class Solution:

def solveNQueens(self, n: int) -> List[List[str]]:

state= [["."] \* n for \_ in range(n)] # start with empty board

res=[]

# for tracking the columns which already have a queen

visited\_cols=set()

# This will hold the difference of row and col

# This is required to identify diagonals

# specifically for diagonals with increasing row and increasing col pattern

# example: square (1,0) = 1-0 = 1

# squares in same diagonals will have same difference

# example: squares (0,0) and (8,8) are in the same diagonal

# as both have same difference which is `0`

visited\_diagonals=set()

# This will hold the sum of row and col

# This is required to identify antidiagonals.

# specifically for diagonals with increasing row and decreasing col pattern

# the squares in same diagonal won't have the same difference.

# example: square (1,0) = 1-0 = 1

# squares in same diagonals will have same difference

# example: squares (0,7) and (1,6) are in the same diagonal

# as both have same sum which is `7`

visited\_antidiagonals=set()

def backtrack(r):

if r==n:

res.append(["".join(row) for row in state])

return

for c in range(n):

diff=r-c

\_sum=r+c

# If the current square doesn't have another queen in same column and diagonal.

if not (c in visited\_cols or diff in visited\_diagonals or \_sum in visited\_antidiagonals):

visited\_cols.add(c)

visited\_diagonals.add(diff)

visited\_antidiagonals.add(\_sum)

state[r][c]='Q' # place the queen

backtrack(r+1)

# reset the path

visited\_cols.remove(c)

visited\_diagonals.remove(diff)

visited\_antidiagonals.remove(\_sum)

state[r][c]='.'

backtrack(0)

return res