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

Faculty of Science and Engineering

Semester: (Spring, Year:2025), B.Sc. in CSE (Day)

Lab Report 03

Course Title: Artificial Intelligence Lab

Course Code: CSE 316 **Section: 221 D9**

Lab Experiment Name: N-Queen Problem using Backtracking

<u>Student Details</u>

Name	ID
Shahadat Hosen Nishan	221002099

Lab Date	:	**13/04/2025**
Submission Date	:	**25/04/2025**
Course Teacher's Name	:	**Md. Riad Hassan**

[For Teachers use only: Don't Write Anything inside this box]

Lab Report

Status Marks: …………………………………… **Signature:.....................**

Comments:... **Date:..............................**

Title:

Solving the N-Queen Problem Using Backtracking in Python

Objective:

To implement the N-Queen problem using backtracking in Python, where the goal is to place N queens on an N×N chessboard such that no two queens attack each other.

Problem Statement:

The N-Queen problem involves placing N queens on an N×N chessboard so that:

No two queens are in the same row, column, or diagonal. The task is to find a valid arrangement or all possible solutions.

Algorithm Used: Backtracking

Steps:

1. Start from column 0
2. Try placing a queen in every row of the current column
3. If safe, place the queen and recursively try the next column
4. If placing a queen leads to no solution, backtrack (remove the queen)
5. Repeat until all queens are placed or all possibilities are exhausted

Safety Checks (isSafe):

- Left side of the row
- Upper-left diagonal
- Lower-left diagonal

Implementation:

<u>Code:</u>

```
class N_Queen:
    def __init__(self, a):
        self.N = a

    def printSolution(self, board):
        for i in range(self.N):
            for j in range(self.N):
                print(f"{board[i][j]} ", end='')
            print()

    def isSafe(self, grid, row, col):
        for i in range(col):
```

```
            if grid[row][i] == 1:
                return False

        i, j = row, col
        while i >= 0 and j >= 0:
            if grid[i][j] == 1:
                return False
            i -= 1
            j -= 1

        i, j = row, col
        while j >= 0 and i < self.N:
            if grid[i][j] == 1:
                return False
            i += 1
            j -= 1

        return True

    def solveNQUtil(self, grid, col):
        if col >= self.N:
            return True

        for i in range(self.N):
            if self.isSafe(grid, i, col):
                grid[i][col] = 1
                if self.solveNQUtil(grid, col + 1):
                    return True
                grid[i][col] = 0  # BACKTRACK

        return False

    def solveNQ(self):
        grid = [[0 for _ in range(self.N)] for _ in range(self.N)]
        if not self.solveNQUtil(grid, 0):
            print(f"Solution does not exist for {self.N} queens.")
            return False
        print(f"Solution found for {self.N} queens:")
        self.printSolution(grid)
        return True


if __name__ == "__main__":
    n = int(input("Number of queens to place: "))
    queen_solver = N_Queen(n)
    queen_solver.solveNQ()

```

Output:

```
nishan@nishan:/media/nishan/Work/Semester Files/8th Semester Spring 25/Artificial Intel
ligence Lab$ /bin/python3 "/media/nishan/Work/Semester Files/8th Semester Spring 25/Art
ificial Intelligence Lab/N_Queen.py"
Number of queens to place: 5
Solution found for 5 queens:
1 0 0 0 0
0 0 0 1 0
0 1 0 0 0
0 0 0 0 1
0 0 1 0 0
```

fig: N_Queen Solution using Backtracking

Conclusion:

The N-Queen problem was successfully solved using backtracking, which systematically explores the search space and uses recursion with backtracking to find a valid configuration. The solution is efficient for small values of N and demonstrates a classic example of constraint satisfaction problems.