N-Queens Problem

Problem Statement

The N-Queens problem is a classic combinatorial problem where you must place N chess queens on an N×N chessboard so that no two queens threaten each other. This means:

1. No two queens can be in the same row.
2. No two queens can be in the same column.
3. No two queens can be on the same diagonal.

The objective is to find all possible solutions for the given N.

Data Structures Used

1. Sets:

- \`cols\`: Tracks the columns occupied by queens.

- \`posDig\`: Tracks positive diagonals (r + c).

- \`negDig\`: Tracks negative diagonals (r - c).

2. Recursive Function:

- \`dfs(r)\`: Implements backtracking to explore valid placements row by row.

3. Board Representation:

- A list of strings where each string represents a row of the chessboard.

N-Queens Problem

Complexity Analysis

Time Complexity:

- The time complexity is O(N!), where N is the size of the board. This is because:
  - In the first row, we have N choices.
  - In the second row, we have N-1 choices, and so on.

Space Complexity:

- The space complexity is O(N), as we use sets (`cols`, `posDig`, `negDig`) and the recursive call stack can go up to N levels deep.

Applications

1. Problem Solving:

- Used in algorithms and AI to solve constraint satisfaction problems.

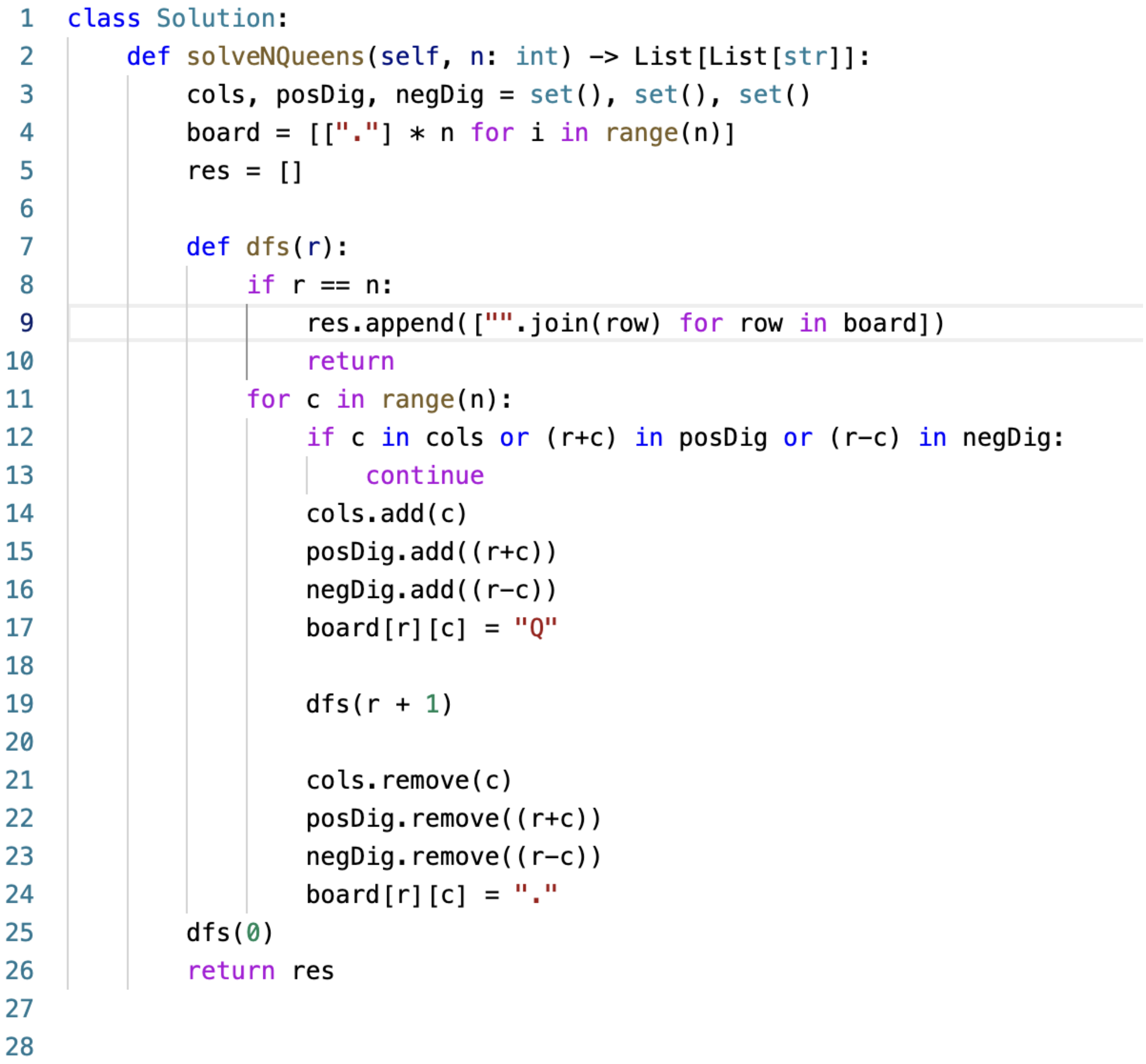
2. Education:

- Serves as a fundamental example of backtracking.

3. Chess Programming:

- Can be used to analyze chessboard-based problems.

N-Queens Problem



```python
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        cols, posDig, negDig = set(), set(), set()
        board = [["."] * n for i in range(n)]
        res = []

        def dfs(r):
            if r == n:
                res.append(["".join(row) for row in board])
                return
            for c in range(n):
                if c in cols or (r+c) in posDig or (r-c) in negDig:
                    continue
                cols.add(c)
                posDig.add((r+c))
                negDig.add((r-c))
                board[r][c] = "Q"

                dfs(r + 1)

                cols.remove(c)
                posDig.remove((r+c))
                negDig.remove((r-c))
                board[r][c] = "."
        dfs(0)
        return res


```