Recursion Written Supplement

Aritro Saha

Overall Narrative

The overall narrative of the visual is supposed to illustrate the steps that the algorithm goes through in order to solve a Tower of Hanoi puzzle given a set of constraints. The Tower of Hanoi puzzle involves moving a set of discs from one rod to another, given three rods and a set of rules. The rules of the puzzle are that a larger disc cannot be on top of a smaller disc, you can only move one disc at once, and you can only move the top disc of a rod. The visual begins with a diagram of the desired movement: moving all the discs from the leftmost rod to the rightmost rod. The depth of the recursive stack is shown by using encapsulating rectangles, and a function call that starts a new level in the stack is shown using a question mark and orange arrow. The green arrows indicate a call to “move_disc” to move a disc from one rod to another rod, following the puzzle’s constraints. These actions are encapsulated in their own boxes to indicate that it was performed by a function call. The blue arrows indicate a step in the recursive function. As such, each level only has two blue arrows (ignoring those inside it), as they are in-between the three steps performed by the recursive function.

Alternate Meaning

A major alternate meaning that one may take from the diagram after looking at it is believing that the question marks indicate that there are multiple possibilities. In the puzzle, there are many different combinations of steps that you can take in order to solve it. However, since the algorithm only solves for the shortest number of steps, there is only one set of steps that can be the shortest for this problem.

Issues

Programming

An issue with programming it that I faced was taking my theoretical algorithm and converting it into code. After thinking about the problem for a while, I tried solving it myself and ended up finding a rough algorithm that could solve it, which would be to move the discs on top of the disc we want to move onto an auxiliary rod (the only rod we don’t use), moving the disc we want to move onto the desired rod, and then moving the rest of the discs on the auxiliary rod on top of it. However, turning this into a recursive algorithm was difficult as it was difficult to see how the function would call itself to continue solving the puzzle. After outlining the different cases and drawing rough diagrams similar to the visual aid, I was able to gain a better understanding on how I would go about writing code to implement the recursive algorithm. In addition, a more

traditional issue that I ran into was the time complexity of my solution. Since recursion involves running itself, its time complexity grows exponentially as O(x^n), where x represents the number of function calls made to itself. Since my solution runs its own function twice, this causes it to have a time complexity of O(2^n). The sluggishness of the solution is apparent when a large set of discs are desired to be moved, such as 100 discs from the leftmost rod to the rightmost rod.

Visual Development

The development of the visual aid was somewhat difficult due to the limitations on what could be used in the diagram. Since we could not use any words or numbers, I was a bit confused at the start about how I would indicate steps in the recursion process. By using a variety of different symbols and colors, I believe that I was able to nicely visualize the recursion that takes place to solve the puzzle. I was also confused by how I would explain the rules of the Tower of Hanoi puzzle without any words. By using symbols and a game state diagram that I use everywhere else, I believe I was able to also explain the rules of the puzzle while following the restrictions.

Aversion of Pitfalls

Two pitfalls of recursion that I did not run into were high memory usage and hard-to-read code. First, after observing the memory usage of the program after running it with large constraints (moving 100 to 500 discs from one rod to another), it would never exceed around 3.8 MB, which is not that much compared to how deep the recursion stack can go. This allows it to run with large constraints without running out of memory. Another pitfall of recursion that I didn't run into was hard-to-read code. The recursive function is very clean as minimal cases are required. By using code documentation standards, it is also very easy to understand how the function works and how it relates to the overall recursive algorithm.