Jesse Rosenthal
Professor Yao
CSCI 323
March 30, 2019

A)

Glass Falling Recursive Optimization

We introduce the expression C(s , n) where s equal the numbers of glasses and n equal the number of floors. There are two cases corresponding to each floor.

Case 1 : The glass brakes.

Case 2: The glass does not brake.

Say we have n=1, n=2 …. n = k where n=1 extends case: 1 C(1,0) on the left side. Also we notice that C(1,0) has no floors so it will remain (1,0). Case 2: C(2,3) extends on the right side of n=1. We use recursive call function GlassFallingRecur(int n numFloors, int m numGlass) on n=1 where C(2,3) now extends case 1: C(1,0) from the left side and case 2: C(2,2). So if we observe closely we notice there are some repeated function calls. Case 1: C(1,0) is repeated twice and this will occur as C(1,0), C(1,0), C(1,0)…..C(1,0)k given that the program continues. Also, say if n=2 extends case 1: C(1,1) on the left side and case 2: C(2,2) on the right side. Once again, we use another function call GlassFallingRecur(int n numFloors, int m numGlass) we notice that we would have (2,2) repeating twice.

Solution:

So to prevent repeated cases would be to utilize a two dimensional array where we can store the corresponding solutions in the corresponding spaces. S represents rows and n represents columns.

CODES:
C) GlassDropTop.java

G) GlassDropBottom.java

B)

GRAPH TREE

RECURSION

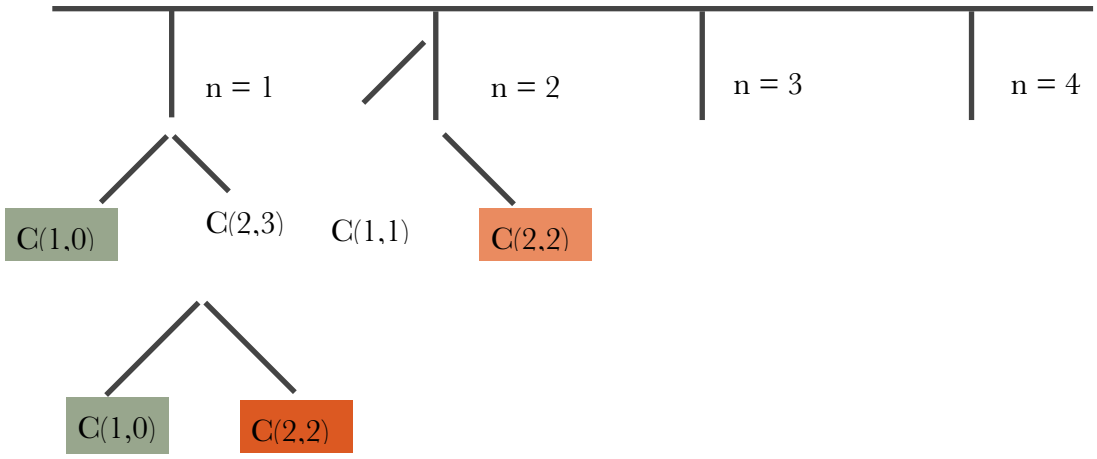
initial (floor-1), (n-1)

(m-floor), (n)

case 1: when glass is equal to 1 then return the number of glass

case 2: when the floor is equal to 0 return floor

case 3: when the floor is equal to 1 return floor



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

Answer = 8

E) How many distinct problems for n floors and m sheets?

Answer = There will be (n*m) for n floors and m sheets that satisfy the condition of the cross product.

F)

Memorize GlassFalling

Description

We could use the GlassFalling code without the process of memorization and it would work. We would get GlassFalling(n) = GF(n-1) + GF(n-2) …. a. However, if we test the runtime we'd probably notice that there is a slight discrepancy. As numerous function calls get executed and the data tree forms on the right side to the left there will be duplicate data reoccurring. Duplicate data will cause a decrease in the accuracy of the run time with in the program and could have 1*2*2*2*…..O(2^n) for the tree. So our solution would be to store the data results one by one in an array. So, now our array has been created such that when we know that data has already been computed there is no need to duplicate it; we just return the stored value. Now we get a linear runtime which is O(n). Memorization of any code is the process of returning stored values.