Eldrid Gonsalves

CS323 3/29/19

1. Falling Glass

a) *Optimal Substructure That Leads to a Recursive Solution*

If you drop a glass sheet from the k^{th} $floor$, $(1 \leq k \leq n)$, there are two cases:

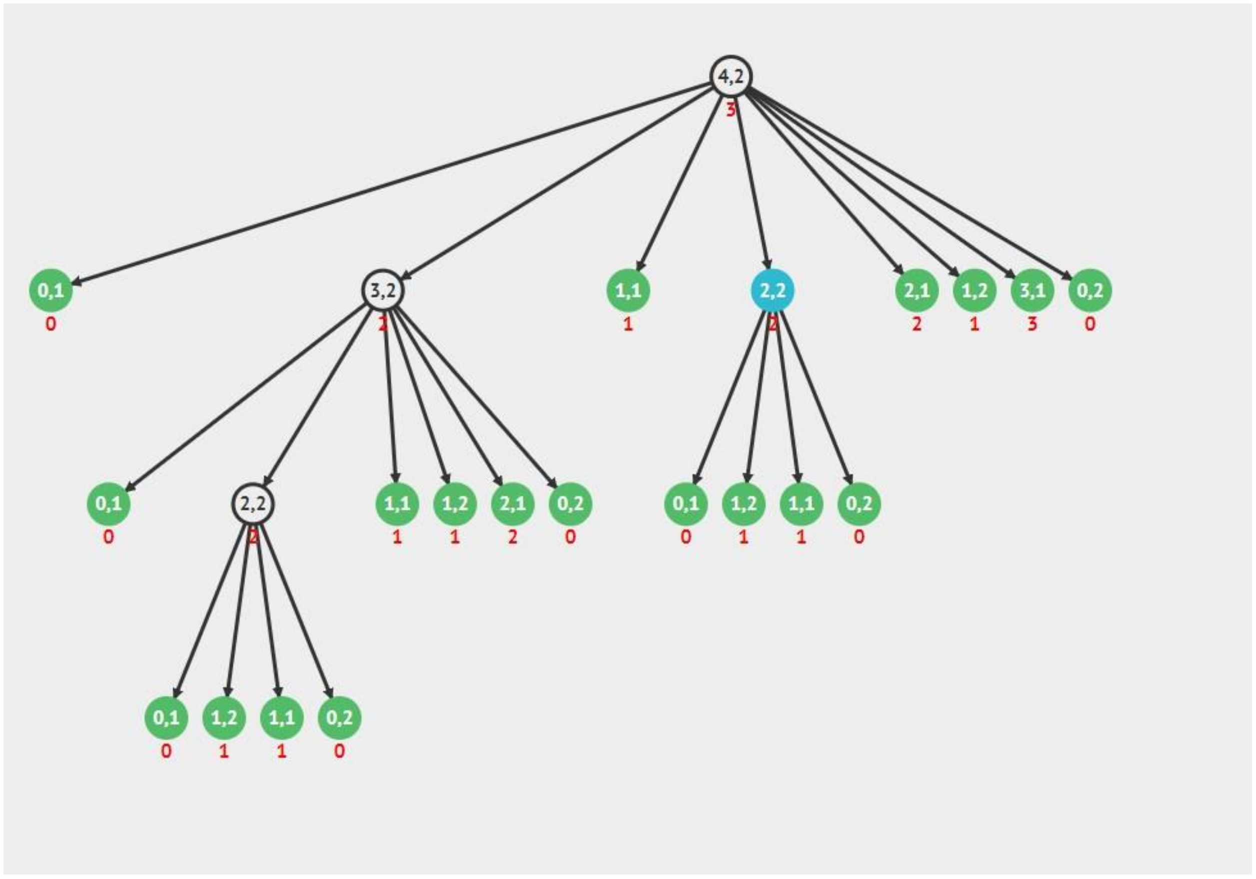
1. The glass sheet breaks.
   If the sheet breaks from the k^{th} floor, then we only need to check for floors lower than k with the remaining number of sheets. The new subproblem is reduced from the original problem to $k - 1$ floors and $m - 1$ glass sheets.
2. The glass sheet does not break.
   If the sheet does not break from the k^{th}floor, we only need to check for floors higher than k. The new subproblem is reduced from the original problem to $n - k$ floors and m glass sheets.

Using the two cases, we can deduce a recursive solution for the Falling Glass problem as follows:

$$gF(n,m) = 1 + \min\{ \max \{ gF(n-1,\ m-1), gF(n-k,\ m)\} \},$$

where n = floors, m = glass sheets, gF = glassFallingRecursiveFunction, $k = 1, \ldots, n$.

b) *Recurrence Tree for Given (floors = 4, sheets = 2)*



c) Solution in GlassFalling.java file.

d) *How many distinct subproblems with given 4 floors and 2 sheets?* Counting the distinct subproblems in the recurrence tree from part b), I came up with 8 distinct subproblems.

e) *How many distinct subproblems for n floors and m sheets?* Using part d) to come up with the number of subproblems for the given 4 floors and 2 sheets, I noticed that the distinct subproblems equate to the product of 4 floors and 2 sheets. There are $n * m$ distinct subproblems for n floors and m sheets.

f) *How would you memoize GlassFallingRecur?* To memoize the recursive function, I would implement a 2D integer array of dimensions $n + 1\ by\ m + 1$, where n = floors and m – sheets, that will hold onto computed values of subproblems so that the algorithm does not need to repeatedly calculate values for problems already solved, which is the main reason the recursive algorithm is very inefficient. Each recursive call will access the 2D array to find a computed value given an index ($O(1)\ operation$), thus reducing the overall time complexity.

g) Solution in GlassFalling.java file.

2. Rod Cutting
   a) Recursion tree on computer paper
   b) 15.1-2 Counter Example:

Length i -	1,	2,	3,	4
Price p_i -	1,	22,	36,	40
Density $\frac{p_i}{i}$ -	1,	11,	12,	10

Given a rod of length 4, the greedy algorithm will cut it into 2 rods – one of length 3 and one of length 1 because length 3 has the greatest density and the total price will be 37. The optimal way would be to cut the rod into two rods of length 2, where the total price will be 44, resulting in more profit than the greedy algorithm.

   c) & d) Solutions in rodCutting.java