

The Superior University

|  |  |  |
| --- | --- | --- |
| Name: Abdul Moeez | Roll No: 017 | Course: PAI Lab |
| Semester: 4th | Section: BSAI 4A | Department: Artificial Intelligence |
| Submitted To: Sir Rasikh | Total Marks: | Date: |

**Lab 4**

**Task: N Queen Problem: Documentation**

#### **Overview**

The N-Queens Problem is a classic puzzle where the goal is to place N queens on an N x N chessboard such that no two queens threaten each other. This means no two queens can share the same row, column, or diagonal. The problem is a well-known example of backtracking algorithms and combinatorial optimization.

The provided Python code solves the N-Queens Problem using a **backtracking approach**. It systematically explores all possible configurations of queen placements and returns all valid solutions.

#### **How the Code Works**

1. **Function: n\_queen(n)**
   1. This function finds all valid solutions to the N-Queens Problem.
   2. **Input Parameter:**
      1. n: Size of the chessboard (N x N).
   3. **Output:**
      1. Returns a list of solutions, where each solution is represented as a list of column indices for each row.
2. **Helper Function: check\_position(board, row, col)**
   1. This function checks if placing a queen at a specific position (row, col) is valid.
   2. **Checks Performed:**
      1. No other queen is in the same column.
      2. No other queen is on the same diagonal.
   3. **Output:**
      1. Returns True if the position is valid, otherwise False.
3. **Backtracking Function: backtrack(row)**
   1. This function recursively places queens row by row and backtracks when a conflict is detected.
   2. **Base Case:**
      1. If row == n, a valid solution is found, and it is added to the solutions list.
   3. **Recursive Case:**
      1. For each column in the current row, the function checks if placing a queen is valid using check\_position.
      2. If valid, the queen is placed, and the function proceeds to the next row.
      3. After exploring all possibilities, the function backtracks by resetting the current row's queen placement.
4. **Function: solutions(solutions, n)**
   1. This function prints all solutions in a readable format.
   2. **Input:**
      1. solutions: A list of valid solutions.
      2. n: Size of the chessboard.
   3. **Output:**
      1. Prints each solution as an N x N grid, where Q represents a queen and . represents an empty cell.
5. **User Input and Validation:**
   1. The user is prompted to input the size of the chessboard (N).
   2. Input validation ensures that N is a positive integer.

#### **Why This Approach?**

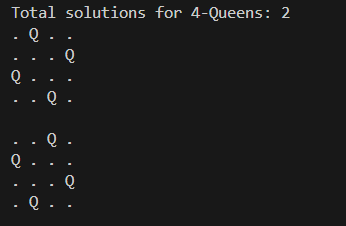
* **Backtracking:** The backtracking algorithm systematically explores all possible configurations and efficiently prunes invalid branches, making it suitable for solving combinatorial problems like the N-Queens Problem.
* **Efficiency:** By checking for conflicts at each step, the algorithm avoids unnecessary computations and ensures that only valid solutions are considered.
* **Scalability:** While the problem's complexity grows exponentially with N, the backtracking approach is effective for small to moderate values of N.

#### **Example Usage**

**Input:**



**Output:**



#### **Key Points**

* The code handles all edge cases, such as invalid inputs.
* The solution is presented in a clear and readable format, making it easy to visualize the chessboard.
* The algorithm is flexible and can be adapted to similar combinatorial problems.

#### **Conclusion**

This implementation provides an efficient and clear solution to the N-Queens Problem using a backtracking approach. It demonstrates the power of systematic exploration and pruning in solving complex combinatorial puzzles. The code can serve as a foundation for solving similar problems or optimizing the N-Queens solution further.