Asim Shariff CSC 323

Dynamic Programming Project

Glass Falling Problem

a) We can use Dynamic Programming to solve for the minimum number of trials that it would take to determine how high a pane of glass can survive after being dropped from a certain number of floors and landing on our shock-proof mat. Let n represent the number of floors that we are testing and let m represent the number of panes of glass. If we are testing only 1 floor, then the minimum number of trial must be 1. If we only have 1 pane of glass, we must start from the first floor and work our way up to the n^{th} floor. Doing it this way, will give us the most accurate results, since if we randomly choose a floor to test the glass on and the glass shatters on that floor, then we wouldn't know the exact number of floors the glass survived, since it could have shattered on any of the previous floors. So, the minimum number of trials when we are testing 1 pane of glass is equal to the number of floors. Say we are testing the glass on the i^{th}floor, there are only two outcomes at every floor: either the glass survives or it shatters. If the glass survives, we know that any floor below the i^{th} floor would also survive, so we can test the floors from n to i. If the glass shatters, then we know the glass would shatter on any floor above the i^{th} floor, so we can test the floors from 1 to i. So, on every floor i from 1 to the n^{th} floor, there are two outcomes. We don't know what outcome will occur on a floor i , and we want to find the minimum number of trials, so we can take the maximum value of the two outcomes. Taking the maximum value will guarantee that it will take a minimum of those amount of trials in the worst case. So, every floor has the same subproblem, find the minimum number of trials on the floors below the i^{th} floor, if the glass shatters, or find the minimum number of trials from the i^{th} floor to the n^{th} floor, if the glass survives. These subproblems can be solved the same way as the overall problem, so this shows that this problem has an optimal substructure.

b)

Asim Shariff

n = # of floors

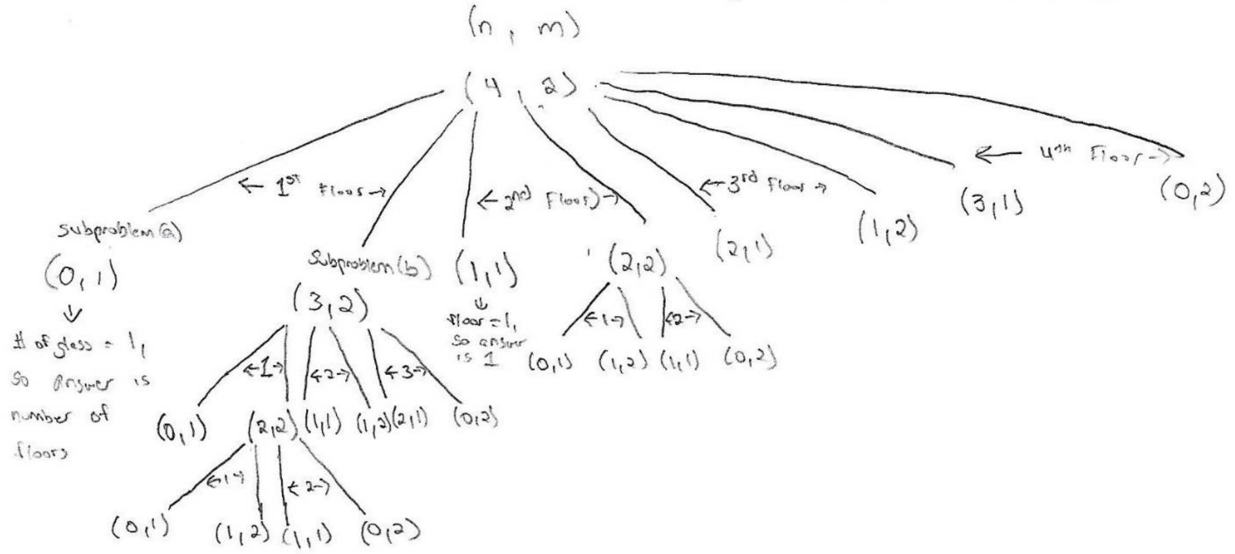
m = # of panes of glass

Each Floor is divided into Two subproblems:

a) [(current floor − 1), (m − 1)]

b) [(n − current floor), (m)]

If floor = 0, return floor.
If floor = 1, return floor.
If glass = 1, return # of floors.



Distinct Subproblems!

1) (0,1)
2) (1,1)
3) (1,2)
4) (2,1)
5) (2,2)
6) (3,1)
7) (0,2)
8) (3,2)

c) Code is in GlassFalling.java

d) When given 4 floors and 2 panes of glass, we have 8 distinct subproblems. The distinct subproblems are: (0,1) , (0,2), (1,1) , (1,2) , (2,1), (2,2) , (3,1) , and (3,2).

e) When given n floors and m sheets, there will be (n * m) distinct subproblems.

f) If you look at the recurrence tree of this problem, you notice that there are repetitions. Since there are repetitions, we can memoize this problem. To do this, I would create a memoized array that has the same number of rows as the number of floors + 1, and the same number of columns as the number of glass + 1. I am adding an extra row and column of zeros to use that as a boundary. After creating the array, I would call a helper function, so when I call the memo function recursively, I am not reinitializing the memo array. In the helper function, I would first check if the problem at n^{th} floor and m^{th}number of glass is already solved, if it is not, I would solve it like the recursive problem, but if it was already solved, then I would just return that value. This avoids solving for repetitions in our problems.

g) Code is in GlassFalling.java

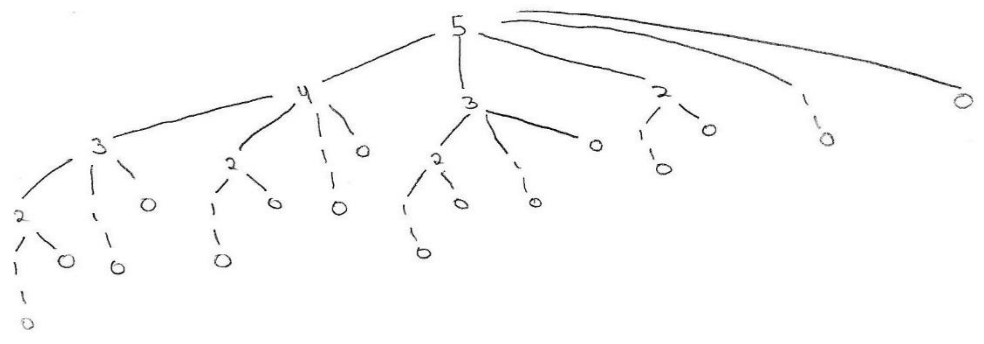
<u>Rod Cutting Problem</u>

a)

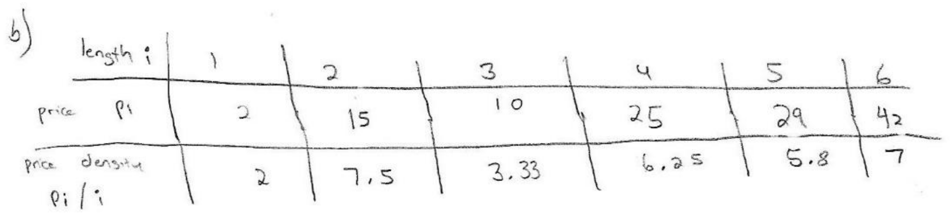
Asim Shariff

a) Rod length = n

At each n-1 spot, we can cut or not cut.



b)



length i	1	2	3	4	5	6
price pi	2	15	10	25	29	42
price density pi / i	2	7.5	3.33	6.25	5.8	7

b) The greedy solution would take the rod with the highest price density first. The price density is calculated by dividing the price by the length. Suppose we are given a rod with the length of 6. In the diagram above, the highest price density would be 7.5, which is 15 for the length of 2. The next highest density that we can have, while staying under or equal to the length of 4 (6-2=4, so the max length we can have and still be under length of 6 would be 4) , would be length 4. The max price according to the greedy algorithm would be 15 + 25, which is 40. This is not correct, since the max price we can actually make would be to take the entire length of 6, since that would give 42. The greedy algorithm would fail, since the price density of 6 wasn’t the highest, so it missed this solution. This is why the greedy algorithm would fail for this problem.

c) Code is in the RodCutting.java

d) Code is in the RodCutting.java