Cameron King, Nick Jonas, Rahul Rangarajan

5/10/2020

CSC 301

Assignment 4

Q1. A given problem has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems. An example of the shortest Path problem would be: If a node x lies in the shortest path from a source node u to destination node v then the shortest path from u to v is combination of shortest path from u to x and shortest path from x to v.

Q2. A problem has overlapping subproblems if the problem can be broken down into smaller problems which are reused several times or a recursive algorithm for the problem solves the same subproblem over and over rather than always generating new subproblems.

Q3. See the Q3 folder for the answer

Q4. When attempting to count the number of ways to reach a sum using 1s and 2s.we decided to first write out the solutions for the first few numbers. When doing this, we realized that there was a pattern. If n = the sum, there are (n/2)+1 solutions. This type of solution is not dynamic programming.because my equation is not being divided into multiple subproblems. Dynamic programming takes a problem and breaks it down into simpler problems. This allows the system to decrease the time complexity. In this case though, the time complexity would be 1 because it is only one line of code. This type of problem will not be broken down into multiple subproblems showing that it is not dynamic programming.

Q6. See MinBinHeap

Q5. With the rod cutting problem, the way we solve it with dynamic programming is to do recursive calls for every possible combination of cuts. This is the optimal substructure for the problem, which allows us to easily divide it into smaller subproblems. By using an array to store the results of smaller cuts, we have greatly reduced the time complexity. This allows for the calculation of the solution sooner than if another algorithm had been used.

Q7.

BT = Binary Tree

node = node on a tree

Algorithm: findHeight(node): //finds the height of a certain node

1. If node != null

2. return 1 + max(findHeight(node.left),findHeight(node.right))

3. return 0

Algorithm: levelOrder(BT): //Prints Tree by level using levelOrderAux

1. for i -> 1 to findHeight(BT.root)

2. levelOrderAux(BT.root, i)

Algorithm: levelOrderAux(node, height):

1. if node != null

2. if height == 1

3. print(node.data)

4. else if height > 1

5. levelOrderAux(node.left, height-1)

6. levelOrderAux(node.right,height-1)