Csp

class NQueens:

def \_\_init\_\_(self, n):

self.n = n

self.solutions = []

self.board = [['.' for \_ in range(n)] for \_ in range(n)]

# Branch and Bound helpers:

self.columns = [False] \* n

self.left\_diagonal = [False] \* (2 \* n - 1)

self.right\_diagonal = [False] \* (2 \* n - 1)

def solve(self, row):

if row == self.n:

# Store solution

solution = [''.join(self.board[i]) for i in range(self.n)]

self.solutions.append(solution)

return

for col in range(self.n):

if (not self.columns[col] and

not self.left\_diagonal[row - col + self.n - 1] and

not self.right\_diagonal[row + col]):

# Place Queen

self.board[row][col] = 'Q'

self.columns[col] = True

self.left\_diagonal[row - col + self.n - 1] = True

self.right\_diagonal[row + col] = True

# Recurse to next row

self.solve(row + 1)

# Backtrack

self.board[row][col] = '.'

self.columns[col] = False

self.left\_diagonal[row - col + self.n - 1] = False

self.right\_diagonal[row + col] = False

def print\_solutions(self):

if not self.solutions:

print("No solutions found.")

else:

print(f"\nTotal solutions for {self.n}-Queens: {len(self.solutions)}\n")

for idx, solution in enumerate(self.solutions, 1):

print(f"Solution {idx}:")

for row in solution:

print(row)

print()

# Main Execution

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

n = int(input("Enter the number of Queens (N): "))

queens = NQueens(n)

queens.solve(0)

queens.print\_solutions()

# explanation :for backtracking : • Recursive Search: The solve method is called recursively to #place queens row by row.

#Backtrack: If a safe spot is found for the queen in the current row (row), the queen is placed, #and the algorithm moves to the next row. If no valid position is found in a row, the algorithm backtracks by removing the queen and trying other positions.

# Branch and bound :The solution uses a bounding technique by maintaining three arrays: columns, left\_diagonal, and right\_diagonal.

# These arrays track the positions where queens are already placed in the respective columns and diagonals.

# Before placing a queen, the algorithm checks whether the current column or diagonal is already occupied (not self.columns[col], not self.left\_diagonal[...], not self.right\_diagonal[...]). This step is essentially a bound that prevents the algorithm from trying configurations that are obviously invalid.