Recursion, Tree Recursion, and Data Abstraction

Abhinav Ashar
CS 61A: Structure and Interpretation of Computer Programs

January 18, 2019

1 Recursion

Recursion is the idea of breaking down part of a problem and leaving the rest of the problem to be solved by recursive call. It can be a bit confusing at times, so the best way to think of it is with an example:

Say that you are standing in a line to enter a Warriors game. The line is pretty long, and you want to know how many people are ahead of you to determine your spot number. You ask the person in front of you, "What is your spot number?". That person, like you, does not know how many people are in front of them. As a result, they ask the person in front of them, "What is your spot number?". This cycle of people asking the people in front of them continues until the first person is asked. At that point, they will say that they are spot 1. From there, the person behind knows they are 1 more spot than that, so they know they are spot 2. As a result, the next person knows they are spot 3, then spot 4, then spot 5...until you find out your spot number. This is the idea of recursion. Rather than stepping out of line to count the number of people ahead of you, you realized that your spot was 1 more than the person in front of you and you asked them what their spot number is.

a) Every recursive call is a call to the original function, but with a different set of parameters.

b) Every recursive call requires a base case that stops the recursion, otherwise it will continue forever!

c) For recursive calls that result in true or false, you need to have a base case for true and a base case for false.

d) Don't try to do to many things in one round of a recursive call! Recursion is often more powerful than you think.

e) Try to break down a recursive problem into 3 parts: tools (the tools you need to solve the problem, calculations, formulas etc.), base cases (conditions that guarantee a True or a False, or some other condition that stops the recursion), and the recursive call.

f) **Looking Ahead:** Recursion is much more computationally heavy compared to iteration because it takes up more memory.

2 Tree Recursion

Tree recursion involves more than one recursive call. For example: return foo(n-1) + foo(n-2). In this example, you are doing 2 recursive calls and adding the results of each. If you want to get into the weeds, remember that both foo(n-1) and foo(n-2) will each have to return their own 2 recursive calls, and same for the next set of calls, and same for the next...But rather than getting bogged down in all the details of tree recursion, just remember that you don't have to be limited to doing only one recursive call at the end of a function!

a) Often, if you see a problem that involves multiple events, like event A and event B, that hints toward the problem possibly dealing with tree recursion.

b) Try to cover the each different event with one recursive call each.

c) The best way to visualize tree recursion is to draw out a tree for small cases, branching out at each call.

d) **Looking Ahead:** I mentioned in the description about how there can be many recursive calls for tree recursion. How many exactly? Well that depends on the specifics of the problem, but to approximate, observe in my example how each call to foo leads to 2 more calls. So 1 call to foo leads to 2 calls to foo, which leads to 4 calls, then to 8 calls, then 16...Notice a pattern?

3 Data Abstraction

Data abstraction is a concept that's meant more for humans than for computers. We want to write beautiful and easy-to-understand code. Any programmer should be able to look at your code and understand what is happening without trying to figure out what the variable x is and what the array 'stuff' is supposed to do. If we are writing a program about birthdays, and your birthday is stored in an array where a[0] = "December", a[1] = 31, a[2] = 1998, don't make someone else that is using your code to call a[0] to get your birth month. Instead, write a function called getBirthMonth() and call a[0] in there. That way, you "abstract" away the information and present the user with a more understandable tool.

a) Always use abstraction when writing code. It makes the code more understandable for you and the reader.

b) Write method names that are descriptive.

c) If you change the code for a certain element and don't use data abstraction, you may have to go and change every instance involved with the changed element. However, if you use a function, you only need to change the code in that function to address all the necessary changes.