ALI SALEHI ASGN3 WRITEUP.pdf

STACK SOLUTION

To formulate my stack solution, I played the towers of Hanoi game dozens of times while analyzing patterns to implement through my code. The first pattern substantial pattern I noticed and then implemented was that the smallest disk is always moved every other move. It starts by moving to a specific peg (that depends on whether the number of disks is even or odd) and then moves every other move. After noting this, and figuring out which disk it moves to (which depends whether the last disk moved was even or odd) I had to decipher what the moves in between were. Quickly I noticed that between moving the smallest disk, the next move was the only legal move available, but was also moving the next smallest disk where the smallest disk was just moved from. Using this knowledge I created two functions that each returned a reference to the peg that was just added to, and used this information when calling the next function. Finally, I created a loop that ran these two functions repeatedly until the two non destination pegs were empty.

RECURSIVE SOLUTION

For the recursive solution, I used the standard approach with a base case, where there is only one disk, and a case for every other disk. My implementation revolved around moving (n-1) disks to the non destination peg, moving the largest (n) disk to the destination peg, and then moving the (n-1) pegs to the destination peg. This solution was effective because every time the function was called with n-1 pegs, it would shrink the problem down until there was only one peg to move, and it would move it to the relative destination in the base case of the function. It then moves back up and transfers the rest of the pegs. This solution was a little more abstract and took longer to absorb, yet was also much more elegant and rewarding.

COMPARISON

Overall, the stack implementation and the recursive implementation are both able to solve the towers of Hanoi problem in the same amount of moves, (2^n)-1, regardless of how many disks. Where the two solutions differ is in their memory usage and ease of reading/understanding. First, it is likely that the stack implementation of the problem uses less memory and is hence more efficient because the recursive method needs to store each successive recursive function call on the stack.

In contrast, the iterative stack implementation uses malloc() to store the exact amount of memory needed to to hold disks on each stack, and is less intensive with memory. In contrast, the programmer must take much more effort to write the iterative stack solution in comparison to the recursive method. To keep track of where the disks are and where they need to move, we had to implement a dynamic stack data type of our own. With this came the implementation of specific functions to be able to use that new data type effectively. In addition, the iterative solution required much more user analyzation of previous moves, current states of the pegs, and locating where disks need to be moved. The recursive solution does this all under the hood by creating subtasks and letting the recursion take care of its implementation. Thus, the recursive method may be more clear and is surely more concise than the stack implementation, but it also takes more memory to solve the same problem.