Sanjana Rathore

UID: 006216683

Project 2 Report

Notable Obstacles and Coding Process:

I think the hardest part of those coding projects was thinking about these functions recursively when trying to figure out how to code. The modulo function was definitely the easiest function to implement. Occurrences is when it started getting a bit harder. For occurrences, I initially tried coding it with only one base case and it wasn’t working. To get my code to work, I made a base case for when both parameters were equal to 0 (in this case I returned 1) and another base case for when the number was equal to d (in this case I also immediately returned 1). If the number was less than 10 and not equal to d, I returned 0. The recursive part of the function worked by using modulus to check if the last digit of the number was equal to d and then passing the initial part of the number with the last number taken off (using /) through the recursive call. Lucky7s had me initially a little stumped so I moved on to combinations. We did a similar problem in our discussion (isSolvable method) and I was able to use similar logic to implement the combinations method. Either the number in the array could be used towards meeting the target or it would not. To implement this logic I created two recursive calls and returned the or of them so that if there was any case that ended up being true the whole function would return true. Once I nailed down combinations, I moved on to lucky7s. For combinations I went backwards to forwards in the array so I tried using similar logic for lucky7s and struggled with the implementation. Once I got the hint from the lecture that lucky7s should be implemented by iterating through the string forward to the end, I was able to figure out the logic.

Testing:

I started off by testing the assert commands provided in the given main function and I was able to get them to work so I proceeded with creating my own edge cases. When testing out modulo, I noticed that if I passed a very large number for the m parameter and a small number for n, the function took a very long time to run. Some edge cases I implemented were m equaling 1 and 0 or n equaling 1. For testing occurrences, the first case I tested was both number and d equaling 0. I had a base case that took care of this so it was successful. I tested cases where the d was present in the number and d was not present. For lucky7s, the edge case I tested (in addition to testing strings where matching characters were and were not present) was an empty string. My base case for lucky7s, returns the string as is if the length is less than or equal to 1 so my code was able to handle such a case. When testing out combinations, I tested various arrays of different lengths and numbers and implemented cases where the numbers added and did not add up to the target. However, I did run into an interesting case which I was not able to figure out. What if the target was 0? My code works by subtracting different combinations of the numbers in the array and when the target equals 0 it returns true because that means there was a combination of numbers from the array that equaled the target. So when I created a test case for a target equalling 0 the case would return true even if there was no 0 in the array which meant there should theoretically be no combination of numbers that adds up to the target. The instructions did not clarify what should happen if the target was equal to 0 so I left my code as is, but it is definitely a case that should be considered depending on how the function is supposed to work. However, it was not clarified in the instructions.