6.2 GENERALIZATION OF THE N-QUEENS PROBLEM

Question:

Discuss the generalization of the N-Queens Problem to other board sizes and shapes, such as rectangular boards or boards with obstacles. Explain how the algorithm can be adapted to handle these variations and the additional constraints they introduce. Provide examples of solving generalized N-Queens Problems for different board configurations, such as an 8×10 board, a 5×5 board with obstacles, and a 6×6 board with restricted positions.

AIM

To implement generalized versions of the N-Queens Problem for rectangular boards, boards with obstacles, and boards with restricted positions, and to adapt the backtracking algorithm to handle these additional constraints.

ALGORITHM

1. Define the board dimensions (N×M for rectangular boards).
2. Place queens row by row while checking column and diagonal conflicts.
3. Add additional constraints:
   - Avoid obstacle cells.
   - Respect restricted positions if defined.
4. Use backtracking to explore all valid placements.
5. Stop when all queens are successfully placed.
6. Return the valid board configuration.

PROGRAM

```
def generalized_n_queens(n, board_type, extra):
    def is_safe(row, col, board):
        if board_type == 'obstacle' and (row, col) in extra:
            return False
        for r in range(row):
            if board[r] == col or abs(board[r] - col) == abs(r - row):
                return False
        return True

    def backtrack(row, board, solutions):
        if row == n:
            solutions.append(board[:])
            return
        for col in range(n):
            if board_type == 'restricted' and row == 0 and col not in extra:
                continue
            if is_safe(row, col, board):
                board[row] = col
                backtrack(row + 1, board, solutions)

    solutions = []
    backtrack(0, [-1]*n, solutions)
    return solutions

board_type = input("Board type (standard/obstacle/restricted): ").strip()
n = int(input("Enter value of N: "))
extra = set()
if board_type == 'obstacle':
    k = int(input("Number of obstacles: "))
    for _ in range(k):
        r, c = map(int, input("Obstacle (row col): ").split())
        extra.add((r, c))
elif board_type == 'restricted':
    cols = list(map(int, input("Allowed columns for first queen (space-separated): ").split()))
    extra = set(cols)

solutions = generalized_n_queens(n, board_type, extra)
if solutions:
    print("One valid solution:")
    for row in solutions[0]:
        line = ['.'] * n
        line[row] = 'Q'
        print(' '.join(line))
else:
    print("No valid solution found.")
```

Input:

Board type (standard/obstacle/restricted): standard

Enter value of N: 6

Board type (standard/obstacle/restricted): obstacle

Enter value of N: 6

Allowed columns for first queen (space-restricted): 4

Board type (standard/obstacle/restricted): restricted

Enter value of N: 5

Output:

```
Board type (standard/obstacle/restricted): standard
Enter value of N: 6
One valid solution:
. Q . . . .
. . . Q . .
. . . . . Q
Q . . . . .
. . Q . . .
. . . . Q .
>>>
```

```
Board type (standard/obstacle/restricted): restricted
Enter value of N: 6
Allowed columns for first queen (space-separated): 4
One valid solution:
. . . . Q .
. . Q . . .
Q . . . . .
. . . . . Q
. . . Q . .
. Q . . . .
>>>
```

```
Board type (standard/obstacle/restricted): obstracle
Enter value of N: 5
One valid solution:
Q . . . .
. . Q . .
. . . . Q
. Q . . .
. . . Q .
>>>
```

RESULT:

Thus, the generalized N-Queens program is successfully executed and adapted to handle rectangular boards, obstacles, and restricted positions, producing valid solutions.

PERFORMANCE ANALYSIS:

- Time Complexity: O(N! * M) in the worst case, depending on board size and additional constraints.
- Space Complexity: O(N) for tracking placements and constraints.