In [2]:
```python
def is_safe(board, row, col):
    # Check if no queens threaten the current cell in the same column
    for i in range(row):
        if board[i] == col or \
           board[i] - i == col - row or \
           board[i] + i == col + row:
            return False
    return True

def solve_n_queens(n):
    def backtrack(row):
        if row == n:
            solutions.append(board[:])
            return
        for col in range(n):
            if is_safe(board, row, col):
                board[row] = col
                backtrack(row + 1)

    board = [-1] * n  # Initialize an empty chessboard
    solutions = []
    backtrack(0)
    return solutions

def print_solution(board):
    for row in board:
        line = ['.'] * len(board)
        line[row] = 'Q'
        print(' '.join(line))
    print()

def main():
    try:
        n = int(input("Enter the number of queens (e.g., 4): "))
        if n < 4:
            print("The number of queens must be at least 4.")
            return
        solutions = solve_n_queens(n)
        if solutions:
            print(f"Found {len(solutions)} solution(s) for {n}-queens proble
            for i, solution in enumerate(solutions):
                print(f"Solution {i + 1}:")
                print_solution(solution)
        else:
            print(f"No solution found for {n}-queens problem.")
    except ValueError:
        print("Invalid input. Please enter a valid integer.")

if __name__ == "__main__":
    main()
```

```
Enter the number of queens (e.g., 4): 4
Found 2 solution(s) for 4-queens problem:
Solution 1:
. Q . .
. . . Q
Q . . .
. . Q .

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

In [ ]: