EX.NO:01 **DATE:**

Reg.no:220701061

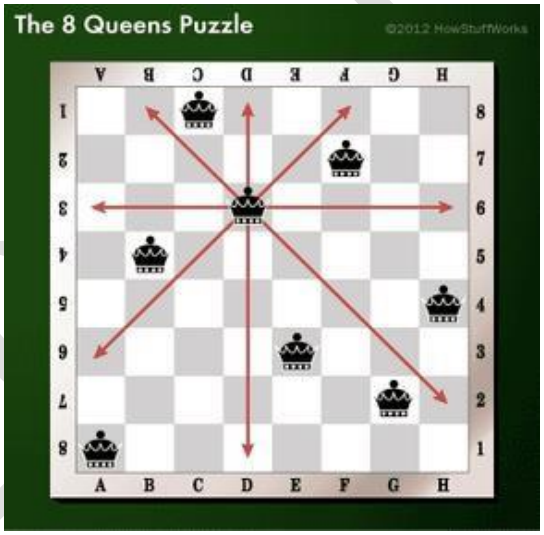
8- QUEENS PROBLEM

AIM:

To implement an 8-Queesns problem using Python.

You are given an 8x8 board; find a way to place 8 queens such that no queen can attack any other queen on the chessboard. A queen can only be attacked if it lies on the same row, same column, or the same diagonal as any other queen. Print all the possible configurations.

To solve this problem, we will make use of the Backtracking algorithm. The backtracking algorithm, in general checks all possible configurations and test whether the required result is obtained or not. For the given problem, we will explore all possible positions the queens can be relatively placed at. The solution will be correct when the number of placed queens = 8.



```python
def is_safe(board, row, col, n):
    # Check this row on the left side
    for i in range(col):
        if board[row][i] == 1:
            return False

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

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

    return True


def solve_n_queens_util(board, col, n):
    # If all queens are placed, return True
    if col >= n:
        return True

    # Try placing the queen in each row in this column
    for i in range(n):
        if is_safe(board, i, col, n):
            # Place the queen
            board[i][col] = 1

            # Recur to place the rest of the queens
            if solve_n_queens_util(board, col + 1, n):
                return True

            # Backtrack and remove the queen
            board[i][col] = 0
```

```python
def print_board(board, n):
    print("\nSolution:")
    for row in board:
        for cell in row:
            if cell == 1:
                print('Q', end=' ')
            else:
                print('.', end=' ')
        print()  # New line after each row


def solve_n_queens(n):
    # Initialize the board
    board = [[0 for _ in range(n)] for _ in range(n)]

    if not solve_n_queens_util(board, 0, n):
        print("No solution exists")
        return False

    # Print the solution in a nice format
    print_board(board, n)
    return True


# Driver code to get user input
n = int(input("Enter the value of N: "))
solve_n_queens(n)
```

```
Enter the value of N: 4

Solution:
. . Q .
Q . . .
. . . Q
. Q . .
True
```

220701061-DHANASREE LP

220701061-DHANASREE LP

```python
def is_safe(board, row, col, n):
    # Check this row on the left side
    for i in range(col):
        if board[row][i] == 1:
            return False

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

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

    return True


def solve_n_queens_util(board, col, n):
    # If all queens are placed, return True
    if col >= n:
        return True

    # Try placing the queen in each row in this column
    for i in range(n):
        if is_safe(board, i, col, n):
            # Place the queen
            board[i][col] = 1

            # Recur to place the rest of the queens
            if solve_n_queens_util(board, col + 1, n):
                return True
```