else if (array[mid] < value)
    {
        return binarySearch(array, mid + 1, last, value);
    }
    else
    {
        return binarySearch(array, first, mid - 1, value);
    }
}
```

Aside: Tail Recursion

- A method is “tail recursive” if the recursive call is the final operation.
- It is straightforward to replace tail recursion with a while-loop.
- If a recursive method is not tail-recursive, then the call stack is doing important work: saving the state of an ongoing computation so that it can resume when the recursive call completes.

Aside: Tail Recursion

- A method is “tail recursive” if the recursive call is the final operation.
- It is straightforward to replace tail recursion with a while-loop.
- If a recursive method is not tail-recursive, then the call stack is doing important work: saving the state of an ongoing computation so that it can resume when the recursive call completes.
- Was the factorial method we saw earlier tail recursive?

Aside: Tail Recursion

- A method is “tail recursive” if the recursive call is the final operation.
- It is straightforward to replace tail recursion with a while-loop.
- If a recursive method is not tail-recursive, then the call stack is doing important work: saving the state of an ongoing computation so that it can resume when the recursive call completes.
- Was the factorial method we saw earlier tail recursive?
    - No. Multiplication occurs *after* the recursive call.

Iterative Binary Search

```java
public static int binarySearchIterative(int[] array,int value)
{
    int first, last, mid, result;
    boolean found;

    first = 0;
    last = array.length - 1;

    result = -1;
    found = false;

    while (first <= last && !found) // Check for "base cases"
    {
        mid = (first + last) / 2;
        if (array[mid] == value)
        {
            result = mid;
            found = true;
        }
        else if (array[mid] < value)
        {
            first = mid + 1; // "Recursively" search right
        }
        else
        {
            last = mid - 1; // "Recursively" search left
        }
    }

    return result;
}
```

The Coin Problem

Determine the minimum number of coins needed to make change for a given amount.

- The easy case:
  - We can use a single coin.
- Reducing the hard case:
  - Try every way of splitting the amount into two parts: j and amount - j
  - recursively find minimum coin solution for each pair
  - return total of the best split.
- Let's look at the code...