```python
def is_safe(board, row, col, n):
    for i in range(row):
        if board[i][col] == 1:
            return False
    for i, j in zip(range(row - 1, -1, -1), range(col - 1, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row - 1, -1, -1), range(col + 1, n)):
        if board[i][j] == 1:
            return False
    return True

def solve_n_queens_bt(board, row, n, solutions):
    if row == n:
        solutions.append([r[:] for r in board])
        return
    for col in range(n):
        if is_safe(board, row, col, n):
            board[row][col] = 1
            solve_n_queens_bt(board, row + 1, n, solutions)
            board[row][col] = 0

def n_queens_backtracking(n):
    board = [[0] * n for _ in range(n)]
    solutions = []
    solve_n_queens_bt(board, 0, n, solutions)
    return solutions

def print_board(board):
    for row in board:
        print(" ".join("Q" if x == 1 else "." for x in row))

def solve_n_queens_bb(n):
    cols = [False] * n
    diag1 = {i: False for i in range(-n + 1, n)}
    diag2 = {i: False for i in range(2 * n - 1)}
    board = [-1] * n
    solutions = []

    def solve(row):
        if row == n:
            solutions.append(board[:])
            return
        for col in range(n):
            if not cols[col] and not diag1[row - col] and not diag2[row + col]:
                cols[col] = diag1[row - col] = diag2[row + col] = True
                board[row] = col
                solve(row + 1)
                cols[col] = False
                diag1[row - col] = diag2[row + col] = False

    solve(0)
    return solutions

def print_bb_solution(board):
    n = len(board)
    for row in range(n):
        line = ""
        for col in range(n):
            if board[row] == col:
                line += "Q "
            else:
                line += ". "
        print(line.strip())

def main_menu():
    while True:
        print("\n========== N-QUEENS MENU ==========")
        print("1. Solve using Backtracking")
        print("2. Solve using Branch and Bound")
        print("3. Exit")
        choice = input("Enter your choice: ")

        if choice in ['1', '2']:
            try:
                n = int(input("Enter value of N (>=4): "))
                if n < 4:
                    print("N must be at least 4.")
                    continue
            except:
                print("Invalid input.")
                continue

            if choice == '1':
                print(f"\nSolving {n}-Queens using Backtracking...\n")
                solutions = n_queens_backtracking(n)
                print(f"Backtracking: Total solutions for N={n}: {len(solutions)}\n")
                for idx, sol in enumerate(solutions, 1):
```

```
                print(f'Solution {idx}:')
                print_board(sol)
                print("-" * (n * 2))
        else:
            print(f"\nSolving {n}-Queens using Branch and Bound...\n")
            bb_solutions = solve_n_queens_bb(n)
            print(f"Branch and Bound: Total solutions for N={n}: {len(bb_solutions)}\n")
            for idx, sol in enumerate(bb_solutions, 1):
                print(f'Solution {idx}:')
                print_bb_solution(sol)
                print("-" * (n * 2 + 1))

    elif choice == '3':
        print("Exiting program successfully...")
        break
    else:
        print("Invalid choice! Please try again.")

main_menu()
```