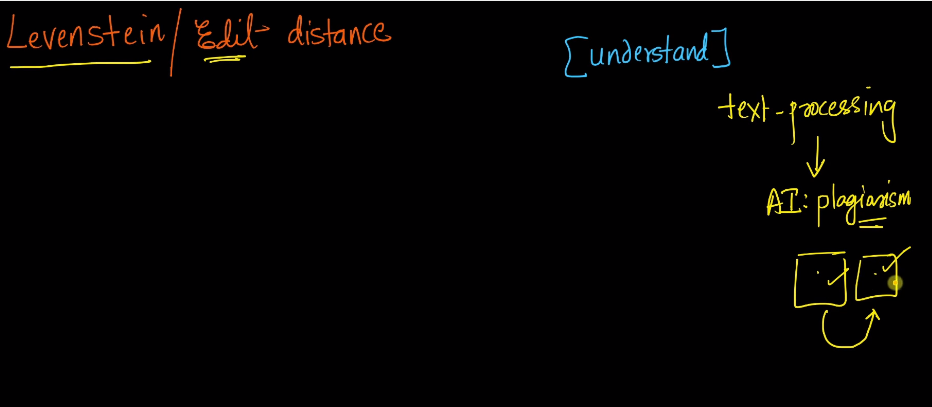
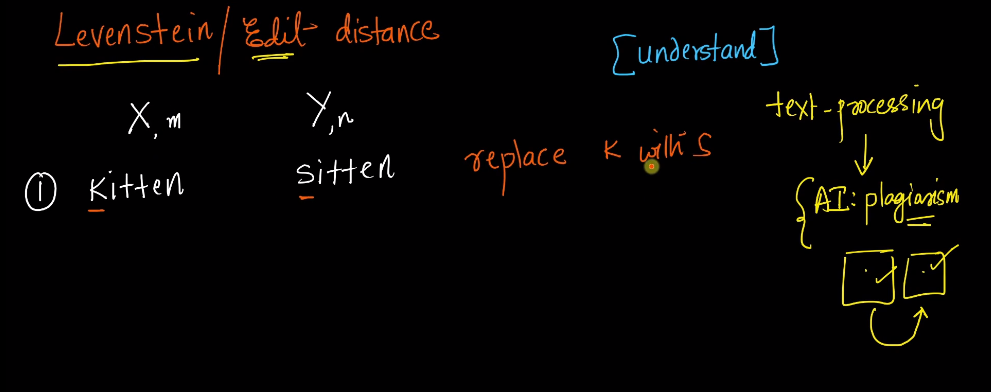
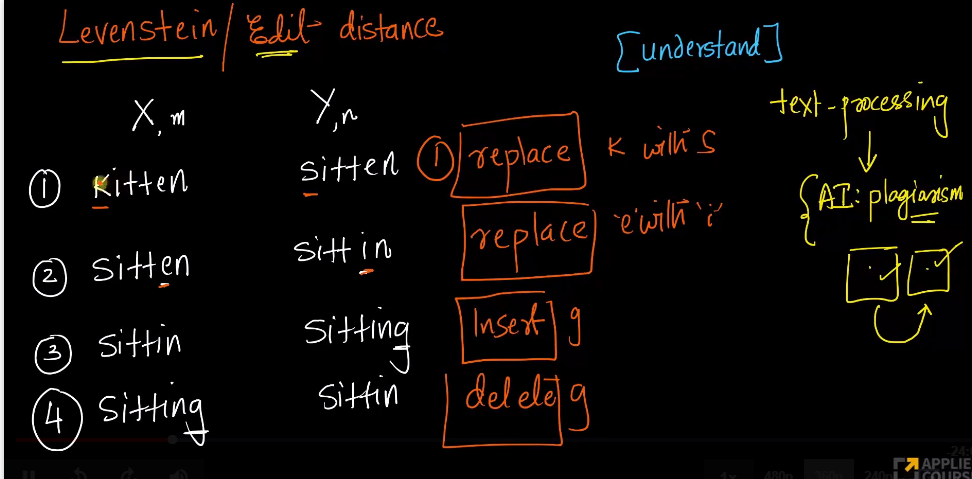
**Levenstein/Edit Distance Problem**



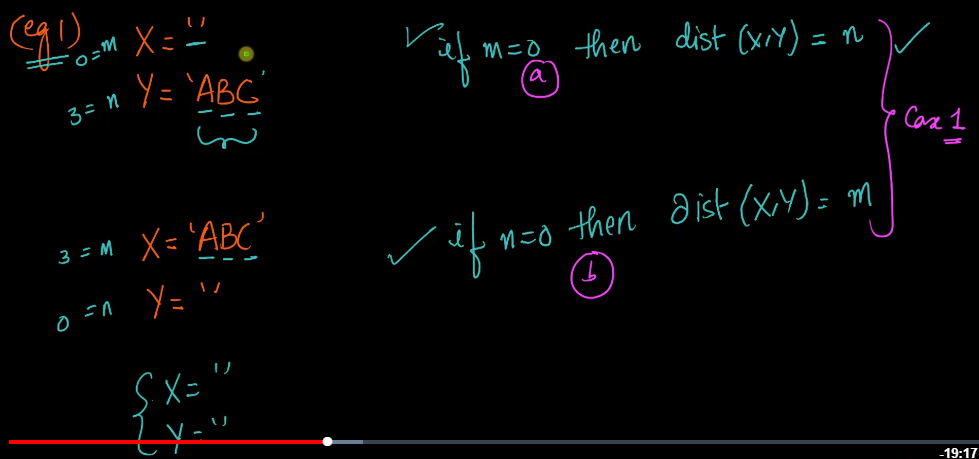
Between two files, if there is a small change required to arrive at file 2, then we can say that two people have copied the code. It is called as Plagiarism.

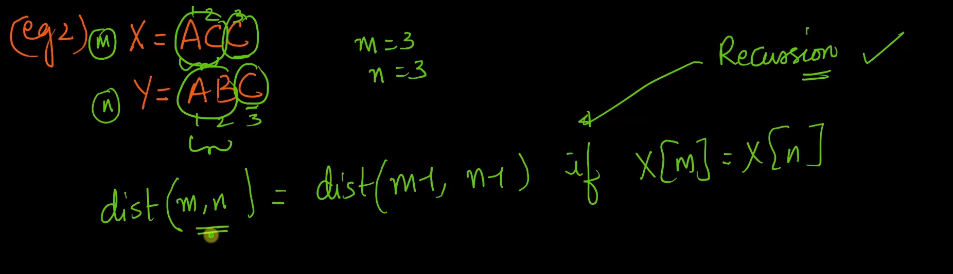


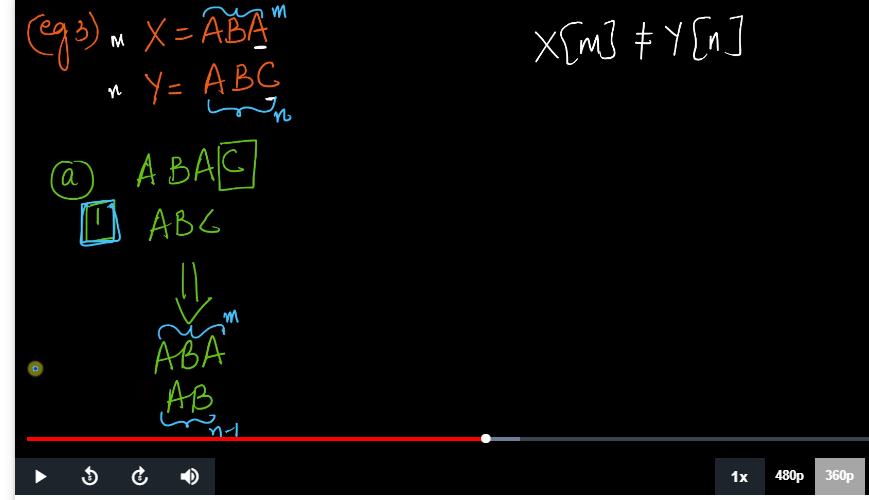
Only with one operation i can get Y from X.



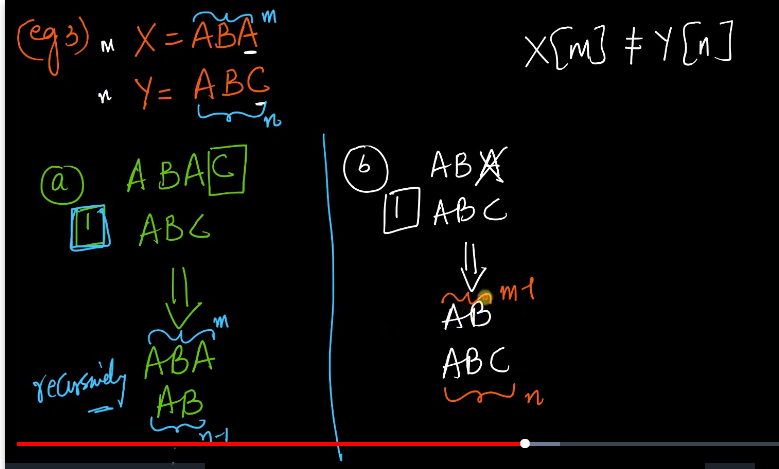




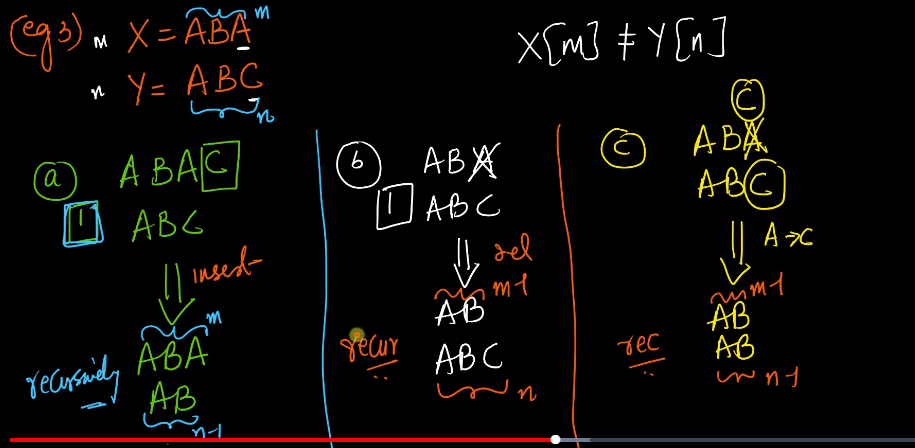




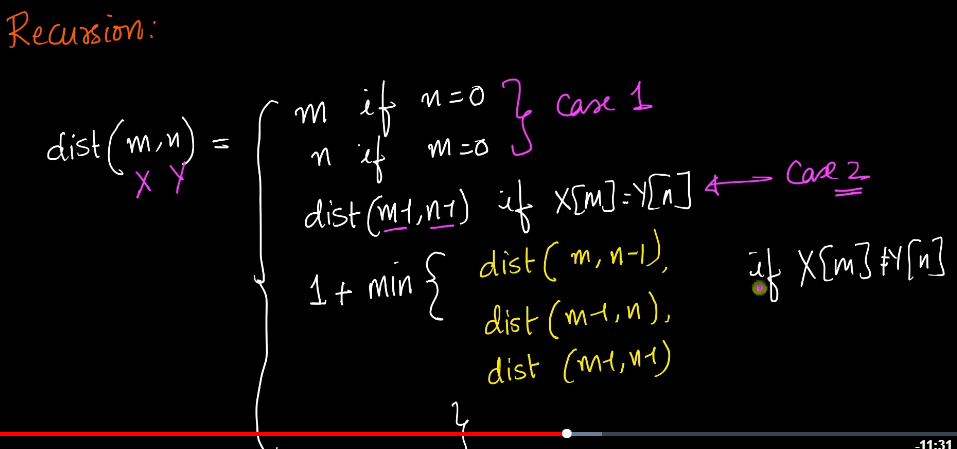
If last character is not matching, then we have to insert one character in first string so that the last chaacter will match and then we can perform recursion on the rest of the items.

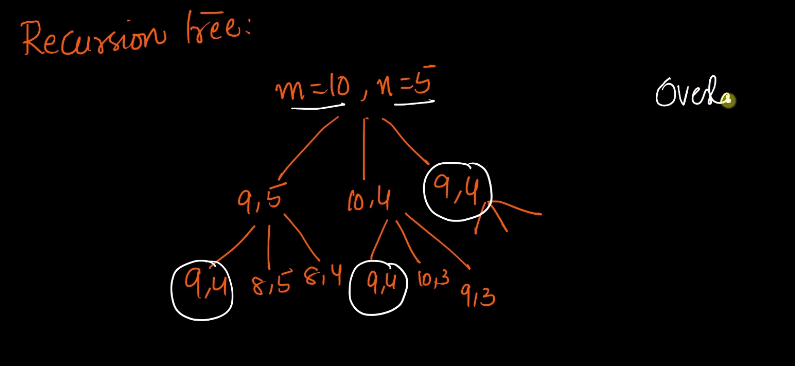


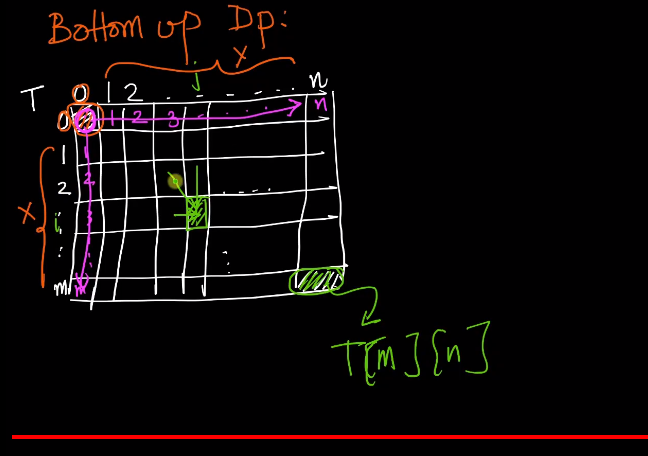
In second case, we are deleting one character here.

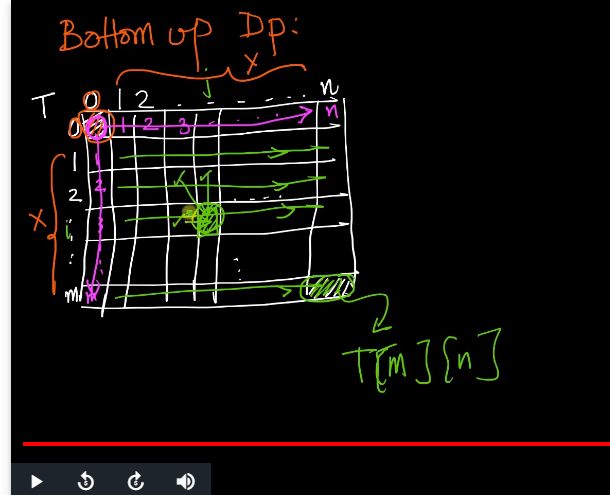


In third case i can substitue it.

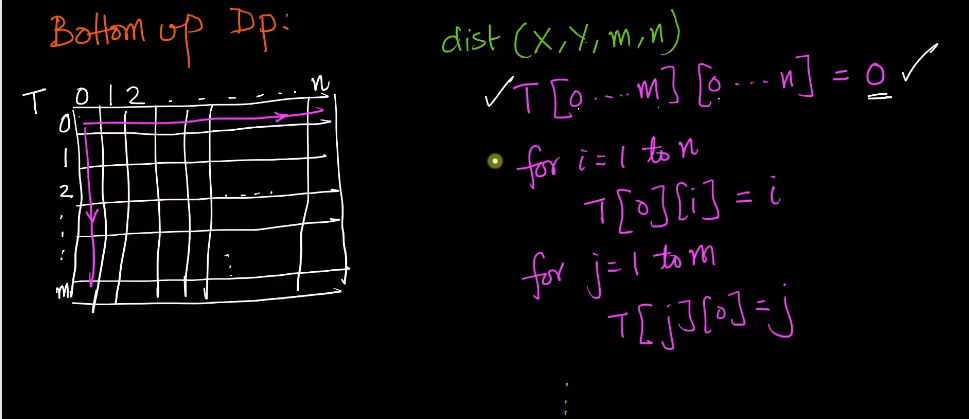


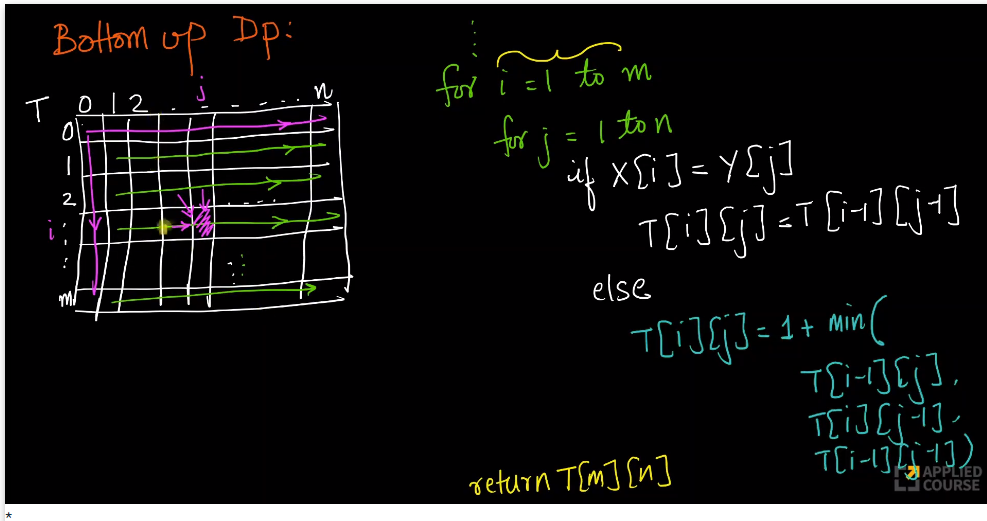






Here we are filling in row by row order. So that everything that i have in previous row is available.





#Recursive Solution

def minDistance(s1,m,s2,n):

if m==0:

return m

if n==0:

return n

if s1[m-1]==s2[n-1]:

cost=0

else:

cost=1

return min(minDistace(s1,m-1,s2,n)+1

,minDistace(s1,m,s2,n-1)+1,

minDistace(s1,m-1,s2,n-1)+cost)

if \_\_name\_\_=='\_\_main\_\_':

print(minDistance("sitting",7,"kittens",7))

#dynamic Programming

def minDistanceDynamic(s1, s2):

x = len(s1) + 1

y = len(s2) + 1

A = [[-1 for i in range(x)] for j in range(y)]

for i in range(x):

A[0][i] = i

for j in range(y):

A[j][0] = j

for i in range(1, y):

for j in range(1, x):

if s1[j- 1] == s2[i - 1]:

A[i][j] = A[i - 1][j - 1]

else:

A[i][j] = min(

A[i - 1][j] + 1,

A[i][j - 1] + 1,

A[i - 1][j - 1] + 1

)

return A[y - 1][x - 1]

if \_\_name\_\_=='\_\_main\_\_':

print(minDistanceDynamic("sitting","kittens"))

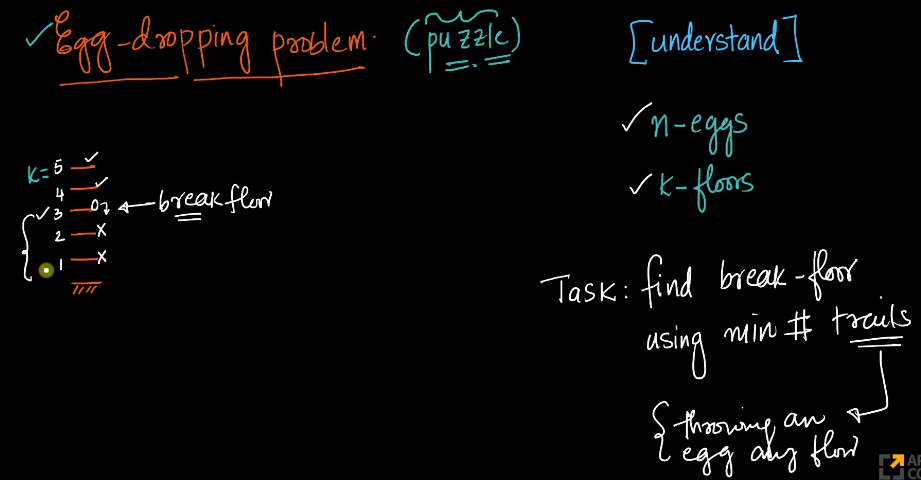
# Egg dropping Problem

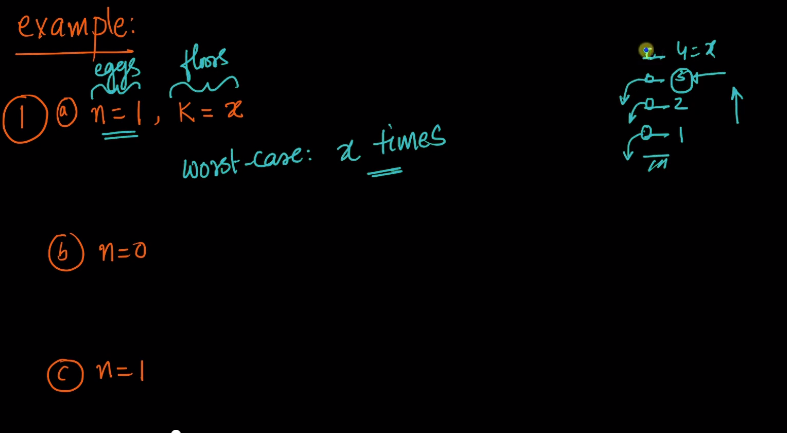
The following is a description of the instance of this famous puzzle involving n=2 eggs and a building with k=36 floors.

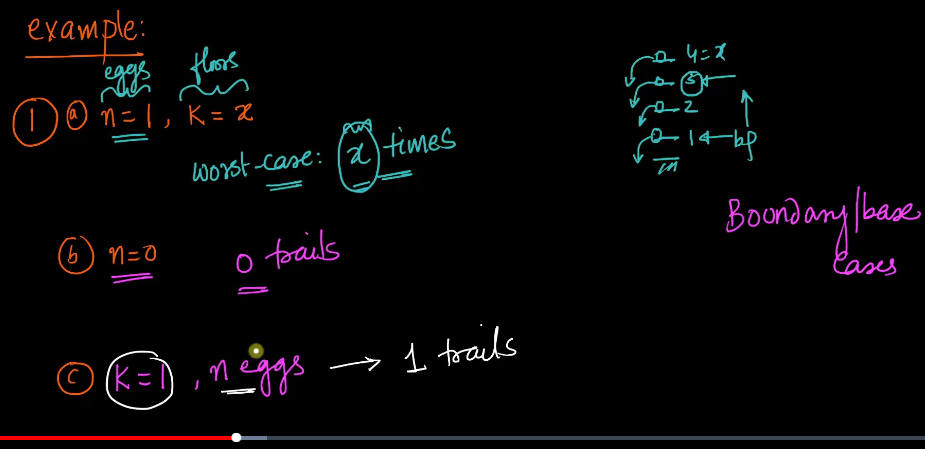
Suppose that we wish to know which stories in a 36-story building are safe to drop eggs from, and which will cause the eggs to break on landing. We make a few assumptions:

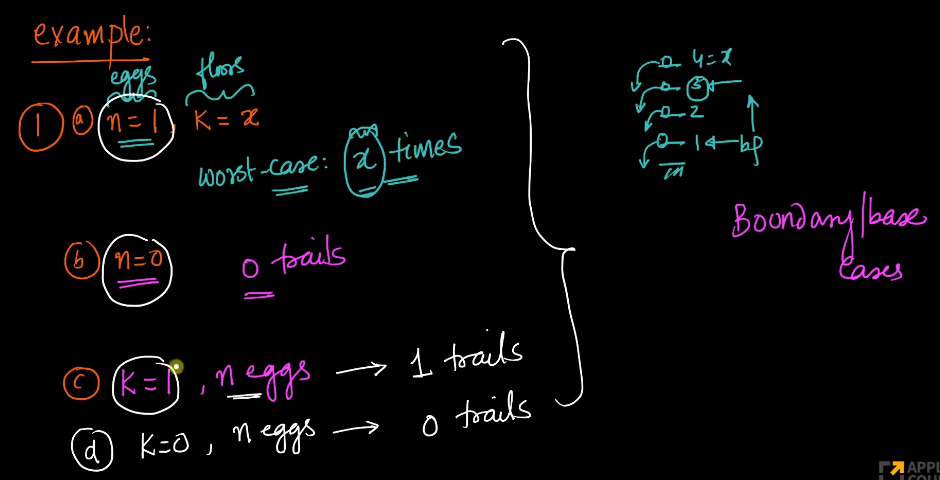
…..An egg that survives a fall can be used again.  
…..A broken egg must be discarded.  
…..The effect of a fall is the same for all eggs.  
…..If an egg breaks when dropped, then it would break if dropped from a higher floor.  
…..If an egg survives a fall then it would survive a shorter fall.  
…..It is not ruled out that the first-floor windows break eggs, nor is it ruled out that the 36th-floor do not cause an egg to break.

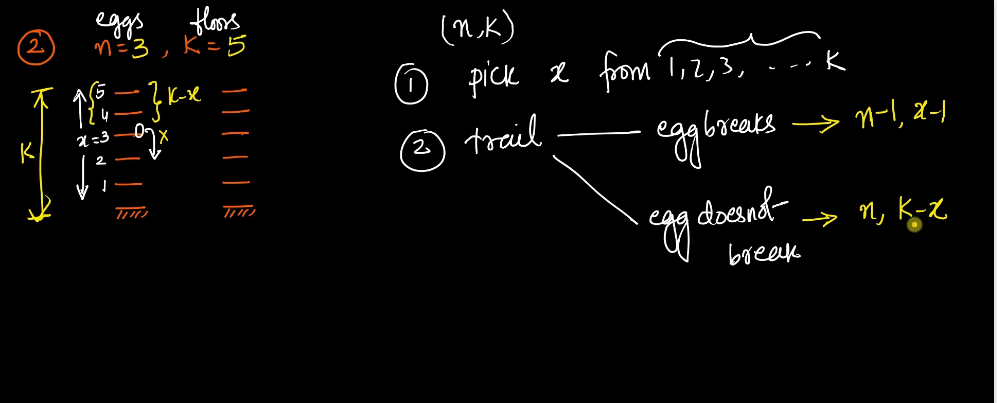
If only one egg is available and we wish to be sure of obtaining the right result, the experiment can be carried out in only one way. Drop the egg from the first-floor window; if it survives, drop it from the second-floor window. Continue upward until it breaks. In the worst case, this method may require 36 droppings. Suppose 2 eggs are available. What is the least number of egg-droppings that is guaranteed to work in all cases?







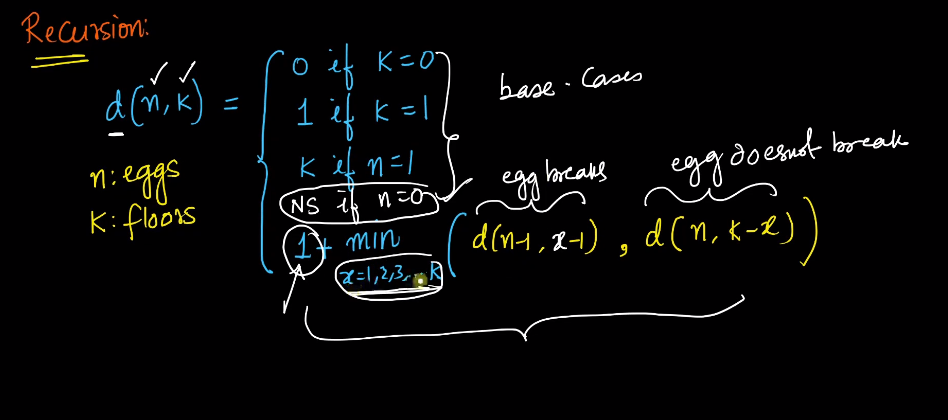


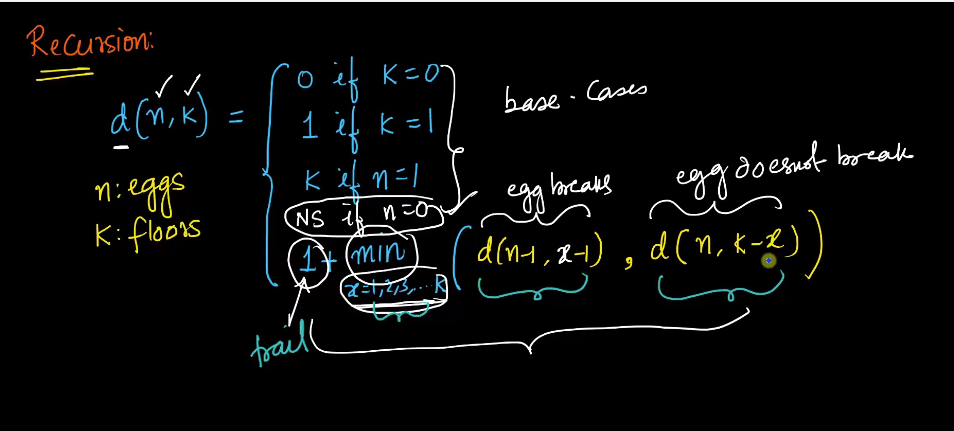


If the egg breaks then we will be left with n-1 eggs and the optimal floor will be less than the current floor that we are at.

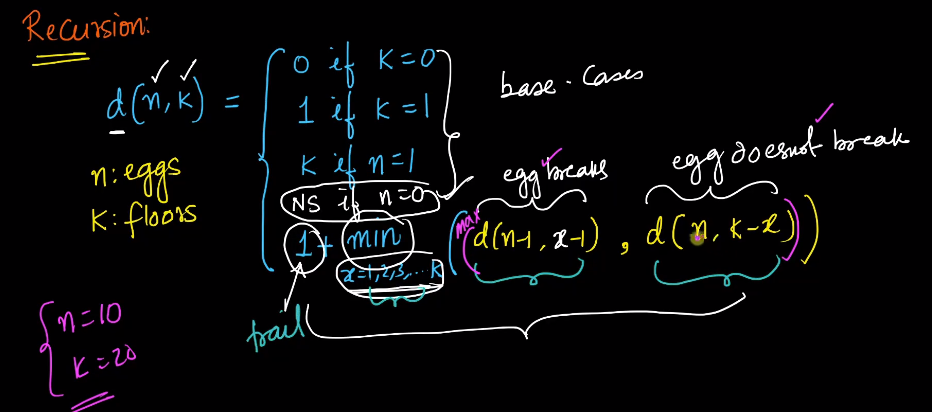
but if egg doesnt break then we will still have n eggs and k-x floor to try.

Here instead of taking one egg, we will take all the eggs because we dont know which is the best case.

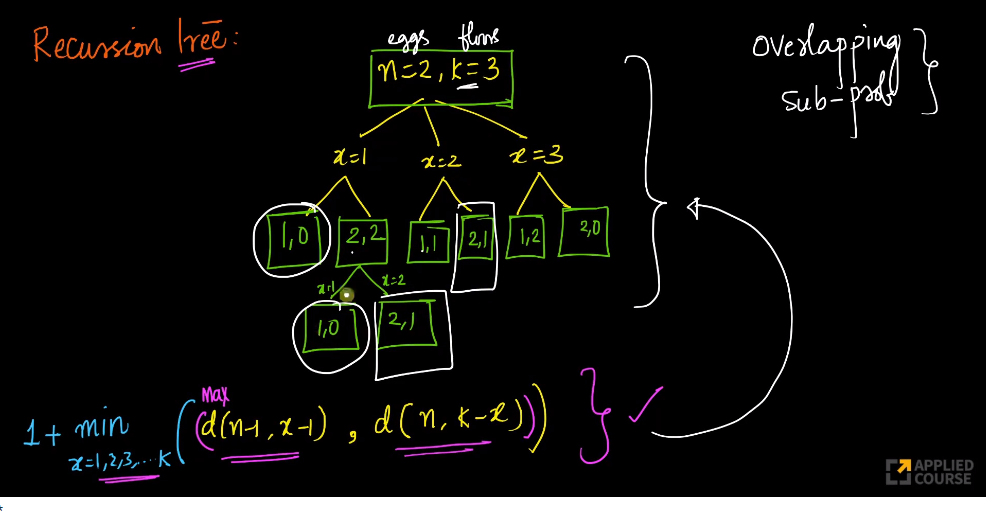


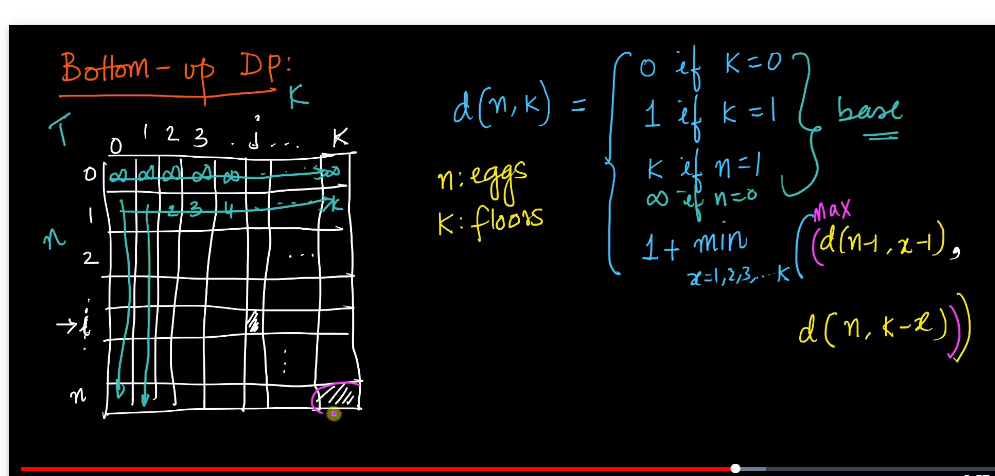


here 1 is the trial that we have to do.



Then we need to add Max here as we need to consider the worst case scenario. We dont know here whether my egg breaks here or not.





Here we have filled this using our base case.

#recursive Approach

import sys

def eggDropsRecursion(eggs,floor):

if floor==0 or floor==1:

return floor

if eggs==1:

return floor

minvalue=sys.maxsize

for i in range(1,floor+1):

temp=max(eggDropsRecursion(eggs-1,i-1),eggDropsRecursion(eggs,floor-i))

if temp<minvalue:

minvalue=temp

return minvalue+1

if \_\_name\_\_=='\_\_main\_\_':

print(eggDropsRecursion(2,10))

#dynamic Programming Appraoch

import sys

def eggDrops(eggs,floor):

dp=[[0 for x in range(floor+1)] for x in range(eggs+1)]

for i in range(1,eggs+1):

dp[i][1]=1

for i in range(1,floor+1):

dp[1][i]=i

for i in range(2,eggs+1):

for j in range(2,floor+1):

dp[i][j]=sys.maxsize

for k in range(1,j+1):

temp=1+max(dp[i-1][k-1],dp[i][j-k])

dp[i][j]=min(temp,dp[i][j])

return dp[eggs][floor]

if \_\_name\_\_=='\_\_main\_\_':

print(eggDropsRecursion(2,10))