www.kiet.edu
Delhi-NCR, Ghaziabad

KIET
GROUP OF INSTITUTIONS
Connecting Life with Learning

Name: Aditya Kumar Singh

University Roll no.: 202401100300016

Batch: CSEAI (A)

REPORT

N-Queens Problem

INRODUCTION:

The N-Queens problem is a classical combinatorial problem that involves placing N chess queens on an N×N chessboard such that no two queens attack each other. This means that no two queens can be in the same row, column, or diagonal. The problem serves as an excellent example of constraint satisfaction and backtracking techniques in computer science.

This report presents a Python implementation of the N-Queens problem using a recursive backtracking approach. The implementation ensures that all possible placements of queens are explored, and valid solutions are printed accordingly.

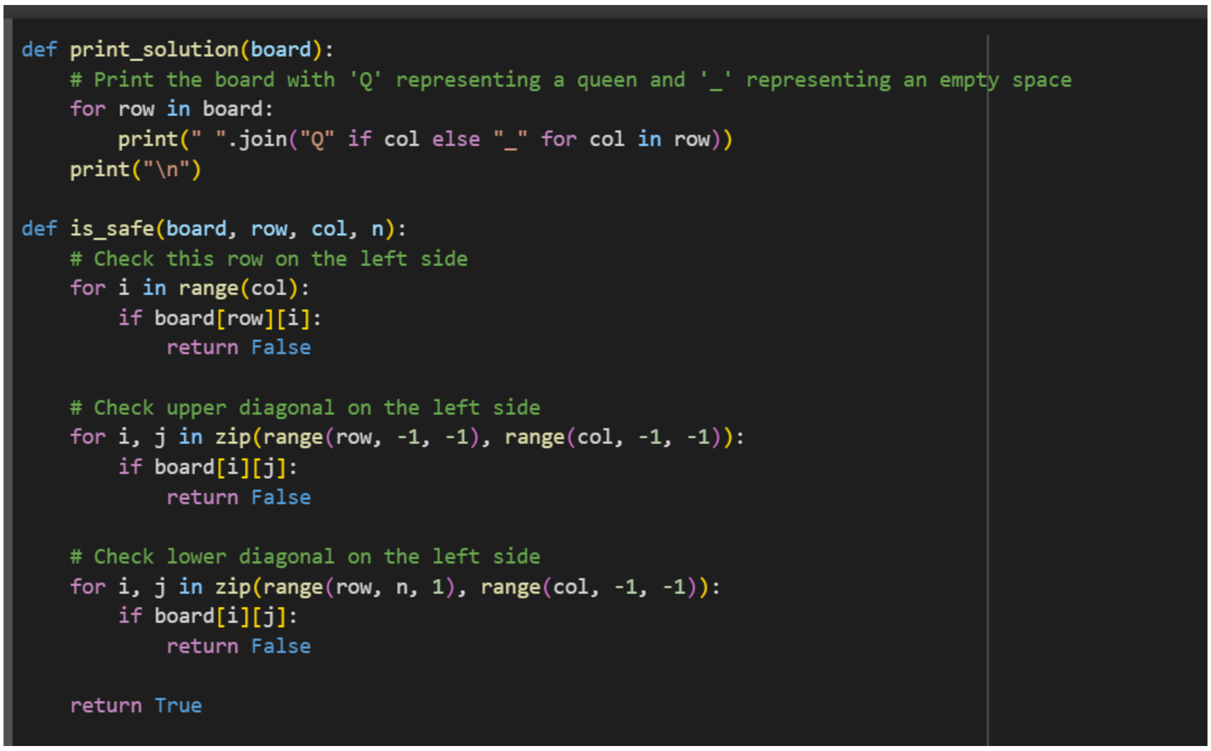
METHODOLOGY:

The solution follows a systematic recursive backtracking approach, which involves the following steps:

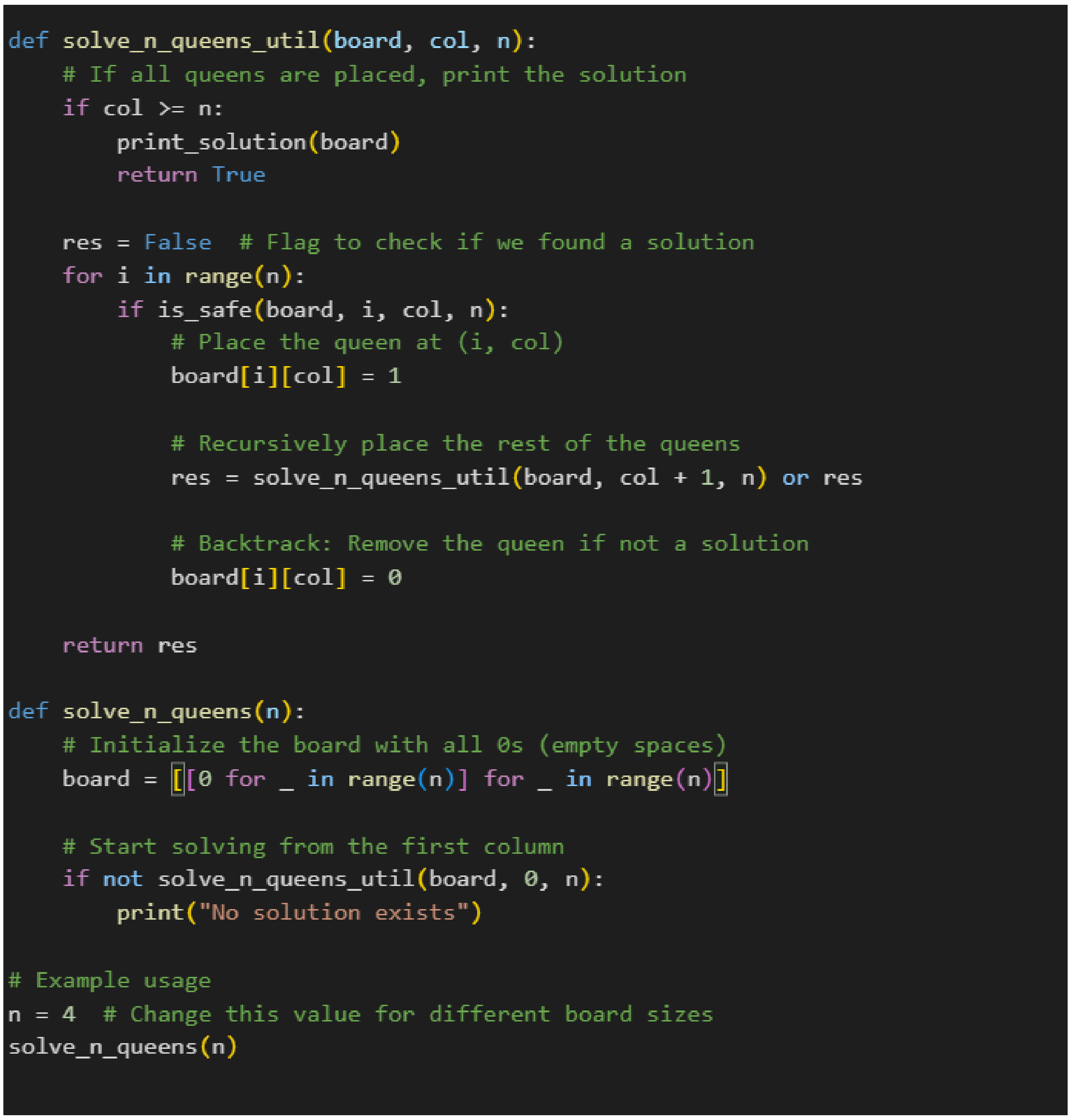
1. **Board Representation:**
   - The chessboard is represented as a 2D list (matrix) of size N×N, where '0' denotes an empty space and '1' denotes a queen.
2. **Checking Safety Constraints:**
   - A function is_safe(board, row, col, n) is used to check whether a queen can be placed at a given position (row, col). It ensures that no other queens exist in the same row, left diagonal, or right diagonal.

3. **Recursive Placement of Queens:**
   - The function solve_n_queens_util(board, col, n) places queens one by one in each column and uses recursion to check for valid placements.
   - If a safe position is found, the queen is placed, and the function is recursively called for the next column.
   - If no valid placement is possible in a column, backtracking occurs by removing the last placed queen and trying alternative positions.
4. **Backtracking Mechanism:**
   - If a solution is found, the board configuration is printed using the print_solution(board) function.
   - If no solution is found for a given configuration, the function backtracks to explore other possibilities.
5. **Driver Function:**
   - The function solve_n_queens(n) initializes an empty board and starts solving from the first column.
   - If no solution exists, a message is displayed.

CODE:

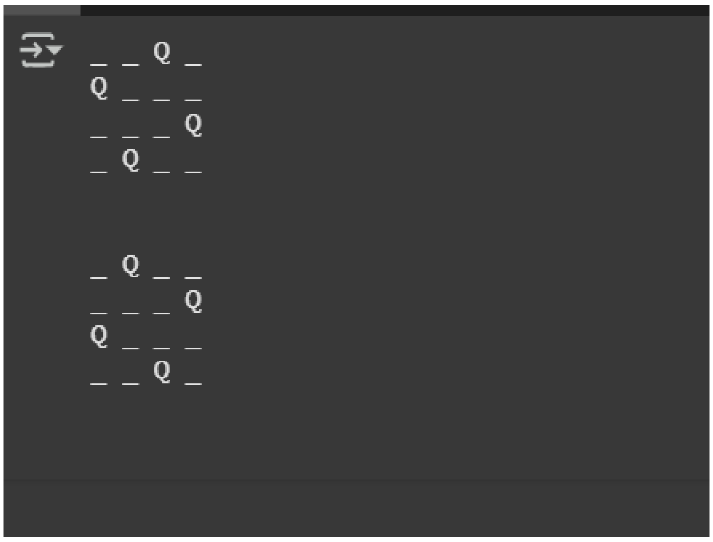


```python
def print_solution(board):
    # Print the board with 'Q' representing a queen and '_' representing an empty space
    for row in board:
        print(" ".join("Q" if col else "_" for col in row))
    print("\n")

def is_safe(board, row, col, n):
    # Check this row on the left side
    for i in range(col):
        if board[row][i]:
            return False

    # Check upper diagonal on the left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j]:
            return False

    # Check lower diagonal on the left side
    for i, j in zip(range(row, n, 1), range(col, -1, -1)):
        if board[i][j]:
            return False

    return True
```



```python
def solve_n_queens_util(board, col, n):
    # If all queens are placed, print the solution
    if col >= n:
        print_solution(board)
        return True

    res = False  # Flag to check if we found a solution
    for i in range(n):
        if is_safe(board, i, col, n):
            # Place the queen at (i, col)
            board[i][col] = 1

            # Recursively place the rest of the queens
            res = solve_n_queens_util(board, col + 1, n) or res

            # Backtrack: Remove the queen if not a solution
            board[i][col] = 0

    return res

def solve_n_queens(n):
    # Initialize the board with all 0s (empty spaces)
    board = [[0 for _ in range(n)] for _ in range(n)]

    # Start solving from the first column
    if not solve_n_queens_util(board, 0, n):
        print("No solution exists")

# Example usage
n = 4  # Change this value for different board sizes
solve_n_queens(n)
```

RESULT:



```
_ _ Q _
Q _ _ _
_ _ _ Q
_ Q _ _


_ Q _ _
_ _ _ Q
Q _ _ _
_ _ Q _
```

REFENCENCES:

1. "Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig.
2. Online resources and tutorials on backtracking algorithms.