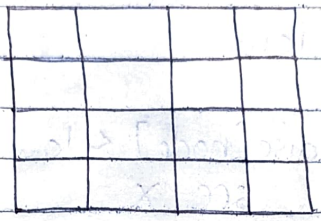
Page :
Date : / /

Day - 222

Backtracking

* **N - Queen:**



N = 4
N - Queen

⇒ We have a N×N chessboard.

⇒ And we have a N-Queen.

⇒ We have to place all the Queens in the chessboard as there will be ~~no~~ no two queen are in the same col, row & diagonal.

⇒ Backtracking is like a brute force approach that we do for solving any problem.

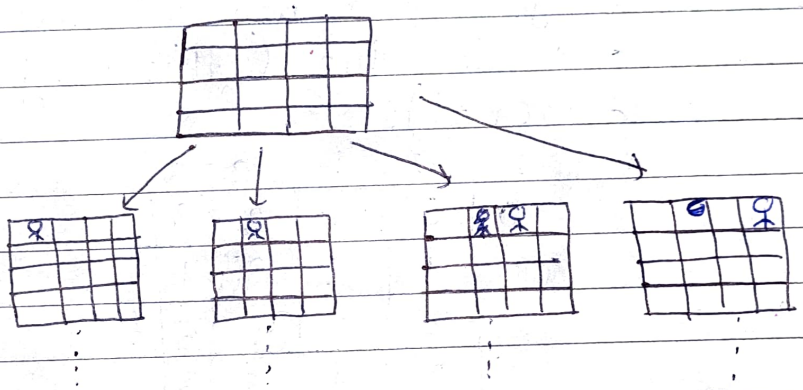
⇒ We will check the queen ~~the~~ location by putting it every cell in the chessboard by following all the rules.

⇒ If we can't be put a queen in a row then we simply change the position of previous queen.

⇒ Now, How to solve this question —

⇒ we will just try all the ~~s~~ options ~~of~~ available.

⇒



⇒ Now, we will again do a function call for next row with valid position.

⇒ We will don't create this chess board again & again instead we will take the reference.

⇒ Also, when we go back, we will ~~co~~ clear the current position.

⇒ Also, we will take a column array. So, that we don't have to

Page:
Date: / /

check for the whole column value.

=> Just check this col array.

=> Now, it's time for optimization.

=> We can see that, we have to check diagonal everytime.

=> So, for this, we will take an array and do this (this is for right diagonal).

	0	1	2	3
0	0	1	2	3
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6

=> We will do the sum of row & column.

=> We know that if we put any queen then we can't put any other queen in the diagonal.

=> So, suppose, we we put queen in the (1, 2) position then we can't put any other queen.

=> That means no other queen can't be put in the $1+2=3$ sum cell.

⇒ Sowe will take an array & put 1 in the sum place.

⇒ And for any other queen, we have to just do the sum & check in the array.

⇒ For the left diagonal, we will do col - row.

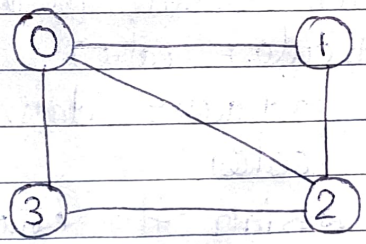
⇒ But this can also give -ve values.

⇒ So, we will add (n-1) into it.

⇒ So, our formula become as —

$$(n-1) + (col - row)$$

* M-coloring Problem:

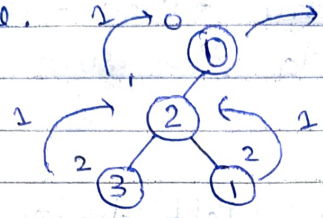


m = 3

⇒ Here, we have to color the nodes of the graph in this way that no two adjacent nodes get the same color.

⇒ In our first approach, we will use our intution.

⇒ First, we start from ① and give it a color and move ~~it~~ to its adjacent node. 1 → 0 → 1 → Successfully colored



⇒ When we can't color any node, we will back track and change the color of the node.

⇒ This is the most intuitive approach.

* Pseudo Code:

⇒ Check each color there —
- ↳ Particular color
  - ↳ adjacent doesn't have that color.
    - → Assign the color to that node.
    - → Go to the adjacent node those who don't have color.
    - → If all the adjacent get the color, return 1.
    - → else → check for another color.

→ return 0.

T.C. → $O(m*n)^n$

⇒ 2nd Method:

⇒ We will color all the nodes and check it is valid or not.

⇒ If not then change the colors.

⇒ We will directly give color and check if it is valid or not. Then go to adjacent node.

* Sudoku Solver:

⇒ A sudoku is a 9x9 grid where we have to fill no. from ~~1x~~ 1 to 9.

⇒ But there are some conditions —

→ No no. will repeat in row & col.

→ And also in the subgrid of 3x3 size.

⇒ We use our intution to solve this problem.

⇒ We will check for valid no. then if it is valid for that position, we will fill & if it is not then we backtrack & change the previously filled cells.

=> we can easily check for row & col. but how we can check for block.

=> So, for every no. in that block we will start ~~with ther~~ checking with their starting row & starting col ~~of~~ of that block.

=> For ex: for 1st block —
0 row & 0 col.

=> 2nd block —
~~0~~ 3 row & 3 col.

=> So, if a no. has coordinates i.e.— (2,3) then it will start checking — 0 row & 3 col.

row ~~col~~	col ~~row~~
0,1,2 → 0	0,1,2 → 0
3,4,5 → 3	3,4,5 → 3
6,7,8 → 6	6,7,8 → 6

(i,j)
(i/3) * 3 → row
(j/3) * 3 → col

Page:
Date: / /

⇒ if ~~i==~~ j== 9 then (i+1,c)

=1 if i==9 then return 1.

k.

e.—