回溯算法是一种通过尝试所有可能的候选解来解决问题的算法。它通常用于解决组合问题、搜索问题或优化问题，其中需要找到所有可能的解或是找到满足特定条件的解。***回溯算法通过逐步构建解决方案，如果在构建过程中发现当前方案不可行或不符合条件，则回溯到之前的状态，尝试其他的选择。***这种回溯过程一直持续到找到所需的解或是确定不存在解为止。回溯算法常常用于解决如八皇后问题、数独、子集和组合等问题。

```python
def is_safe(board, row, col, n):
    # 检查当前位置的列是否安全
    for i in range(row):
        if board[i][col] == 1:
            return False

    # 检查左上方的对角线是否安全
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # 检查右上方的对角线是否安全
    for i, j in zip(range(row, -1, -1), range(col, n)):
        if board[i][j] == 1:
            return False

    return True

def solve_n_queens_util(board, row, n, result):
    if row == n:
        # 当所有行都放置了皇后，将当前解添加到结果中
        result.append(["".join(row) for row in board])
        return

    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1
            solve_n_queens_util(board, row + 1, n, result)
            board[row][col] = 0  # 回溯到上一步，尝试其他的列

def solve_n_queens(n):
    board = [[0] * n for _ in range(n)]
    result = []
    solve_n_queens_util(board, 0, n, result)
    return result

# 测试
n = 8
solutions = solve_n_queens(n)
for i, solution in enumerate(solutions):
    print(f"Solution {i+1}:")
    for row in solution:
        print(row)
    print()
```

这段代码通过递归和回溯的方式，尝试在棋盘上放置八个皇后，使得它们互相不攻击。