Bharatiya Vidya Bhavan's
Sardar Patel Institute of Technology
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India
(Autonomous College Affiliated to University of Mumbai)

NAME:	Gitanjali Gangurde
UID NO:	2021300034
BATCH:	B
EXP NO:	7

Aim: To use Backtracking to solve the N Queen problem

Theory:

- **Backtracking**

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time.

Backtracking can also be said as an improvement to the brute force approach. So basically, the idea behind the backtracking technique is that it searches for a solution to a problem among all the available options.

Initially, we start the backtracking from one possible option and if the problem is solved with that selected option then we return the solution else we backtrack and select another option from the remaining available options. There also might be a case where none of the options will give you the solution and hence we understand that backtracking won't give any solution to that particular problem

Algorithm: is

safe(chessboard,row, col, n)

```
1. for i = 0 to row - 1
2.     if chessboard[i][col] == 1
3.         return false
4. for i = row, j = col; i >= 0 and j >= 0; i--, j--
5.     if chessboard[i][j] == 1
6.         return false
7. for i = row, j = col; i >= 0 and j < n; i--, j++
8.     if chessboard[i][j] == 1
```

9. return false
10. return true

solve_nqueen(chessboard, row, n)

1. if row == n
2. print "Solution " ++ sol_count ++ ":"
3. print_chessboard(chessboard, n)
4. return
5. for i = 0 to n - 1
6. if is_safe(chessboard, row, i, n)
7. chessboard[row][i] = 1
8. solve_nqueen(chessboard, row + 1, n)
9. chessboard[row][i] = 0

PROGRAM:

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 100

int board[MAX_SIZE], solutions = 0;

void printBoard(int n) {
    int i, j;
    printf("Solution %d:\n", solutions++);
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (board[i] == j) {
                printf("Q ");
            } else {
                printf("- ");
            }
        }
        printf("\n");
    }
}
```

```c
int isSafe(int row, int col) {
    int i;
    for (i = 0; i < row; i++) {
        if (board[i] == col || abs(board[i] - col) == abs(i - row)) {
            return 0;
        }
    }
    return 1;
}

void solve(int n, int row) {
    int col;
    if (row == n) {
        printBoard(n);
    } else {
        for (col = 0; col < n; col++) {
            if (isSafe(row, col)) {
                board[row] = col;
                solve(n, row + 1);
            }
        }
    }
}

int main() {
    int n;
    printf("Enter the number of queens: ");
    scanf("%d", &n);
    solve(n, 0);
    return 0;
}
```

Output:

```
Enter the number of queens: 5
Solution 0:
Q - - - -
- - Q - -
- - - - Q
- Q - - -
- - - Q -
Solution 1:
Q - - - -
- - - Q -
- Q - - -
- - - - Q
- - Q - -
Solution 2:
- Q - - -
- - - Q -
Q - - - -
- - Q - -
- - - - Q
Solution 3:
- Q - - -
- - - - Q
- - Q - -
Q - - - -
- - - Q -
Solution 4:
- - Q - -
Q - - - -
- - - Q -
- Q - - -
- - - - Q
Solution 5:
- - Q - -
- - - - Q
- Q - - -
- - - Q -
Q - - - -
```

```
Solution 5:
- - Q - -
- - - - Q
- Q - - -
- - - Q -
Q - - - -
Solution 6:
- - - Q -
Q - - - -
- - Q - -
- - - - Q
- Q - - -
Solution 7:
- - - Q -
- Q - - -
- - - - Q
- - Q - -
Q - - - -
Solution 8:
- - - - Q
- Q - - -
- - - Q -
Q - - - -
- - Q - -
Solution 9:
- - - - Q
- - Q - -
Q - - - -
- - - Q -
- Q - - -
PS D:\anshu\Documents\Sem 4>
```

Observation:

- The n-queen problem has a significant computational complexity for large board sizes.
- The number of solutions increases rapidly with increasing board size.
- Backtracking algorithms like the one used in this implementation can efficiently find all solutions.

The time complexity is O(n!) as n queen is an combinatorics problem, we are just arranging n items.

Conclusion:

In conclusion, the n-queen problem is a challenging and computationally complex problem. However, with backtracking algorithms, it is possible to efficiently find all solutions for small to moderate board sizes.