REPORT ON N-QUEENS PROBLEM

Problem Statement

The N-Queens puzzle is about placing N chess queens on an N×N chessboard so that no two queens attacks each other. This means no two queens share the same row, column, or diagonal.

SUBMITTED BY:

- **Name**: DEVKESH YADAV
- **Roll No.**: 202401100400081
- **Course**: CSE(AI&ML)
- **Date**: 10/03/2025

SUBMITTED TO:

ABHISHEK SHUKLA SIR

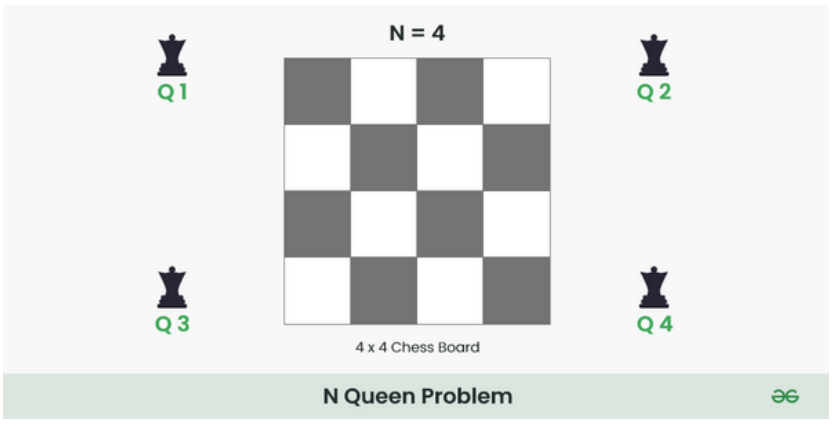
–

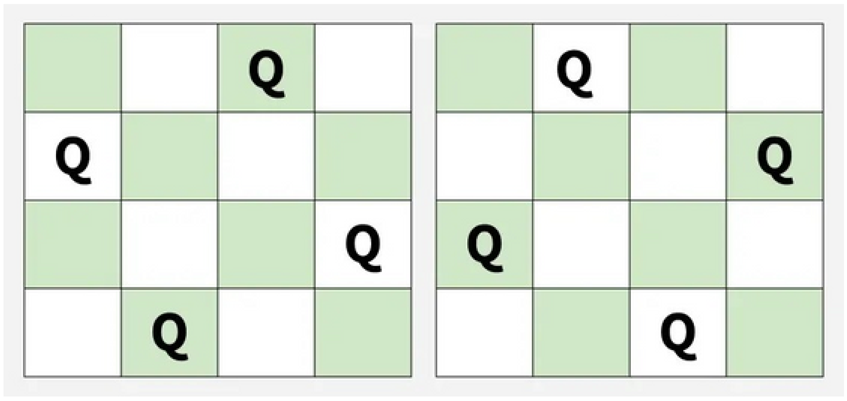
Introduction

What is the N-Queens Problem?

The N-Queens problem is a classic backtracking challenge in computer science and mathematics. A solution is valid if no two queens can attack each other, which means:

1. No two queens are on the same row.
2. No two queens are on the same column.
3. No two queens are on the same diagonal.





Method Overview

We will use **backtracking** to solve the N-Queens problem. Backtracking is a depth-first search (DFS) algorithmic technique.

Steps:

1. **Place a queen** in a safe spot in the current row.
2. **Check if it is valid** to place the queen there.
3. If valid, **move to the next row** and repeat the process.
4. If not valid, **backtrack**: remove the queen and try a different column in the same row.
5. Continue until all rows have a valid queen placement or no valid solution is found.

Code Implementation

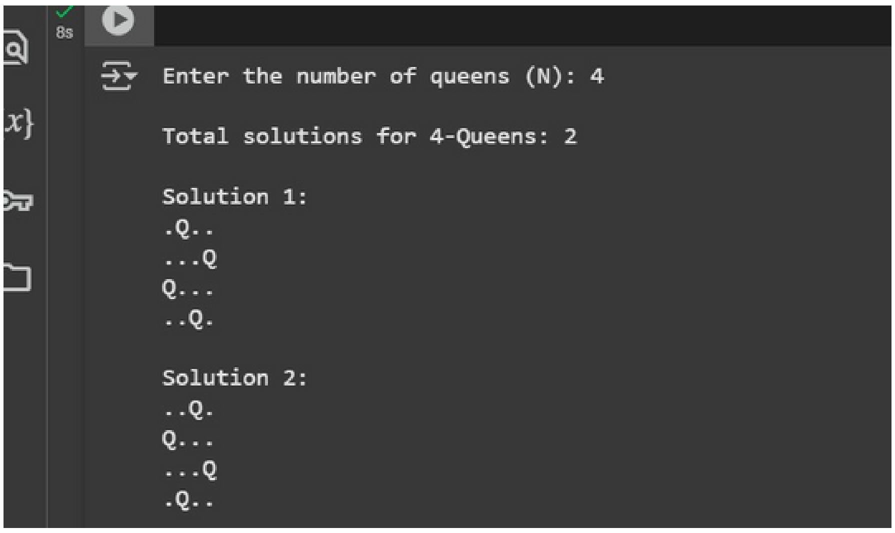
```
def is_position_safe(board, current_row, col, n):
# Check if any queen is in the same column.



  for previous_row in range(current_row):
    if board[previous_row] == col:
          return False

  # Check the upper left diagonal.
  i, j = current_row - 1, col - 1
  while i >= 0 and j >= 0:
    if board[i] == j:
          return False
    i -= 1
    j -= 1

  # Check the upper right diagonal.
  i, j = current_row - 1, col + 1
  while i >= 0 and j < n:
    if board[i] == j:
          return False
    i -= 1
    j += 1

  return True


def place_queens(board, row, n, solutions):
  if row == n:
    solution = []
    for i in range(n):
          row_representation = ["Q" if j == board[i] else "." for j in
range(n)]
          solution.append("".join(row_representation))
    solutions.append(solution)
    return
  for col in range(n):
    if is_position_safe(board, row, col, n):
          board[row] = col
          place_queens(board, row + 1, n, solutions)
          # Implicit backtracking when the loop continues

def solve_n_queens(n):
  board = [-1] * n
  solutions = []
  place_queens(board, 0, n, solutions)
  return solutions

# Sample run for N = 4
N = 4
results = solve_n_queens(N)
print(f"Total solutions for {N}-Queens: {len(results)}")
for idx, sol in enumerate(results, 1):
  print(f"Solution {idx}:")
  for row in sol:
    print(row)
  print()
```

Code Explanation

- **Safety Check:** The is_position_safe function checks if a queen can be placed at a specific location.

- **Recursive Placement:** The place_queens function handles placing queens row by row and backtracks when necessary.
- **Driver Function:** The solve_n_queens function initializes the board and collects all valid solutions.

OUTPUT



```
Enter the number of queens (N): 4

Total solutions for 4-Queens: 2

Solution 1:
.Q..
...Q
Q...
..Q.

Solution 2:
..Q.
Q...
...Q
.Q..
```

References & Credits

- "Eight Queens Puzzle" on Wikipedia.
- Various online resources on backtracking algorithms.
- Images from "geeksforgeeks.org".
- Collage logo "universitykart.com"