Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL401**	**Course Name:**	**Analysis of Algorithm Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**9**
Title of the Experiment:	**To implement N Queen problem using Backtracking method**
Date of Performance:	**13/03/2025**
Date of Submission:	**20/03/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mrs. Sneha Yadav

Signature :

Date:

Experiment No. 9

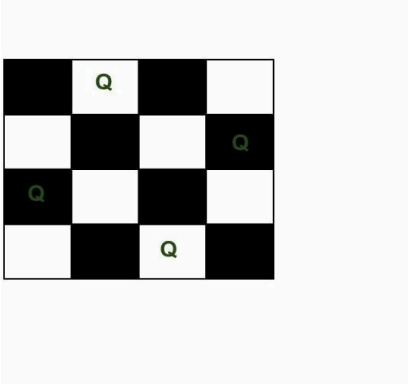
Title: N-Queen

Aim: To study and implement N-Queen

Objective: To introduce Backtracking and Branch-Bound methods

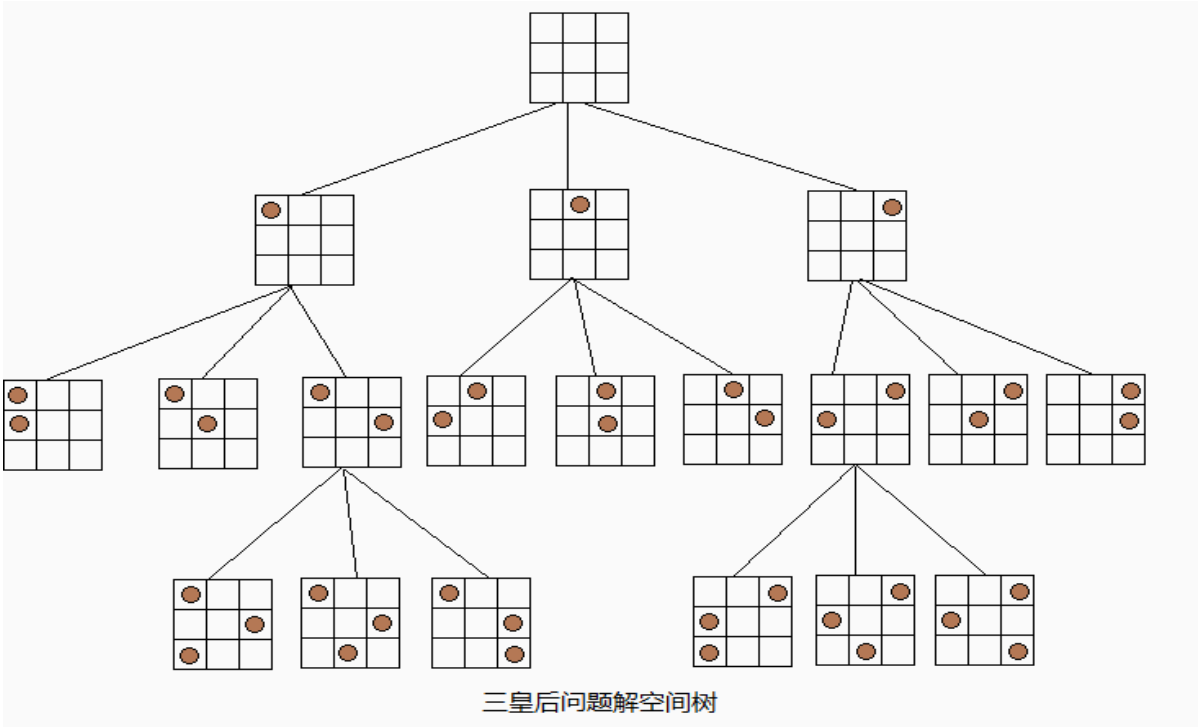
Theory:

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other. For example, following is a solution for 4 Queen problem.



- Start in the leftmost column
- If all queens are placed return true
- Try all rows in the current column. Do the following for every row.
    - If the queen can be placed safely in this row
        - Then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
        - If placing the queen in [row, column] leads to a solution then return true.
        - If placing queen doesn't lead to a solution then unmark this [row, column] then backtrack and try other rows.
    - If all rows have been tried and valid solution is not found return false to trigger backtracking.

Example:



Implementation:

```
#include <stdio.h>

int n, board[10][10];

void printSolution();
int isSafe(int row, int col);
int solveNQueen(int col);

void main() {
   printf("Enter the number of queens: ");
   scanf("%d", &n);

   // Initialize board with 0
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         board[i][j] = 0;

   if (solveNQueen(0))
      printSolution();
   else
```

```
        printf("Solution does not exist.\n");
}

// Function to check if a queen can be placed at board[row][col]
int isSafe(int row, int col) {
    int i, j;

    // Check the left side of the row
    for (i = 0; i < col; i++)
        if (board[row][i])
            return 0;

    // Check upper diagonal on left side
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j])
            return 0;

    // Check lower diagonal on left side
    for (i = row, j = col; i < n && j >= 0; i++, j--)
        if (board[i][j])
            return 0;

    return 1;
}

// Function to solve N-Queen using Backtracking
int solveNQueen(int col) {
    if (col >= n) // If all queens are placed
        return 1;

    for (int i = 0; i < n; i++) {
        if (isSafe(i, col)) {
            board[i][col] = 1; // Place queen

            if (solveNQueen(col + 1)) // Recur to place rest of the queens
                return 1;

            board[i][col] = 0; // Backtrack
        }
    }

    return 0;
}

// Function to print the chessboard
void printSolution() {
    printf("\nSolution:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

```
            if (board[i][j] == 1)
                printf("Q\t");
            else
                printf(".\t");
        }
        printf("\n");
    }
}
```

```
Enter the number of queens: 4

Solution:
.       .       Q       .
Q       .       .       .
.       .       .       Q
.       Q       .       .
```

```
Enter the number of queens: 2
Solution does not exist.
```

Conclusion:

The N-Queen problem demonstrates the power of backtracking in solving complex constraint satisfaction problems. By exploring all potential placements and systematically eliminating invalid options, the algorithm efficiently finds valid configurations. It highlights key problem-solving concepts such as recursion, pruning, and state tracking. The challenge not only strengthens logical thinking but also has practical applications in areas like scheduling and puzzle solving. Despite its simplicity in concept, it scales well with optimization techniques like bitmasking. Ultimately, the N-Queen problem is a classic example that blends elegance and computational depth in algorithm design.