**SOFTWARE LABORATORY 1**

**GROUP B – EXPERIMENT 4**

**TITLE:**

Solving Constraint Satisfaction Problems Using Branch and Bound and Backtracking (N-Queens/Graph Coloring)

**CODE:**

def is\_safe(queen\_positions, current\_row, current\_col):

"""

Check if placing a queen at (current\_row, current\_col) is safe.

queen\_positions[i] stores the column of the queen in row i.

"""

# Check all previously placed queens

for row in range(current\_row):

col = queen\_positions[row]

# Check same column

if col == current\_col:

return False

# Check diagonals: difference in rows == difference in cols means diagonal attack

if abs(current\_row - row) == abs(current\_col - col):

return False

# No conflict found, position is safe

return True

def solve\_n\_queens(n):

solutions = [] # To store all solutions found

queen\_positions = [-1] \* n # queen\_positions[row] = col index of queen in that row

def backtrack(row=0):

# If all queens placed (row == n), save the solution

if row == n:

solutions.append(queen\_positions.copy())

return

# Try placing a queen in every column of the current row

for col in range(n):

if is\_safe(queen\_positions, row, col):

queen\_positions[row] = col # Place queen

backtrack(row + 1) # Move to next row

queen\_positions[row] = -1 # Remove queen (backtrack)

backtrack() # Start from row 0

return solutions

def print\_solutions(solutions):

for sol in solutions:

n = len(sol)

for row in range(n):

line = [' . '] \* n

line[sol[row]] = 'Q' # Put queen in correct column

print("".join(line))

print() # Blank line between solutions

# Example: Solve and print 4-Queens problem

n = 4 solutions = solve\_n\_queens(n)

print(f"Total solutions found: {len(solutions)}\n")

print\_solutions(solutions)

**Output:**

Total solutions found: 2

. Q . .

. . . Q

Q . . .

. . Q .

. . Q .

Q . . .

. . . Q

. Q . .

Process finished with exit code 0

