Task 04

Implementation of N-Queen Problem (Dynamic)

Problem Overview:

The N-Queen problem is a classic combinatorial problem that requires placing N queens on an N×N chessboard such that no two queens threaten each other. This means:

- No two queens share the same row.
- No two queens share the same column.
- No two queens share the same diagonal.

Code Explanation:

Initialization:

- An N×N chessboard is initialized with all cells set to 0.
- A recursive function is used to place queens one by one.
- The function checks for a valid placement using a helper function `isValid`.

Rules:

At each step, the algorithm:

1. Checks if placing a queen in a given row and column is valid.
2. Places a queen in a column of the current row if valid.
3. Recursively attempts to place queens in subsequent rows.
4. If placing a queen leads to a solution, it returns true.
5. If not, it backtracks by removing the queen and trying the next possible placement.
6. If all rows are filled successfully, a solution is found.
7. If no solution is found after all possibilities are exhausted, it prints "Solution not found!"

Recursive Execution:

- The `recur_nqueen` function starts with an empty board.
- It places queens one by one, ensuring each placement is valid.
- If it reaches the last row with a valid configuration, it prints the board.
- If a placement leads to failure, it backtracks and tries the next possibility.

Goal Check:

If all queens are successfully placed:

- The chessboard is printed with '1' representing a queen and '0' representing an empty space.
- If no valid placement is found, a message indicating failure is displayed.

Example Execution:

The program takes user input for the value of N and attempts to solve the N-Queen problem using recursion.

```
print("N Queen Problem:")
num = int(input("Enter the number of queens: "))
NQueen(num)
```

Output:

If a valid configuration exists, the program outputs a chessboard representation with queens placed correctly.

If no solution exists for the given N, it prints:

```
Solution not found !!
```

Conclusion:

This implementation efficiently solves the N-Queen problem using backtracking and recursion. The `isValid` function ensures that queens are placed in safe positions, and the `recur_nqueen` function systematically explores all possibilities, backtracking when necessary. The problem demonstrates the power of recursion and constraint satisfaction in combinatorial problems.

Screenshot: