Joan Vizcaino

November 27, 2019

Midterm #2

CS 1290 – Special Topics in Computing

Fall 2019

**1 – Stone Game**

The problem is to return True if and only if Alex wins against Lee by collecting the most stones from a pile of stones. Each person beginning with Alex will grab either the front or end pile of stones. And continue until there are no more piles left. The person with the most stones wins the game.

This problem solution is very similar to the House Robber problem given earlier as homework during the semester. To solve this problem, I decided to create 2 separate lists with the same length as the original list given to us. One list will contain the max while the other min or in other words Alex and Lee. This will continue on recursively until the end of the list is reached using the same recursive equation as the house robbing problem. Which would be dp[i][j] = max(piles[i]-dp[i+1][j],piles[j]-dp[i][j-1]. In the picture below you can see how I would use the two lists similar to the house robbing problem.

A screenshot of a cell phone screen with text

Description automatically generated

I used IDEAL to help better understand the problem by identifying what the problem was and finding the goals I needed to accomplish and answer the problem. It also helped me to explore many different instances and variations of the problem in trying to break my code and thinking for the problem. With this it helped me anticipate different things and work on them to where I would be able to solve the problem. IDEAL helped me a lot with solving the problem by getting me to actually look and understand the problem and learn from mistakes and assumptions I was making when trying to solve this problem.

**2 – Minimum Falling Path Sum**

Given a square array of integers A, return the minimum sum of a falling path through A. A falling path is defined as any element in the first row and picks one element from each row. The new row chosen needs to be from a different column then the first one by at most one.

To solve this problem, I wanted to use a 2D array to store all the elements from the square array and manipulate it to where it gives you the minimum value from each row and column to ensure the minimum sum of a falling path. For this we would you the recursive equation of path = min(path,sq[I + 1][j + 1][j – 1], min(path, sq[I +1][j +1] to help recursively solve smaller instances of the problem and use it to solve the bigger problem.

For this problem I used Duke’s 7 Steps to help better understand and create a solution for this problem. I used Dukes 7 steps by creating different instances of the problem to fully understand the problem and its requirements. From there I was able to find patterns that can useful to solve duplicate work which allowed me to convert it all to code and solve any errors.

**4 – Arithmetic Slices**

An arithmetic is a sequence of numbers that consists of at least three elements and if the difference between any two consecutive elements is the same. Return the number of arithmetic slices in the given array A.

When trying to solve this problem I noticed that for an arithmetic slice to be formed it needed a size of at least 3. As well as the solution needs to satisfy certain conditions to be considered an arithmetic slice. The recursive equation for this solution would be dp[I – j – 1] to check if the next sequence is greater than the previous to make the requirements. This would solve sub problems to help solve the bigger problem.

For this problem I used Duke’s 7 Steps to better understand this problem by thinking of different instances to write down calm steps in solving them to hopefully find patterns to solve small or duplicate work to help solve the problem. I can check these steps by tracing the instances I created to test if my algorithm will work and hold up. From there I tried to write the code to test and see if it works and debug any errors.

**8 – Partition to k Equal Sum Subsets**

Given an array of integers called nums and a positive integer k, find whether it’s possible to divide this array into k non-empty subsets whose sums are all equal.

To solve this problem, I would use an array list to store my answers and start by solving small instance of the problem and use them to solve duplicate work to solve the big problem. For example, I could use backtracking by going through the list and checking if each index goes into either k groups to see if the array can be split by equal sums. I planned to solve the problem by using backtracking and returning either true or false depending if the list can be split equally into k groups equaling to the same sum.

To solve this problem, I used Duke’s & Steps by creating small instances of the problem and working out the original list of arrays. For example, I created some different variations where I wrote down my steps. From there I was able to see patterns or repeated work of the problem and working it out. From there I changed it over to code and ran different test cases to ensure it worked with different variations of the problem.

**9 – Perfect Squares**

Given a positive integer find the least number of perfect square numbers sum up to the integer.

To solve this problem I wanted to create a new list where I would check to see if each index is a number from a perfect square and if it was of multiple squares I would collect the minimum one and save this into the new list and use this recursive method to save subproblem solutions used to solve duplicate work or the bigger problem. The recursive equation/ algorithm I used was min(dp[I – j \* j]) so it would collect the minimum perfect square value if it had multiple. With this it will solve smaller sub solutions to solve the bigger problem and produce the solution needed to solve the problem.

For this problem I used both IDEAL and DUKES 7 steps to get a better understanding of the problem and its requirements to solve this and come up with a solution. I identified the problem by creating different instances of the problem to understand it all better. I then described the problem and wrote down steps I used to possible solve the problem. With explaining to myself why I did certain steps helped me find patterns of duplicate work which helped me create an algorithm and catch any errors to solve the problem.