

How to solve the peg-jumping problem in general?

memo[board B] ← lookup table for boards - returns "true" or "false"  
Set all memo[] to false  
done[board B] ← lookup table to say whether this board has been solved yet.  
Set all done[] to false  
Can solve (B) {

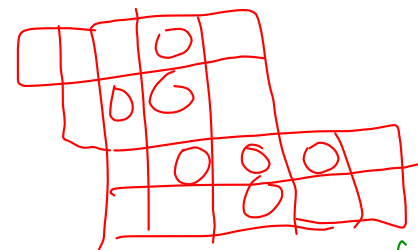
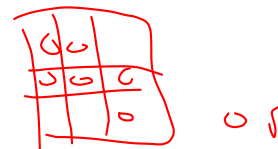
if done[B] = true  
then return memo[B]

else  
for each possible move, m  
if Can solve (B after move m)  
memo[B] = true

done[B] = true  
return memo[B].

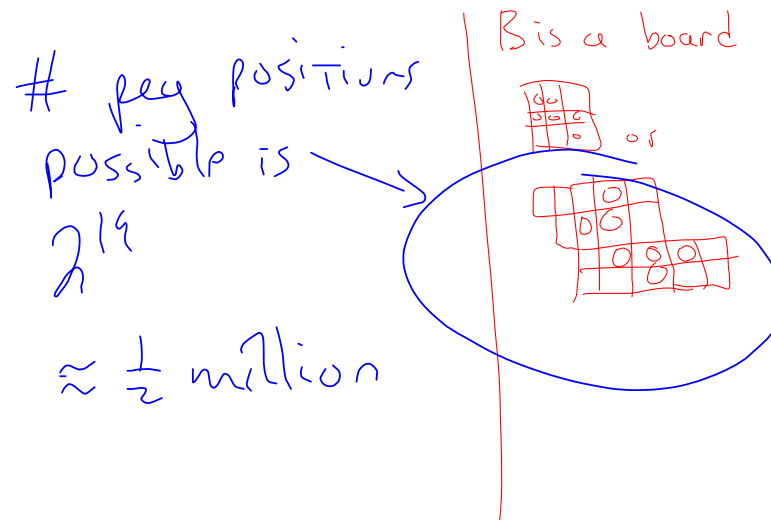
}

B is a board



= Playing surface  
+ pegs in  
locations

Memoization is Recursion with memos remembering which subproblems have been solved, what the solution was, and checking the memos before ~~recursive calls~~. trying to solve a given subproblem.



Dynamic Programming is like memorization

except : \* do smaller cases first, so that  
when you are considering a problem,  
all of its subproblems will  
already have been solved.

Build table  
first

\* Done iteratively, NOT

for  $i \leftarrow 1$  TO 18      recursively.

for all boards  $B$  with  $i$  pegs

for all moves  $m$  on  $B$

check if ( $B$  after  $m$  is solvable)

if so,  $B$  is solvable.

SET  $memo[B]$  TO TRUE.

Can Solve ( $B$ ) {  
return  $memo[B]$

~~return~~  
Lookup answer  
in table.

# Longest Common Substring Problem

Given two strings, find the longest (not necessarily contiguous) substring that they have in common

Ex.

a b c c b c b b a c b b a

c c b b a b b a c a c

a b c c h c b b a c b b a  
c c b b a b b a c a c