Step 1 – Description

The Week 4 assessment Algorithms and Data Structures module required us to create and traverse binary trees, and create a number of functions that may be useful to when using binary trees. I’m primarily going to talk about the ancestor question in this critical review.

Step 2 – Feelings

Before approaching this task, I felt confident as I’d tackled all of the lab questions and had spent quite a lot of time trying to understand recursion and how it worked. That confidence was short lived when I realised how difficult the ancestor question, specifically, was. The previous questions hadn’t proved too difficult, but this one was much harder.

Initially I tried to brute force it, I could find a way that worked without recursion, but I knew that all of our teaching had been about recursion, and the code I was writing wasn’t very elegant nor was it efficient. I was getting very frustrated when I couldn’t find an answer straight away – this made me step back and reconsider my approach.

Step 3 – Evaluation

I think that fact that this question made me take a step back very much helped me to understand recursion in general. I knew there must be a way to code it recursively, so I set about breaking the problem down into smaller parts.

I knew we had 2 arguments passed in, but only one output at the end. So I knew somewhere in the function we would need review the data in a node, compare it to the arguments passed in, and return something if it was a match. I settled on the idea of just returning the node when this happened, because I felt it was more applicable to all scenarios (as opposed to returning a string, if the nodes were storing integers in the future). Then I had to figure out what I do with the values being returned –

I decided that the best way to identify if a node was a common ancestor, was to continuously pass the found nodes back up the tree until a node had 2 values returned to it. This led me to the realisation that all nodes would have to have 2 values returned to them, but one of more would be null. Dealing with double null was easy – you return null. Dealing with a single null, you return the non null value, then a double value was the point you found the ancestor.

From this point, you no longer needed the nodes of the strings you passed in, so you return the node of the common ancestor up instead. Because this is now returning a node up the tree, any other nodes on the journey will have one null, and one not null node, meaning the common ancestor node would eventually be returned.

Step 4 – Analysis

The experience was definitely very positive – overall I think this question was the hardest of the course so far and pushed my problem solving skills. I found recursion incredibly difficult to get my head around when we started doing it – I was able to somewhat work with the lab questions and make them work by tweaking existing examples we’d been given – but this wasn’t the case here. I spent a lot of time running through examples on places like Code Academy and consulting the textbook I’d bought for my previous module on object oriented programming, which gave me a much greater understanding of what I was doing.

Step 5 – Conclusion

At the start of the course, there was a mention about the different between coders and programmers, and that’s reiterated by the article link on moodle – this question more than any other helped me to understand that. I knew there were various different ways I could approach it, but I wanted to look at it efficiently – I believe I achieved that; if Im correct the Big O notation is n – but not only in time efficiency, in memory efficiency too – as another approach I tried generated strings of the entire tree in different traversal methods, generating redundancy along the way.

As from that, I think the solution I came up with works well in a simplistic world – however, there could be problems later down the line if the tree is very very large, and there are multiple nodes that aren’t unique and could have multiple instances of the strings we pass in as arguments.