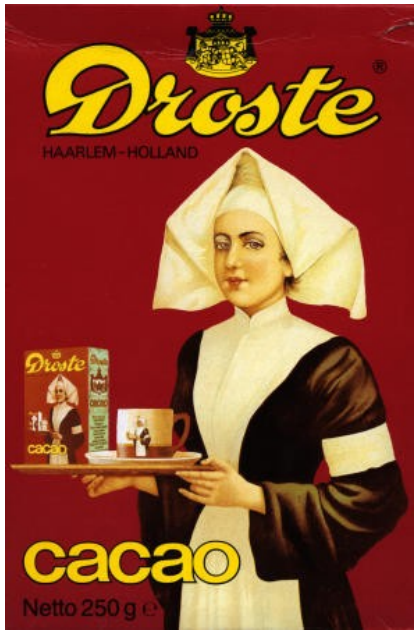
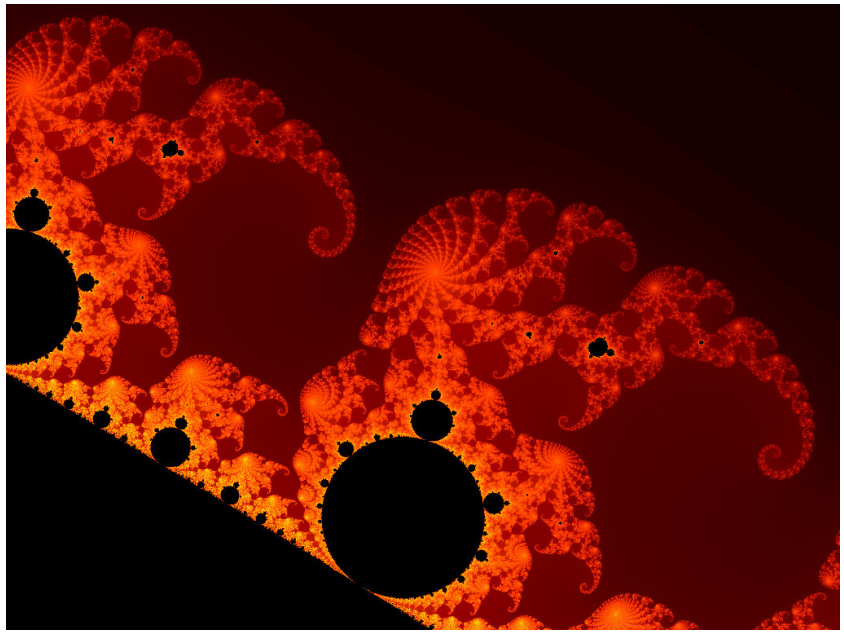
Multiple Recursion

Daniel Archambault

Previously in CS-115





To understand recursion you need to understand recursion.

Previously in CS-115

- What is recursion?

Previously in CS-115

- What is recursion?
- What are the two main parts to a recursive algorithm?

Previously in CS-115

- What is recursion?
- What are the two main parts to a recursive algorithm?
- What are the four stages to a recursive algorithm?

Previously in CS-115

- What is recursion?
- What are the two main parts to a recursive algorithm?
- What are the four stages to a recursive algorithm?
- What data structure is useful for tracing recursive algorithms?

Previously in CS-115

- What about two recursive calls?

Multiple Recursion

What if there are multiple recursive calls?

- Things get more complicated (tracing with a stack is a mess)
- However much of it is the same.
- You have already seen this with sorting maybe...

Hints to Writing Recursive Functions

- Think of cases that are solved easily
  - These are most likely your base cases
- Figure out how to divide the data
  - Division should approach simple cases
- Figure out how to synthesise solution from solutions

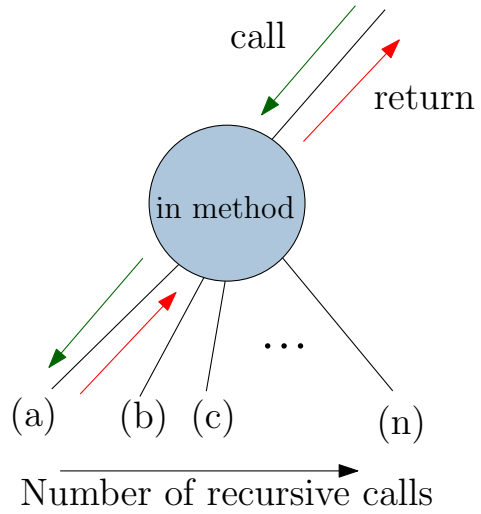
Elements of a Recursive Program:

- Base Case
  - Simple case of problem where we know the solution already
- Break down large instance
  - Input is broken down into smaller instances of the problem that move towards the base case
- Recursive Case
  - Apply the method being defined to the smaller instances
- Synthesise solution
  - Larger solution is synthesised from smaller solution

Steps to trace recursive programs

- Start at the top and trace down
- Whenever we encounter a recursive case:
  - Push new stack frame
  - Call the function again on its new arguments
- Draw a new node on your recursion tree with line number
- After base case achieved:
  - Return back up the recursion tree
  - Start from place where function already called

Tracing with recursion trees




```
public void main recFunction (...) {
   ... base case somewhere ...

   (a) recFunction (...);
   (b) recFunction (...);
   (c) recFunction (...);
         ...
   (n) recFunction (...);
}
```

Common Errors Writing Recursive Functions

- No Base Case
  - ▶
- Don’t Divide Input into Parts
  - ▶
-

Common Errors Writing Recursive Functions

- No Base Case
  - Recurse indefinitely
- Don't Divide Input into Parts
  -
-

Common Errors Writing Recursive Functions

- No Base Case
  - Recurse indefinitely
- Don't Divide Input into Parts
  - No progress made in recursive calls
-

Common Errors Writing Recursive Functions

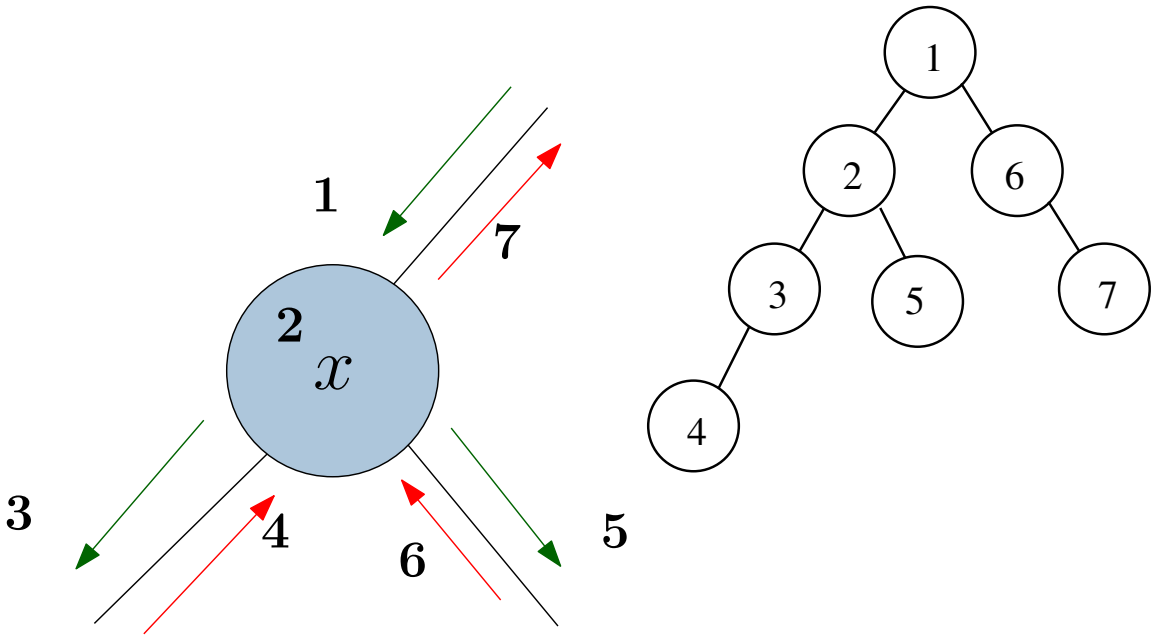
- No Base Case
  - Recurse indefinitely
- Don't Divide Input into Parts
  - No progress made in recursive calls
- Don't forget to make your recursive call

Examples of Multirecursion

- We've already seen this concept
  - but, I didn't tell you it was multirecursion
- Tree traversals are naturally recursive
  - visit is a call to a Java method that does work
  - the other two calls are recursive calls to the traversal

Preorder Traversal

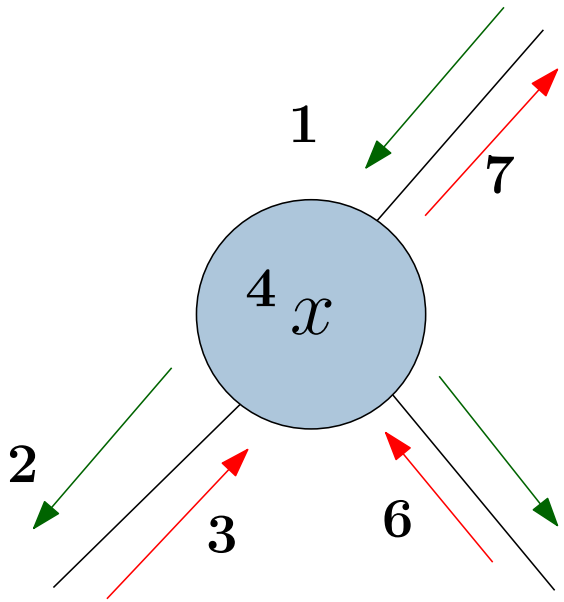
- Visit node and then its children

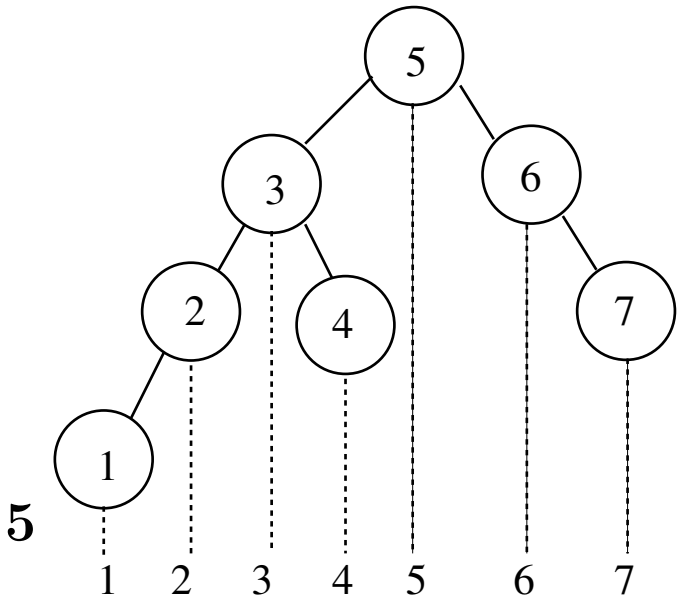



```
public void preOrder (TreeNode n)
{
   visit (n);
   if (n.left != null)
      this.preOrder (n.left);
   if (n.right != null)
      this.preOrder (n.right);
}
```

Inorder Traversal

- If a binary search tree, this traversal would be in order

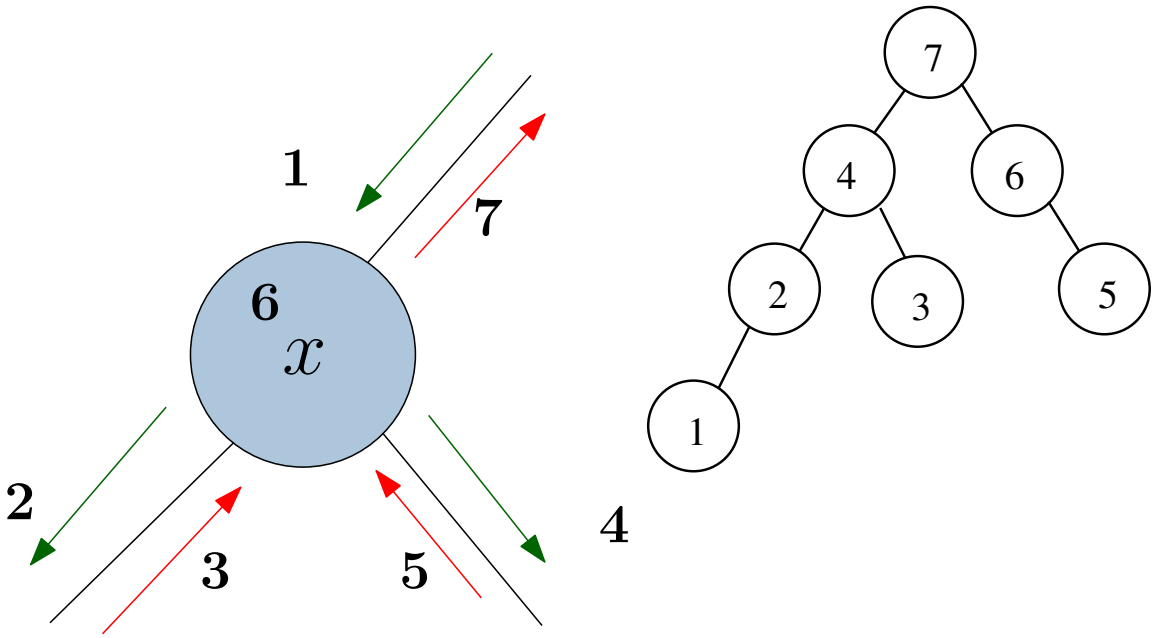





```
public void inOrder (TreeNode n)
{
   if (n.left != null)
      this.inOrder (n.left);
   visit (n);
   if (n.right != null)
      this.inOrder (n.right);
}
```

Postorder Traversal

- Visit both children and then the node




```
public void postOrder (TreeNode n)
{
   if (n.left != null)
      this.postOrder (n.left);
   if (n.right != null)
      this.postOrder (n.right);
   visit (n);
}
```

Find in a Binary Search Tree

- Finding an element in a binary search tree is also multirecursive
- However, recursion tree is a chain (because of the if statement)
- Returns null if does not exist and the tree node if it does

```
public TreeNode find (String value) {
  if (this.value.equals (value)) {
    return this;
  }
  else if (this.value <= value) {
    if (this.left == null) {
      return null;
    }
    else {
      this.left.find (value);
    }
  }
  else { //same for the right if this.value > value
```

Tracing Multirecusion

- To trace multirecursion, use a (conceptual) tree

```
public class UhOhRecursion {
  public static int numFun (int x) {
    if (x == 0) {
        return 1;
    }
    else if (x == 1) {
        return 3;
    }
    else {
        return numFun (x - 2) + 2*numFun (x - 1);
    }
  }
  ...
  System.out.println (UhOhRecursion.numFun (4));
}
```