The Superior University, Lahore

Assignment-I (Fall 2023)

Course Title:	Programming for AI			Course Code:	CAI601410	Credit Hours:	4
Instructor:	Prof. Rasikh Ali			Programme Name:	BSDS		
Semester:	4th	Batch:	F23	Section:	BSDSM-4A	Date:	1st February, 2025
Time Allowed:				Maximum Marks:			
Student's Name:	Asaad Iqbal			Reg. No.	SU92-BSDSM-F23-020		

Lab-Task 4

1: N-Queen Problem

Task 1

N-queen Problem Documentation

Code:

```
def safe(board, row, col):
    for r in range(row):
        if board[r] == col or abs(r - row) == abs(board[r] - col):
            return False
    return True

def solve_n_queens(n):
    board = [-1] * n

    def backtrack(row):
        if row == n:
            return True

        for col in range(n):
            if safe(board, row, col):
                board[row] = col
                if backtrack(row + 1):
                    return True
                board[row] = -1
        return False
    return board if backtrack(0) else None

def print_board(solution):
    n = len(solution)
    for row in solution:
        line = ['.'] * n
        line[row] = 'Q'
        print(' '.join(line))
```

```python
def main():
    while True:
        try:
            n = int(input("\nEnter chessboard size (N > 0): "))
            if n <= 0:
                print("Please enter a positive integer.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter an integer.")
    solution = solve_n_queens(n)
    if solution:
        print(f"\nSolution for {n}x{n} chessboard:")
        print_board(solution)
    else:
        print(f"\nNo solution exists for {n}x{n} chessboard.")
```
[3] ✓ 0.0s

```python
main()
```
[4] ✓ 1.5s

```
Solution for 5x5 chessboard:
Q . . . .
. . Q . .
. . . . Q
. Q . . .
. . . Q .
```

This presents a solution to the N-Queens problem in a backtracking algorithm. The N-Queens problem involves placing N queens on an N × N chessboard in such a way that no two queens attack each other. This implies:

- No two queens are in the same row.
- No two queens are in the same column.
- No two queens are in the same diagonal.

The program determines a valid position (if any) and displays the solution as a graphical representation of the chessboard.

Code Explanation

1. Safety Check (safe function)

This function is used to check whether it is safe to put a queen at a specified position (row, col).

- It checks whether any queen in the earlier rows is in the same column.
- It checks whether any queen is on the same diagonal by checking the absolute row difference is equal to the absolute column difference.

2. Backtracking Algorithm (solve_n_queens function)

- A list (board) is used to store column positions of queens in each row.
- The function recursively attempts to put queens row by row.
- If a queen can be safely put in a row, the function proceeds to the next row.

- If it encounters a full valid solution, it returns True.
- If a placement fails in a row, it backtracks by retracting the last queen put and attempting an alternative column.
- If no solution exists, it returns None.

3. Printing the Chessboard (print_board function)

- Translates the board list into a visual chessboard representation.
- Uses "Q" to denote queens and "." for blank squares.

4. User Input and Execution (main function)

- Prompts the user for the chessboard size (N).
- Validates the input as a positive integer.
- Calls solve_n_queens(n) and prints the solution or a message if there is no solution.

Backtracking Process in Steps

- Begin with an empty board.
- Put a queen in column 0 of the first possible row.
- Move down to the next row and try to put a queen safely.
- If a safe place is not found, move back to the last row and try a different column.

Continue until either:

- All queens have been placed (solution).
- No solution can be placed (no solution).

This algorithm guarantees all constraints are met while finding a solution efficiently with backtracking.