N-Queens Problem Report

Problem Statement: N-Queens problem

Name: Amritanshu Chaudhary

Roll No.: 202401100300036

Introduction

The N-Queens problem is a classic combinatorial problem in which we must place N queens on an N×N chessboard so that no two queens threaten each other. This means that no two queens should be in the same row, column, or diagonal. The problem is commonly solved using backtracking or other AI techniques such as constraint satisfaction and genetic algorithms.

Methodology

To solve the problem, we use a backtracking algorithm:

1. Start by placing a queen in the first row.
2. Move to the next row and place a queen in a valid column.
3. If a safe position is found, move to the next row.
4. If no valid position exists, backtrack to the previous row and try the next available column.
5. Repeat until all N queens are placed successfully.
6. If a solution is found, store it and continue searching for other solutions if required.

The algorithm ensures that each queen is placed in a non-attacking position, adhering to the constraints of the problem.

Code

Below is the Python implementation of the N-Queens problem using backtracking:

```
# Function to check if placing a queen is safe
def is_safe(board, row, col, N):
   for i in range(row):
      if board[i] == col or \
        board[i] - i == col - row or \
        board[i] + i == col + row:
         return False
   return True
# Backtracking function to find all solutions
def solve_n_queens_util(board, row, N, solutions):
   if row == N:
      solutions.append(board[:])
      return
 for col in range(N):
      if is_safe(board, row, col, N):
         board[row] = col
         solve_n_queens_util(board, row + 1, N, solutions)
         board[row] = -1
# Function to solve N-Queens problem and return solutions
def solve_n_queens(N):
   solutions = []
   board = [-1] * N
   solve_n_queens_util(board, 0, N, solutions)
   return solutions
```

```
# Function to print each solution
def print_solution(solution):
  for row in solution:
    board_row = ['Q' if col == row else '.' for col in range(len(solution))]
    print(" ".join(board_row))
# Set N to 8 for the 8-Queens problem
N = 8
solutions = solve_n_queens(N)
print(f"Total Solutions: {len(solutions)}")
for i, solution in enumerate(solutions, 1):
  print(f"\nSolution {i}:")
  print_solution(solution)
```

Output/Result

Below is a sample output for N = 8:

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

References/Credits

1. Artificial Intelligence: A Modern Approach - Stuart Russell & Peter Norvig.
2. Online resources such as GeeksforGeeks and Stack Overflow for algorithm optimization.
3. Python documentation for list comprehensions and recursive functions.