Cameron King, Nick Jonas, Rahul Rangarajan

5/10/202

CSC 301

Assignment 4 Report

With assignment 4, we followed a similar approach as assignment 3, in which we loosely divided the questions between us and got to work. Most of them were fairly different from the previous assignments, as they asked for explanations rather than java or pseudocode. However, this posed no major difficulty and encouraged each of us to try and solve each other's problems if we had spare time. Of course we still peer read and asked each other for help, but on top of that we had meetings to compare work. We met a total of two times, and in both these times we managed to solve all our problems we had.

When researching optimal substructure property, we learned that the goal is to find the most optimal solution as quickly as possible using sub problems of the original problem given. What we mean by this, is breaking the problem into multiple smaller problems in order to solve one large problem. During our research we came across an example of what the shortest path was and felt as though that would be the best way to voice our finding.

While researching overlapping subproblems we learned that overlapping is similar to optimal substructure (as in we break the problem down into multiple subproblems) but this time we use a recursive algorithm that will solve the same sub problem instead of constantly creating new sub problems.

When attempting to solve number 3, we faced our biggest problem: trying to understand the algorithm itself. We discussed this problem multiple times, only after trying to explain the problem to one another did we manage to understand it. Once we understood it, we each tried to solve the problem on our own making many mistakes along the way such as accidentally taking the max instead of the min, or unknowingly writing down the wrong number and creating errors in calculations. After catching these mistakes and discussing our solutions, we solved the problem.

Question 4 was quite a simple task as all it required was noticing the pattern when adding 1s and 2s. The difficulty was discerning if this was truly dynamic programming. This was mostly because we struggled to define dynamic programming. After some discussion, we decided that dynamic programming was the use of breaking down a problem into subproblems using subcalls of a singular method. This allowed us to say that question 4 was not dynamic programming.

For question 5, we knew that the rod cutting problem was a classic example used to introduce the concept of dynamic programing. The optimal solution was easily defined and the optimal substructure was easy enough to devise. If anything, we struggled with seeing a way to figure out the problem without the use of dynamic programming that was still near as efficient. Of course a bottom up algorithm might have been able to come close, but without dynamic programming’s approach, countless steps would still be repeated.

For question 6, we still had our java program for a max heap, so we started by thinking about how we could modify it. In order for the smallest number to appear as a root, while the largest appear as the bottom children, we decided only a simple difference was present. With guaranteed ascending order down the tree, it was apparent that instead of looking for the index with the largest number, we needed to find the index with the smallest number. A few changes to the comparisons and the code worked. We tested multiple values and found nothing wrong with it, but it's still possible we missed something.

For question 7, the largest issue we had was figuring out how to keep track of the levels in the tree. So we started by making a simple method to find the “height” of a node, with the root being the tallest. Using two functions, we wrote that one function would utilize a for loop to make multiple calls of the second one per level. The second function would then only print once it reached the specified level. We figured this out thanks to some loose inspiration from the function on the quiz. While it serves a completely different purpose, the way it handled the recursive calls was enough to convince us to try something similar.