Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

ACADEMIC YEAR: 2024-25

Course: Analysis of Algorithm Lab

Course code: CSL401

Year/Sem: SE/IV

Experiment No.: 08
Aim: To implement the N Queen Problem using a backtracking technique
Name: SOHAM HEMENDRA RAUT
Roll Number: 24
Date of Performance: 20/03/2025
Date of Submission: 27/03/2025

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	5	3	2
Understanding	5	3	2
Journal work and timely submission.	10	8	4

Checked by

Name of Faculty : Mrs. Komal Champanerkar

Signature :

Date :

- **Aim: To implement the N Queen Problem using a backtracking technique**
- **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.

The expected output is in form of a matrix that has 'Q's for the blocks where queens are placed and the empty spaces are represented by '.'s . For example, the following is the output matrix for the above 4 queen solution.

```
. . Q .
Q . . .
. . . Q
. Q . .
```

The idea is to place queens one by one in different columns, starting from the leftmost column. When we place a queen in a column, we check for clashes with already-placed queens. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution. If we do not find such a row due to clashes, then we backtrack and return false.

- **Algorithm:**

Step 1: Initialize the empty chessboard of size N×N

Step 2: Start with the leftmost column and place the queen in the first row of that column.

Step 3: Move to the next column and place the queen in the first row of that column.

Step 4: Repeat step 3 until all N queens have been placed or it is impossible to place the queen in the current column without violating the rules of the problem.

Step 5: If all N queens have been placed, print the solution.

Step 6: If it is not possible to place a queen in the current column without violating the rules os the problem, then backtrack to the previous column.

Step 7: Remove the queen from the previous column and move it to the next row.

Step 8: Repeat steps 4-7 until all the possible configurations have been tried.

❖ **Program:**

```
def print_solution(board):
    for row in board:
        print(" ".join("Q" if cell else "." for cell in row))
    print()

def is_safe(board, row, col, n):
    # Check this column on upper side
    for i in range(row):
        if board[i][col]:
            return False

    # Check upper diagonal on left side
    i, j = row, col
    while i >= 0 and j >= 0:
        if board[i][j]:
            return False
        i -= 1
        j -= 1

    # Check upper diagonal on right side
    i, j = row, col
    while i >= 0 and j < n:
        if board[i][j]:
            return False
        i -= 1
        j += 1

    return True

def solve_n_queens_util(board, row, n):
    if row >= n:
        print_solution(board)
        return True

    res = False
    for i in range(n):
```

```
        if is_safe(board, row, i, n):
            board[row][i] = 1
            res = solve_n_queens_util(board, row + 1, n) or res
            board[row][i] = 0  # Backtrack
    return res

def solve_n_queens(n):
    board = [[0 for _ in range(n)] for _ in range(n)]
    if not solve_n_queens_util(board, 0, n):
        print("No solution exists")

n = int(input("Enter the value of N: "))
solve_n_queens(n)
```

❖ Output:

```
PS C:\Users\SOHAM> & C:/Users/SOHAM/anaconda3/python.exe c:/Users/SOHAM/.conda/N-Queen.py
Enter the value of N: 4
. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. . . Q
. Q . .
```

❖ Conclusion:

The N-Queens problem, when solved using backtracking, has a worst-case time complexity of O(N!), as it explores potential placements for each queen. However, effective pruning can reduce the best-case complexity to O(N). The space complexity is typically O(N^2), but it can be optimized to O(N) with improved representations. Backtracking is a systematic, depth-first search strategy that eliminates invalid configurations, making it an efficient method for solving combinatorial problems. Despite this efficiency, it may still experience exponential time complexity in larger cases.