Hui Lin DP Writeup

Falling Glass

(a) Describe the optimal substructure/recurrence that would lead to a recursive solution.

Suppose we start with *n* sheets and *k* floors. We can choose to drop the first sheet from among any of the *k* floors. If we drop the sheet from some floor *i*, 1 <= *i* <= k, then there are two cases: the sheet breaks or the sheet does the break.

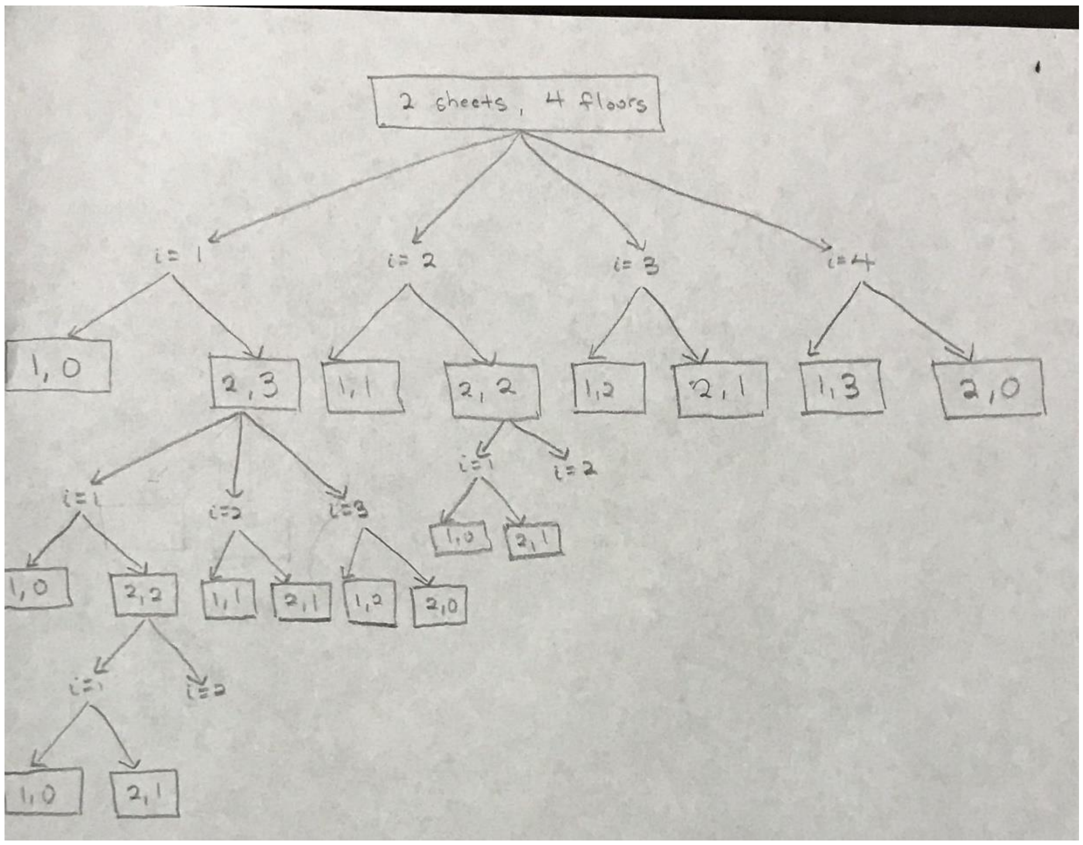
If the sheet breaks from falling from floor *i*, then we'd have to the test the lower *i – 1* floors with 1 less sheet, so we'd need the optimal solution to the subproblem with *n – 1* sheets and *i – 1* floors.

If the sheet does not break, then we would have the same number of sheets of glass, but we'd have to the test the floors above, so we'd need the optimal solution to the subproblem with *n* sheets and *k – i* floors.

Since we are minimizing the number of trials in the worst case, we can take the worse of the two cases described above for each floor and pick the floor with gives us the least number of trials.

Our boundary conditions are: if *k*, the number of floors is 0, then 0 trials are required. If k is 1, then 1 trial is required. If the there is 1 sheet, then we'd have to try every floor, so *k* trials are required.

(b) Draw recurrence tree for given (floors = 4, sheets = 2)



(d) How many distinct subproblem for 4 floors and 2 sheets?

There are 10 distinct subproblems, since there is either 1 sheet or 2 sheets and 5 different floors (since we include 0th floor) to test trials from.

(e) How many distinct subproblems for n floors and m sheets?

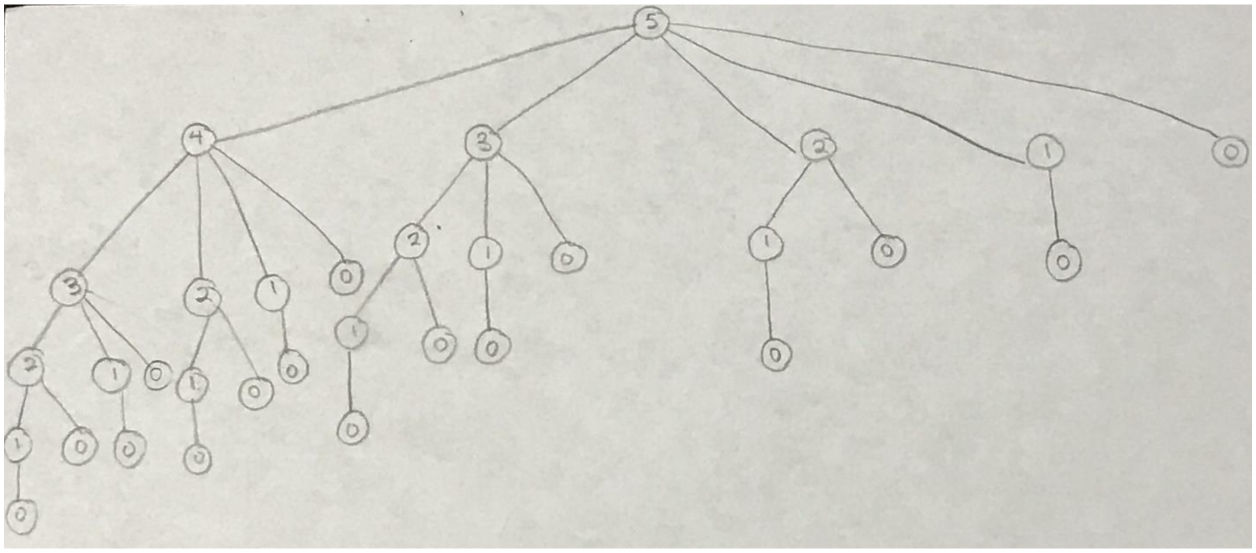
There are (n + 1) distinct floors to consider, since we also look at floor 0. For each sheet, we consider perform the trial of dropping from each floor. So, there are (n +1) * m distinct subproblems, and O(nm) distinct subproblems.

(f) Describe how you would memoize GlassFallingRecur.

I would use a 2d array of size n + 1 by k + 1, where n is the number of sheets and k is the number of floors. Let's call that 2d array memo. The elements of the array would be initially all Integer.MAX_VALUE to indicate that the calculation for the subproblems have not been made. memo[i][j] would store the minimum amount of trials needed in worst case with i sheets and k floors. If the calculation has been made for the subproblem with i sheets and k floors, then when we try to access the value at memo[i][j] the value would be less than Integer.MAX_VALUE, so the calculation for each subproblem only needs to be done once.

Rod Cutting

(a) Draw the recursion tree for a rod of length 5.



(b) On page 370: answer 15.1-2 by coming up with a counterexample, meaning come up with a situation / some input that shows we can only try all the options via dynamic programming instead.

Counter example:

Prices	2	27	41	29
Density	2.0	13.5	13.67	7.25

If we apply the greedy algorithm, the length of the first cut would be 3, since the density is highest at 3. We would then be left with a rod of length 1. The prices of the two rods are 41 (length 3) and 2 (length 1) for a total of 43. The dynamic programming algorithm gives us 54 in this case, which outperforms the greedy algorithm.