PROBLEM STATEMENT : Solve the N-Queens problem using Python and implement the solution on Google Colab.

Name – Brajesh Kumar Keshari

Univ. Roll No. – 20240110030090

Branch – CSEAI

Section - B

Course – Introduction To AI

INTRODUCTION

The N-Queens problem is a classic combinatorial optimization problem in computer science and mathematics. The goal is to place N queens on an N×N chessboard such that no two queens attack each other. This means that:

1. No two queens should be placed in the same row.
2. No two queens should be placed in the same column.
3. No two queens should be placed in the same diagonal (both major and minor diagonals).

This problem is significant in algorithmic studies because it demonstrates backtracking, an efficient technique used to solve constraint satisfaction problems. The problem has applications in various domains, including artificial intelligence and optimization problems.

METHODOLOGY

To solve this problem, we use the **backtracking algorithm**, which follows these steps:

1. **Start with an empty board**: A two-dimensional list (N×N matrix) is initialized with all entries set to False, indicating that no queens are placed initially.
2. **Place queens one by one**: The algorithm attempts to place a queen in each column, ensuring that it does not attack other queens.
3. **Check for safety**: Before placing a queen at a particular position, a function is called to verify whether the placement is valid based on the constraints (row, column, and diagonal checks).
4. **Recursive placement**: If a valid position is found, the algorithm moves to the next column and attempts to place another queen.
5. **Backtracking**: If no valid position is found in a particular column, the algorithm backtracks to the previous column and tries a different position.
6. **Print solutions**: Once all queens are placed successfully, the board configuration is displayed.
7. **Handle unsolvable cases**: If no solution exists for a given N, the algorithm returns an appropriate message.

The implementation is written in Python and executed on Google Colab, with detailed comments for clarity.

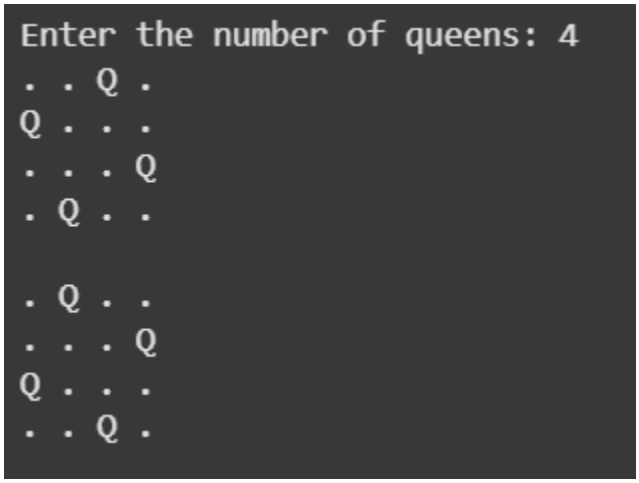
CODE

```
def print_solution(board):
    # Prints the board configuration where 'Q' represents a queen and '.' represents an
empty space
    for row in board:
        print(" ".join("Q" if col else "." for col in row))
    print()

def is_safe(board, row, col, n):
    """
    Checks if it's safe to place a queen at position (row, col)
    A position is safe if:
    - No queen exists in the same row on the left side
    - No queen exists in the upper left diagonal
    - No queen exists in the lower left diagonal
    """

    # 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

def solve_n_queens_util(board, col, n):
    """
    Recursively tries to place queens column by column
    Base case: If all queens are placed successfully, print the solution
    """
    if col >= n:
        print_solution(board)
        return True

    res = False  # To track if at least one solution is found
    for i in range(n):
        # Check if it's safe to place a queen at board[i][col]
        if is_safe(board, i, col, n):
            board[i][col] = True  # Place the queen
            res = solve_n_queens_util(board, col + 1, n) or res  # Recur for next column
            board[i][col] = False  # Backtrack and remove the queen

    return res
```

```python
def solve_n_queens(n):
    """
    Initializes the board and starts solving the N-Queens problem
    """
    board = [[False] * n for _ in range(n)]  # Create an empty NxN board
    if not solve_n_queens_util(board, 0, n):
        print("No solution exists")

# Get input from the user
n = int(input("Enter the number of queens: "))
solve_n_queens(n)
```

OUTPUT

Below is an example output for N=4:




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

Explanation:

- The `Q` represents a queen placed on the board.
- The `.` represents an empty space.
- The placement ensures that no two queens attack each other.

If `N` is such that no valid solution exists (e.g., N=2 or N=3), the program outputs:

```
No solution exists
```

REFERENCES

1. **Google Colab**: Used for executing and testing the code.
2. **Python Documentation**: Helped in understanding recursion and list operations.
3. **Online Algorithm Resources**: Used for understanding the backtracking approach.