Federico Toscano
Dynamic Programming

Glass Falling

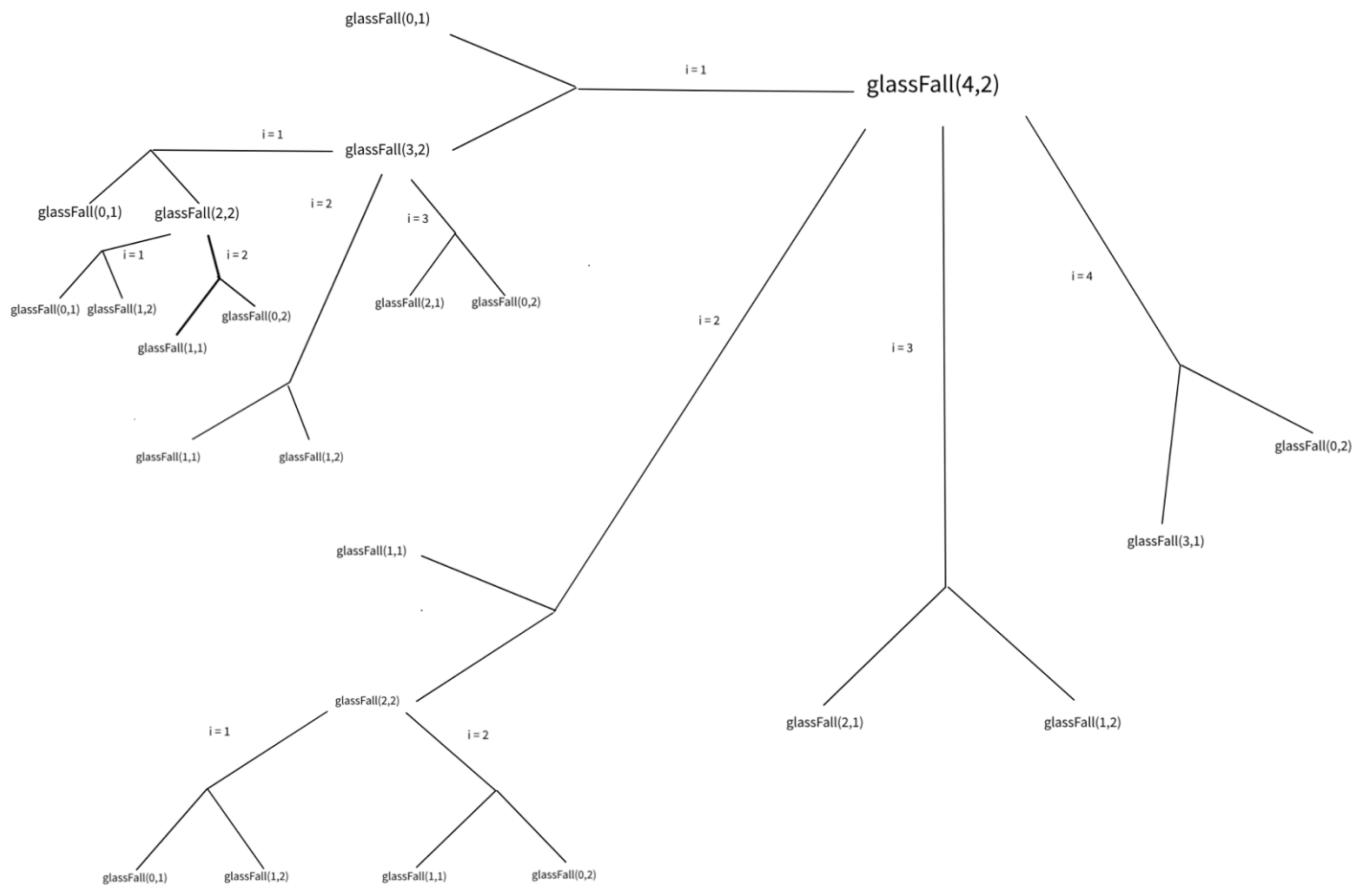
Problem A)

When a sheet of glass is dropped from a floor, 2 possible situations may occur

1. The sheet of glass breaks, which means we must check the floor below, with 1 less piece of glass. The problem is now reduced to n – 1 floors and m – 1 pieces of glass.
2. The sheet of glass does not break, which means the floors from the current floor x to the ground floor do not need to be checked. This reduces the problem to n – x floors to be checked, with m sheets of glass.

Since we want the minimum number of trials in the worst possible case, we have to find the maximum between both cases and then compare it to our current minimum drops. This process is then repeated for each floor, finally resulting in the minimum number of trials necessary to find the critical floor

Problem B)



Federico Toscano
Dynamic Programming

Problem D)

There are 8 distinct subproblems to solve when floors = 4 and sheets = 2. Below are the unique subproblems.

(0, 2)	(1,1)	(1,2)	(2,1)	(2,2)	(3,1)	(3,2)	(4,2)

Problem E)

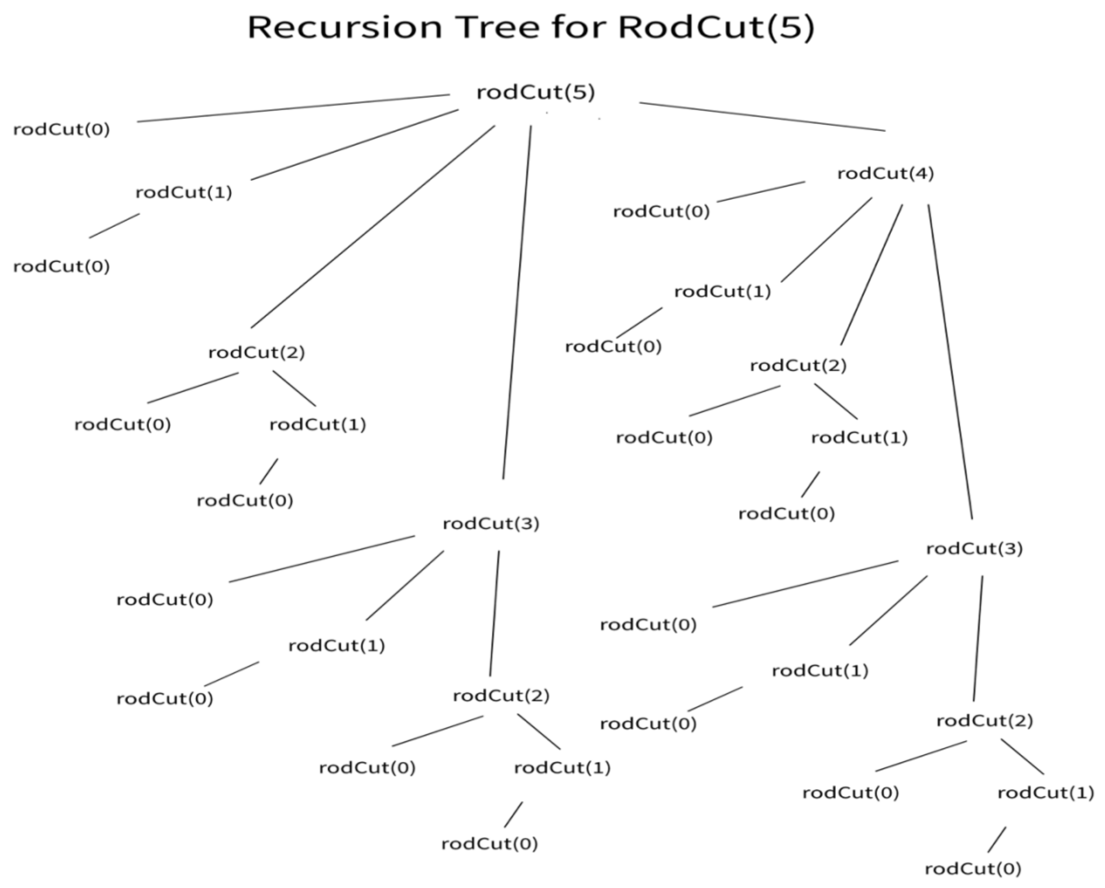
n * m unique subproblems will need to be calculated. Each result from the sub problem (n, m) represents the minimum number of trials necessary to find the critical floor with n sheets of glass and m floors.

Problem F)

In order to memoize glassFallingRecur an array of size n * m would need to be created to cache the results. Initially each element in the array would be given a very large value (m + 1 or infinity). Then the recursive helper would be called, being passed n, m, and the array containing the cached results. When a subproblem needs to be solved, the cached results are checked first and only if the cached result = the large value intially inserted into it, will we recursively solve for the sub problem.

Rod Cutting

Problem A)



Federico Toscano
Dynamic Programming

Problem B)

The greedy strategy can occasionally produce an incorrect result when computing the best price for a rod.
Eg. Consider the following table, listing the price per rod length, and density of the rod length

Length of Rod (i)	1	2	3	4
Price of Rod p_i	2	13	21	24
Density of Rod p_i / i	2	6.5	7	6

The greedy algorithm would first choose to cut a rod of length 3 for a price 21, and then a rod of length 1 for a price of 2 resulting in a price of 23 (21 + 2). However the the maximum value the can be obtained, only occurs when you cut 2 rod's of length 2 for a price of 26 (13 + 13).