Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering
Semester: (Spring, Year:2024), B.Sc. in CSE (Day)

Lab Report NO #03
Course Title: Artificial Intelligence Lab
Course Code: CSE 316 Section: 213-D5

Lab Experiment Name: Solve N-Queen Problem Using Backtracking Algorithm

Student Details

Name		ID
1.	Mostofa Rafid	213902097

Lab Date : 15/05/2024
Submission Date : 28/05/2024
Course Teacher's Name : Sagufta Sabah Nakshi

Lab Report Status

Marks: …… **Signature:.....**
Comments:..... **Date:.....**

1. TITLE OF THE LAB REPORT EXPERIMENT

Solve N-Queen Problem Using Backtracking Algorithm.

2. OBJECTIVES

- To understand how backtrack algorithm performs to solve constraint satisfaction problem
- To understand how N-queen problem is solvable by using backtrack method.

3. PROCEDURE

1. Prompt user for N.
2. Define solve_n_queens(N) to create an N×N board and initiate backtracking.
3. Define n_queen(board, row, n, solutions):
    a. If n == 0, add the board to solutions.
    b. For each column, use is_attack(board, row, col) to check safety.
    c. If safe, place a queen, recurse for next row, and backtrack if needed.
4. Define is_attack(board, i, j):
    a. Check column above and diagonals for queens.
5. Define print_solutions(solutions) to print solutions.
6. Call solve_n_queens(N) to find solutions.
7. Print solutions using print_solutions(solutions).

4. IMPLEMENTATION

Problem statement: The N-queens puzzle is the problem of placing N queens on a (N×N) chessboard such that no two queens can attack each other. Find all distinct solution to the N-queen problem.

Code:

```
def print_solutions(solutions):
    for solution in solutions:
        for row in solution:
            print(row)
        print()
def is_attack(board, i, j):
    for k in range(0, i):
        if board[k][j] == 1:
            return True
    k = i - 1
    l = j + 1
    while k >= 0 and l <= len(board) - 1:
        if board[k][l] == 1:
            return True
        k = k - 1
        l = l + 1
    k = i - 1
```

```
    l = j - 1
    while k >= 0 and l >= 0:
        if board[k][l] == 1:
            return True
        k = k - 1
        l = l - 1
    return False
def n_queen(board, row, n, solutions):
    if n == 0:
        solutions.append([row[:] for row in board])
        return
    for j in range(len(board)):
        if not is_attack(board, row, j):
            board[row][j] = 1
            n_queen(board, row + 1, n - 1, solutions)
            board[row][j] = 0
def solve_n_queens(N):
    board = [[0] * N for _ in range(N)]
    solutions = []
    n_queen(board, 0, N, solutions)
    return solutions
N = int(input("Enter the number of queens: "))
solutions = solve_n_queens(N)
print_solutions(solutions)
```

5. OUTPUT

```
PS D:\Python\Artificial Intelligence> python -u "d:\Python\Artificial Intelligence\NQueenM.py"
Enter the number of queens: 4
[0, 1, 0, 0]
[0, 0, 0, 1]
[1, 0, 0, 0]
[0, 0, 1, 0]

[0, 0, 1, 0]
[1, 0, 0, 0]
[0, 0, 0, 1]
[0, 1, 0, 0]

PS D:\Python\Artificial Intelligence>
```

6. ANALYSIS AND DISCUSSION

The N-Queen problem is effectively solved using backtracking, which explores all potential queen placements on an N×N chessboard while ensuring no two queens threaten each other. The algorithm recursively places queens row by row, using the is_attack function to check for conflicts and backtracking as needed. This ensures all solutions are found and stored, showcasing the efficiency and elegance of backtracking in constraint satisfaction problems. The implementation is versatile for any N, highlighting its scalability and robustness in solving combinatorial puzzles.