**Code:**

def solve\_nqueens(n: int) -> list[list[str]]:

"""

Solves the N-Queens problem using backtracking.

n: The size of the board (N x N).

Returns:

A list of all distinct solutions, where each solution is a list

of strings representing the board (e.g., "Q..." for the first row).

"""

# Stores the results

solutions = []

# Sets to track occupied columns and diagonals for O(1) checking

occupied\_cols = set()

occupied\_diag1 = set() # (row + col)

occupied\_diag2 = set() # (row - col)

# A board represented by a list of strings, where each string is a row.

board\_placement = [-1] \* n

def is\_safe(row, col):

"""Checks

if a queen can be placed at (row, col) safely.

"""

# 1. Check column (O(1) using the set)

if col in occupied\_cols:

return False

# 2. Check main diagonal (row + col)

if (row + col) in occupied\_diag1:

return False

# 3. Check anti-diagonal (row - col)

if (row - col) in occupied\_diag2:

return False

return True

def format\_solution(placement: list[int]) -> list[str]:

"""Converts the list of column placements into the required board format."""

formatted\_board = []

for col\_index in placement:

row\_str = "." \* col\_index + "Q" + "." \* (n - 1 - col\_index)

formatted\_board.append(row\_str)

return formatted\_board

def backtrack(row):

"""

Recursive function to try placing a queen in the current 'row'.

The base case is when all N queens have been placed successfully.

"""

# Base case: All N queens are placed

if row == n:

solutions.append(format\_solution(board\_placement))

return

# Try placing a queen in every column of the current row

for col in range(n):

if is\_safe(row, col):

# 1. Place the queen (CHOICE)

occupied\_cols.add(col)

occupied\_diag1.add(row + col)

occupied\_diag2.add(row - col)

board\_placement[row] = col

# 2. Recurse to the next row

backtrack(row + 1) # EXPLORE

# 3. Backtrack (UNDO THE CHOICE)

occupied\_cols.remove(col)

occupied\_diag1.remove(row + col)

occupied\_diag2.remove(row - col)

board\_placement[row] = -1 # Optional: reset for clarity

# Start the backtracking from the first row (row 0)

backtrack(0)

return solutions

# --- Example Usage ---

N = 4

result = solve\_nqueens(N)

print(f"Found {len(result)} solution(s) for N={N}:")

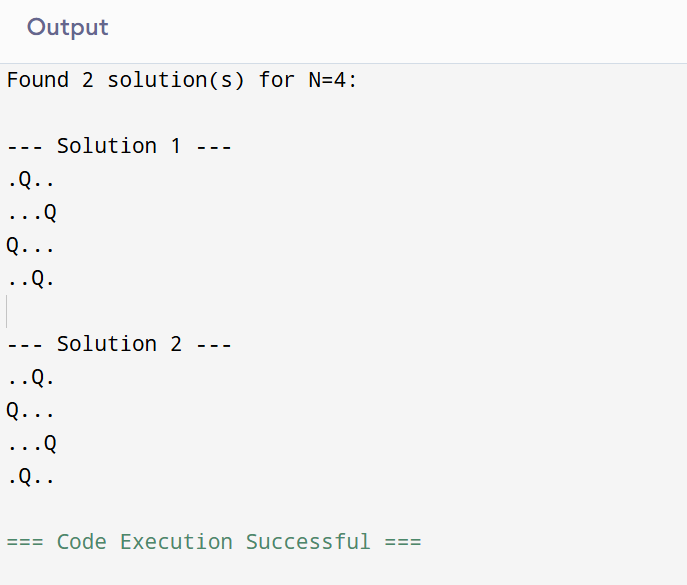
for i, solution in enumerate(result):

print(f"\n--- Solution {i + 1} ---")

for row in solution:

print(row)

**Output:**

****