## Experiment 4:

Design n-Queens matrix having first Queen placed. Use backtracking to place remaining Queens to generate the final n-queen‘s matrix.

**Code:**

```
// C++ program to solve N Queen Problem using backtracking

#include <bits/stdc++.h>
#define N 4
using namespace std;

// A utility function to print solution
void printSolution(int board[N][N])
{
	for (int i = 0; i < N; i++)
	{
		for (int j = 0; j < N; j++)
			if(board[i][j])
				cout << "Q ";
			else cout<<". ";
			cout<<"\n";
		}
	}

// A utility function to check if a queen can
// be placed on board[row][col]. Note that this
// function is called when "col" queens are
// already placed in columns from 0 to col -1.
// So we need to check only left side for
// attacking queens
bool isSafe(int board[N][N], int row, int col)
{
	int i, j;

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

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

	// Check lower diagonal on left side
```

```
for (i = row, j = col; j >= 0 && i < N; i++, j--)
	if (board[i][j])
		return false;

return true;
}

// A recursive utility function to solve N
// Queen problem
bool solveNQUtil(int board[N][N], int col)
{
	// base case: If all queens are placed
	// then return true
	if (col >= N)
		return true;

	// Consider this column and try placing
	// this queen in all rows one by one
	for (int i = 0; i < N; i++)
	{

		// Check if the queen can be placed on
		// board[i][col]
		if (isSafe(board, i, col))
		{

			// Place this queen in board[i][col]
			board[i][col] = 1;

			// recur to place rest of the queens
			if (solveNQUtil(board, col + 1))
				return true;

			// If placing queen in board[i][col]
			// doesn't lead to a solution, then
			// remove queen from board[i][col]
			board[i][col] = 0; // BACKTRACK
		}
	}

	// If the queen cannot be placed in any row in
	// this column col then return false
	return false;
}

// This function solves the N Queen problem using
// Backtracking. It mainly uses solveNQUtil() to
```

```
// solve the problem. It returns false if queens
// cannot be placed, otherwise, return true and
// prints placement of queens in the form of 1s.
// Please note that there may be more than one
// solutions, this function prints one of the
// feasible solutions.
bool solveNQ()
{
        int board[N][N] = { { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 },
                            { 0, 0, 0, 0 } };

        if (solveNQUtil(board, 0) == false)
        {
                cout << "Solution does not exist";
                return false;
        }

        printSolution(board);
        return true;
}

// Driver program to test above function
int main()
{
        solveNQ();
        return 0;
}
```

**Output :**