SUPERIOR UNIVERSITY

Submitted By:

Name: **Ali Raza**

Roll number **Su92-bsdsm-f23-018**

Section: **4A**

Task: **04**

Subject: **Programming For Ai (lab)**

Submitted to: **Sir Rasikh Ali**

1. Checking if a Position is Safe

```
def is_position_safe(board, row, col):
    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
```

This function checks if a queen can be placed at position (row, col) without being attacked by other queens.

Conditions Checked:

1. **Same Column:** If any previous queen is already placed in the same column.
2. **Left Diagonal Conflict:** If any previous queen is on the left diagonal (board[i] - i == col - row).
3. **Right Diagonal Conflict:** If any previous queen is on the right diagonal (board[i] + i == col + row).

If **none of these conditions are met**, the function returns True (safe position).

---

2. Solving the N-Queens Problem Using Backtracking

```
def solve_n_queens_dynamic(size):
    def solve(row, board):
        if row == size:
            solutions.append(board[:])
            return
        for col in range(size):
            if is_position_safe(board, row, col):
                board[row] = col
                solve(row + 1, board)
                board[row] = -1
```

This function **recursively** tries to place queens on the chessboard.

Steps:

1. **Base Case:** If row == size, all queens are placed successfully, so **store the board configuration**.
2. **Loop Through Each Column:** Try placing a queen in each column of the current row.
3. **Check Safety:** If safe, place the queen (board[row] = col) and move to the next row (solve(row + 1, board)).
4. **Backtrack:** If placing the queen leads to a dead end, **remove it** (board[row] = -1) and try the next column.

---

3. Initializing and Running the Solver

```
solutions = []

board = [-1] * size

solve(0, board)

return solutions
```

- solutions: Stores all valid board configurations.
- board: A list where board[i] represents the column index of the queen placed in row i. (-1 means no queen placed in that row).
- The solve() function is called with row = 0 to begin the search.

---

4. Displaying the Solutions

```
def display_n_queens_solutions(solutions):
    for solution in solutions:
        for i in range(len(solution)):
            row = ['.'] * len(solution)
            row[solution[i]] = 'Q'
            print(" ".join(row))
        print("\n")
```

This function prints the **N-Queens solutions in a visual format**.

Output for N = 4:

```
. Q . .

. . . Q

Q . . .
```

. . Q .

Each **. represents an empty square**, and **Q represents a queen**.

---

5. Running the Solver for N = 6

board_size = 6

solutions = solve_n_queens_dynamic(board_size)

print(f"Number of solutions: {len(solutions)}")

display_n_queens_solutions(solutions)

- The solver is run for a **6×6 board**.
- The total number of **valid solutions** is printed.

The solutions are displayed using display_n_queens_solutions().

```python
def is_position_safe(board, row, col):
    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_dynamic(size):
    def solve(row, board):
        if row == size:
            solutions.append(board[:])
            return
        for col in range(size):
            if is_position_safe(board, row, col):
                board[row] = col
                solve(row + 1, board)
                board[row] = -1
    solutions = []
    board = [-1] * size
    solve(0, board)
    return solutions
def display_n_queens_solutions(solutions):
    for solution in solutions:
        for i in range(len(solution)):
            row = ['.'] * len(solution)
            row[solution[i]] = 'Q'
            print(" ".join(row))
        print("\n")
board_size = 6
solutions = solve_n_queens_dynamic(board_size)
print(f"Number of solutions: {len(solutions)}")
display_n_queens_solutions(solutions)
```

```
Number of solutions: 4
. Q . . . .
. . . Q . .
. . . . . Q
Q . . . . .
. . Q . . .
. . . . Q .


. . Q . . .
. . . . . Q
. Q . . . .
. . . . Q .
Q . . . . .
. . . Q . .


. . . Q . .
Q . . . . .
. . . . Q .
. Q . . . .
. . . . . Q
. . Q . . .

...
. . . Q . .
. Q . . . .
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...