**Date:17.09.25**

**TASK:8**

Implementation of **N-queen problem using backtracking algorithm.**

Implementation of N-queen problem using backtracking algorithm using python. In the 4 Queens problem the object is to place 4 queens on a chessboard in such a way that no queens can capture a piece. This means that no two queens may be placed on the same row, column, or diagonal.

**Tools: Python**

**PROBLEM STATEMENT: CO3 S3**

In a distant kingdom, the Queen is challenged with a classic chess puzzle called the N-Queens problem. She must place N queens on an N×N chessboard in such a way that no two queens threaten each other. This means that no two queens can be in the same row, the same column, or on the same diagonal.

The Queen wants to find one valid arrangement of the queens that satisfies these conditions, so she can demonstrate a solution to her court. Using her knowledge of chess and logical reasoning, she seeks a methodical approach to place the queens one by one, backtracking whenever a conflict arises, until a safe configuration is found.

**IMPLEMENTATION OF N-QUEEN PROBLEM**

**USING BACKTRACKING ALGORITHM**

**AIM**

To Implement N-Queen's problem by using backtracking algorithm using python.

**ALGORITHM**

1. Initialize board: Create an N×N chessboard and set all cells to empty.
2. Start at first column: Begin placing queens in the first column.
3. Place queen: Attempt to place a queen in the current column, starting from the first row.
4. Check safety: Before placing, check if the cell is safe (no other queen in the same row, column, or diagonal).
5. Place if safe: If the position is safe, place the queen in that cell.
6. Move to next column: Recursively attempt to place a queen in the next column.
7. Backtrack if needed: If no safe position is found in a column, remove the previously placed queen (backtrack) and try the next row in the previous column.
8. Repeat steps 3–7: Continue until all queens are placed successfully or all possibilities are exhausted.
9. Solution found: Once N queens are placed safely, stop recursion and record this arrangement as a solution.
10. Display solution: Print the board showing the positions of the N queens.

**PROGRAM**

**N-Queens Problem**

def is\_safe(board, row, col, N):

# Check this row on the left

for i in range(col):

if board[row][i] == 1:

return False

# Check upper diagonal on left side

for i, j in zip(range(row, -1, -1), range(col, -1, -1)):

if board[i][j] == 1:

return False

# Check lower diagonal on left side

for i, j in zip(range(row, N, 1), range(col, -1, -1)):

if board[i][j] == 1:

return False

return True

def solve\_nqueens\_one\_solution(board, col, N):

if col >= N:

return True # Found one solution

for row in range(N):

if is\_safe(board, row, col, N):

board[row][col] = 1

if solve\_nqueens\_one\_solution(board, col + 1, N):

return True

board[row][col] = 0 # backtrack

return False

def print\_solution(board, N):

for row in board:

print(" ".join('Q' if x else '.' for x in row))

# Example: Solve 4-Queens

N = 4

board = [[0]\*N for \_ in range(N)]

if solve\_nqueens\_one\_solution(board, 0, N):

print(f"One solution for {N}-Queens:")

print\_solution(board, N)

else:

print(f"No solution exists for {N}-Queens")

**OUTPUT**

**A screenshot of a computer

AI-generated content may be incorrect.**

**RESULT**

Thus, the Implementation of N-queen problem using backtracking algorithm using Python was successfully executed and output was verified.