**Solving the N-Queens Problem Using Python**

**By – Jahnavi Nagar**

**57**

**AI - B**

**Introduction**

The N-Queens problem is a classic combinatorial problem in computer science and mathematics. The objective is to place N chess queens on an N×N chessboard such that no two queens threaten each other. This means no two queens can share the same row, column, or diagonal. The N-Queens problem is widely used in algorithm studies to explore backtracking methods and combinatorial optimization.

In this report, we present a Python solution that uses a backtracking algorithm to efficiently solve the N-Queens problem for any given value of N. The solution will display all possible configurations where the queens are safely positioned.

**Methodology**

The solution is implemented using the following steps:

1. **Initialization:**
   * An empty N×N chessboard is created, represented by a 2D list initialized with False values.
2. **Safety Check:**
   * A function is\_safe verifies if placing a queen at a specific position is valid. It checks:
     + The column above the current cell.
     + The upper-left diagonal.
     + The upper-right diagonal.
3. **Backtracking Algorithm:**
   * The solve\_n\_queens function recursively attempts to place queens row by row.
   * If a valid position is found, the queen is placed, and the algorithm moves to the next row.
   * If no valid position is found in a row, backtracking occurs, and the previous queen's position is adjusted.
4. **Output:**
   * The valid configurations are printed using the print\_solution function, which formats the chessboard with 'Q' representing queens and '.' representing empty cells.

**Code Implementation**

def print\_solution(board):

for row in board:

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

print("\n")

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

for i in range(row):

if board[i][col]:

return False

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

if board[i][j]:

return False

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

if board[i][j]:

return False

return True

def solve\_n\_queens(board, row, n):

if row == n:

print\_solution(board)

return True

res = False

for col in range(n):

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

board[row][col] = True

res = solve\_n\_queens(board, row + 1, n) or res

board[row][col] = False # Backtrack

return res

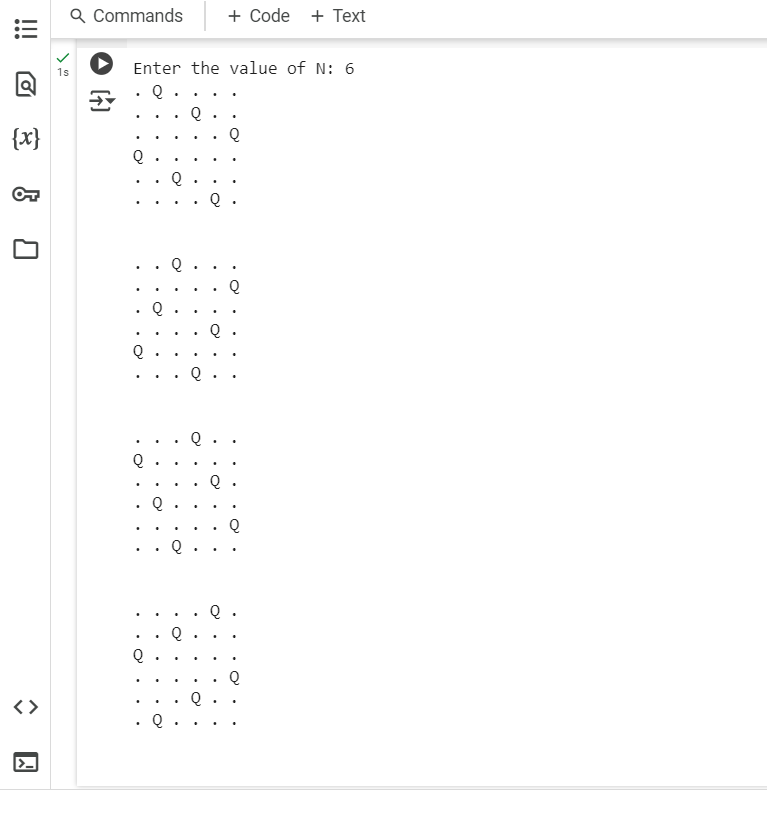
def n\_queens(n):

board = [[False for \_ in range(n)] for \_ in range(n)]

if not solve\_n\_queens(board, 0, n):

print("No solution exists")

n = int(input("Enter the value of N: "))



Screenshot of the output

**Conclusion**

This Python implementation efficiently solves the N-Queens problem using a backtracking algorithm. The solution effectively identifies all valid queen placements on the chessboard. This method is adaptable for various values of N and demonstrates the power of backtracking in solving complex combinatorial problems. Future improvements can include visual representations and performance optimizations for larger values of N.