Adam vanWestrienen
CSCI 323 – Prof Yao

1. Falling Glass

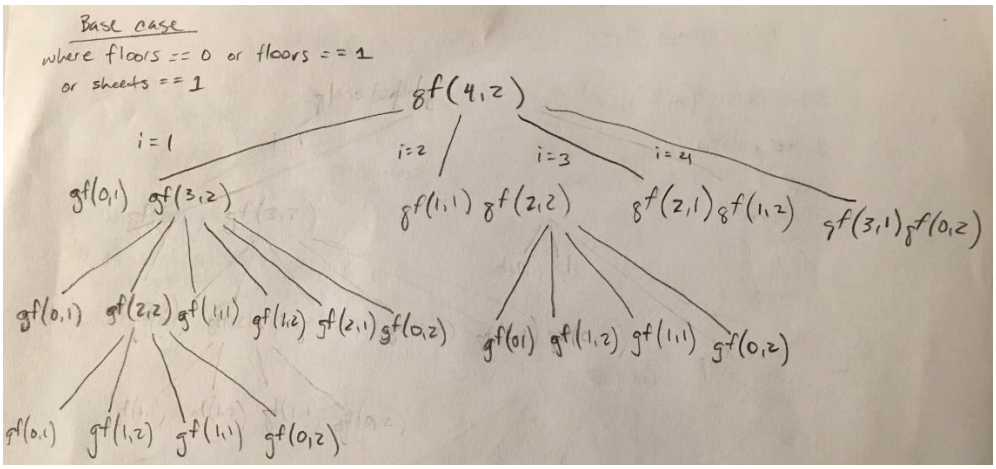
a) Optimal Substructure / Recurrence

Step one of dynamic programming says that we need to find a recursive solution to the problem. The problem we are faced with is to determine the minimum number of attempts the company should make to determine glass breakage from different heights given a number of floors and a number of sheets to experiment with. By nature, this problem is recursive because the exercises we're performing, i.e. dropping a sheet from a floor, recording whether it breaks or not, and moving to the next floor, is repetitive.

There are two outcomes for each drop: the glass sheet breaks or the glass sheet does not break. If the glass breaks on the floor we are currently testing, we know we can eliminate the floors above the current floor and only focus on the floors below it. We know this by the problem description that the glass will also break from every floor above the current floor where the glass broke. Additionally, if the glass breaks on the current floor, we are left we one fewer sheets of glass. Therefore, if we are testing 1 to x floors, with n glass sheets, our problem now becomes our current floor -1 and n - 1 glass sheets. In other words, we now only focus on everything below our current floor with one less sheet of glass at our disposal.

Likewise, if the glass does not break, it means we can go higher and our efforts focus above our current floor. So if we have x floors, we now have the current floor – x floors to check and we still have n number of glass sheets to test with. Because we are looking to find the highest floor for which the glass will not break using the minimum number of attempts, the optimal recurrence solution becomes combining these two scenarios; the glass breaking / the glass not breaking, at each floor.

b) Recurrence Tree for 4 floors and 2 sheets



c) Recurrence Solution

See GlassFalling.java file

d) How many distinct subproblems do you end up with given 4 floors and 2 sheets?

There are 8 distinct subproblems given 4 floors and 2 sheets.

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

The formula here is 2(n) subproblems because at each floor we have two possibilities.

f) Describe how you would memorize GlassFallingRecur

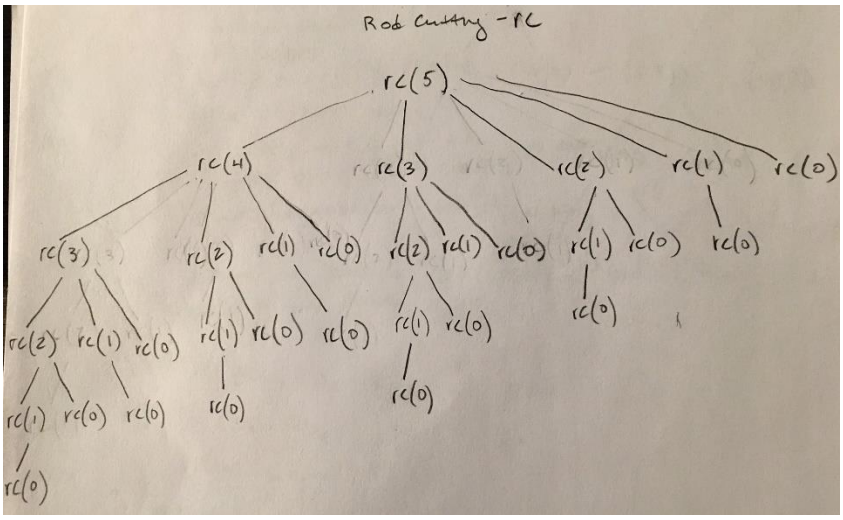
To memorize this problem we would essentially use the structure of the recursive function but we add a 2D matrix or table that we can store values in. This allows us to lookup a result before calling the function recursively every single time, saving time by not repeating work that has already been done.

g) Bottom-up solution

See GlassFalling.java file

2. Rod Cutting

a) Recursion Tree for rod of length 5



b) Counter example

The counter example demonstrates that a greedy approach does not always find the optimal solution. To do so we establish a density (as the book calls it) or a price per inch for a rod: Pi/i

where i is the piece of the rod cut. To start, the greedy strategy is to cut the maximum density first, followed by the length n – i, where n is the length of the remaining rod.

Lets let the length (n) = 5 and the price (pi) = 1, 10, 12 and density (Pi/i) =1, 5, 4

If we have a rod with length 5, under the greedy method of taking the greatest price per inch, we cut a piece of length 2 for $10 and now have 3 inches left. We cut the 3 inches into pieces of length 2 and 1 from there and we have a total of $21. To show this is not the optimal strategy, had we just cut the rod one time, leaving one 2 inch piece and one 3 inch piece we would have $22. This shows the greedy solution is not the optimal solution.

c) Memoized recursive solution

See RodCutting.java file

d) Bottom-up solution

See RodCutting.java file