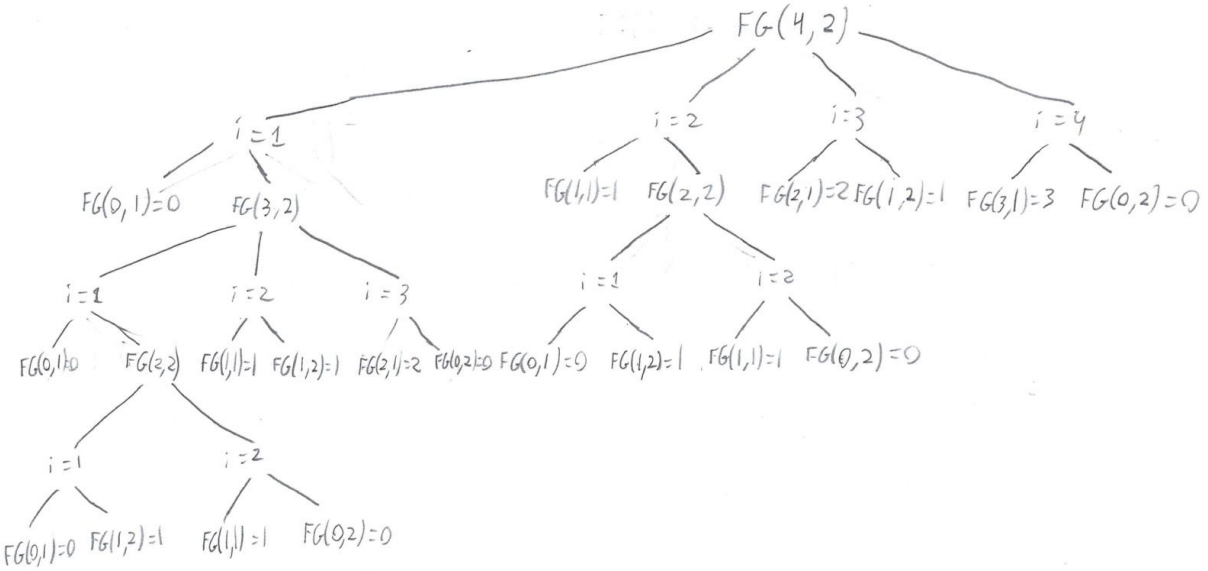
Dynamic Programming Writeup

Falling Glass:

a) Description of optimal substructure/recurrence:

There are two scenarios when you drop a glass at floor = i: (a) it shatters, and you have to search the number of floors below i with one less glass or (b) it doesn't shatter, and you have to check the number of floors above i with the same amount of glass. In either scenario, the problem left to solve is a smaller subproblem of the original, leading to a natural recursive/dynamic programming solution.

b) Recurrence tree (floors = 4, sheets = 2):



d) Distinct subproblem (floors = 4, sheets = 2):

	0	1	2	3	4
0					
1	x	x	x	x	
2	x	x	x	x	

4 * 2 = 8 (see e for a more general explanation)

e) Distinct subproblem (floors = n, sheets = m):

For the problem of n floors and m sheets, you will have to solve n * m subproblem. This is because you have to check all the amounts of floors less than the current number

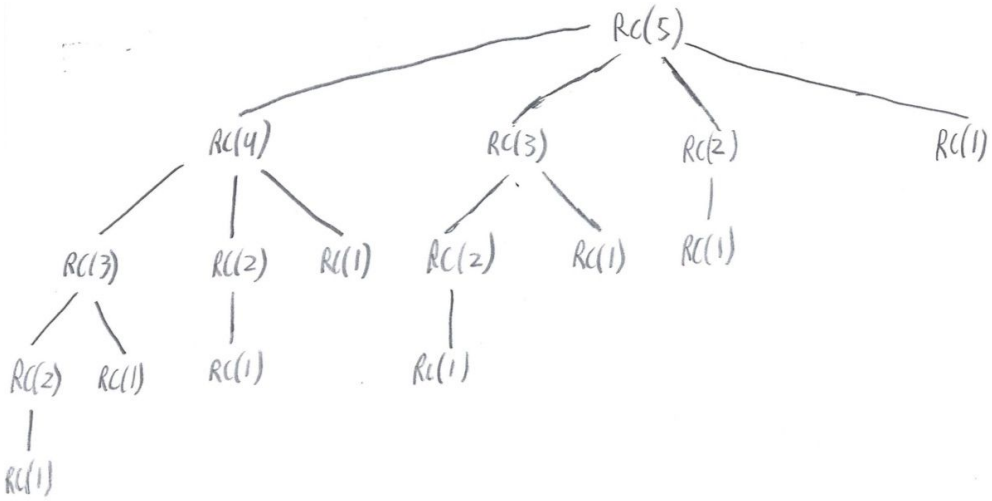
(including 0) with the same amount of sheets, as well as all the amounts of floors less than the current number with 1 less sheet. Recursively, this means that you will end up solving all variations of problem where 0 <= floors < n and 0 < sheets <= m.

f) Description of memoized version of GlassFallingRecur:

To memoize GlassFallingRecur, I reused most of my code from GlassFallingRecur within a helper function. The helper function is passed in an (n + 1) * (m + 1) array (where floors = n & sheets = m) initialized to -1. The helper function first checks if there is a valid answer for the number of floors & sheets in the array, if so, it returns that answer, if not, it solves for that number of floors & sheets and put the answer in the correct location in the array before returning it. This way, if a particular subproblem has previously been solved, it doesn't need to be recalculated.

<u>Rod Cutting:</u>

a) Recursion tree (rodLength = 5):



b) Question 15.1-2 (Page 370):

A counterexample would be a rod with prices:

Length: 1 2 3 4 5 6

Price: 1 1 7 10 1 1

According to the greedy algorithm, the first choice would be a rod of length 4 (density = 2.5), followed by two rods of length 1 (density 1), for a total of 12. However, the optimal solution would be to make two rods of length 3 (density 2.3) for a total of 14.