CS159

Nathan Sprague

November 9, 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
A parent of an 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:

```
1  int factorial(int n)
2  {
3      int value;
4
5      if    (n == 0)
6          value = 1;
7      else
8          value = n * factorial(n - 1);
9
10     return value;
11 }
```

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

```
1  int factorial(int n)
2  {
3      int value = 1;
4
5      for (int i=2; i <= n; i++)
6      {
7          value *= i;
8      }
9
10     return value;
11 }
```

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:

```
1
2  recursiveMethod(input)
3  {
4        if (input represents a base case)
5        {
6             handle the base case directly.
7        }
8        else
9        {
10            call recursiveMethod one or more times
11            passing it only part of the input.
12       }
13 }
```

There may be more than one base case.

Recursion Pseudocode Variations

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

```
1  recursiveMethod(input)
2  {
3       if (input is NOT the base case)
4       {
5            call recursiveMethod one or more times
6            passing it only part of the input.
7       }
8
9       // No else statement.  Nothing to do for the base case.
10 }
```

Example: Binary Search

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

```
1  public static int binarySearch(int[] array, int first, int last, int value)
2  {
3      int mid;
4
5      if (first > last) // Easy/Base case.  No elements left. Failure!
6      {
7          return -1;
8      }
9
10     mid = (first + last) / 2;
11
12     if (array[mid] == value) // Another base case.  Success!
13     {
14          return mid;
15     }
16     else if (array[mid] < value)
17     {
18         return binarySearch(array, mid + 1, last, value);
19     }
20     else
21     {
22         return binarySearch(array, first, mid - 1, value);
23     }
24 }
```

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

```
1   public static int binarySearchIterative(int[] array,int value)
2    {
3        int first, last, mid, result;
4        boolean found;
5
6        first = 0;
7        last = array.length - 1;
8
9        result = -1;
10       found = false;
11
12       while (first <= last && !found) // Check for "base cases"
13       {
14           mid = (first + last) / 2;
15           if (array[mid] == value)
16           {
17               result = mid;
18               found = true;
19           }
20           else if (array[mid] < value)
21           {
22               first = mid + 1; // "Recursively" search right
23           }
24           else
25           {
26               last = mid - 1; // "Recursively" search left
27           }
28       }
29
30       return result;
31   }
```

The Coin Problem

Problem: determine the minimum number of coins needed to make change for a given amount. E.g. we have pennies, nickels and dimes, and we need to 7 cents worth of change. How many coins will it take? Pseudocode:

- If the change amount is equal to a coin denomination.
  - Return 1; // Base case! only one coin is needed.
- Otherwise consider every possible way of breaking the change amount in two. Each split results in two smaller versions of the change problem. Solve the recursively, add the results, and return the lowest sum.
- E.g. if the amount is 7 cents we can try:
  - 1 and 6 $\rightarrow 1 + 2 = 3$
  - 2 and 5 $\rightarrow 2 + 1 = 3$
  - 3 and 4 $\rightarrow 3 + 4 = 7$