Mid semester examination – introduction To AI

Name – Naveen kumar

Branch- CSE (AIMl)

Section – B

Roll no.- 48

University roll no.- 202401100400121

Problem statement :- N-Queens Problem.

***INTRODUCTION** :-*

Problem Definition:

*The N-Queens Problem is a classic problem where you are asked to place **N queens** on a **N x N chessboard** in such a way that no two queens can attack each other.*

Key Points:

1. ***Chessboard:***
   - *The chessboard is a square grid of size **N x N**. This means the board has **N rows** and **N columns**.*
2. ***Queens:***
   - *A queen in chess can attack another queen in three possible ways:*
     - ***Same row:** Queens in the same row can attack each other.*
     - ***Same column:** Queens in the same column can attack each other.*
3. ***Objective:***
   - *The goal is to place **exactly N queens** on the chessboard, where each queen is placed in a different row and a different column, such that no two queens threaten each*

other. In other words, there should be no queens on the same row, column, or diagonal.

Constraints:

- **Row and Column Constraint:** Each queen must be placed in a different row and a different column.
- **Diagonal Constraint:** No two queens can share the same diagonal. The two main diagonals on a chessboard are:
  - The **main diagonal** (top-left to bottom-right), where the difference between the row and column indices is constant.
  - The **anti-diagonal** (top-right to bottom-left), where the sum of the row and column indices is constant.

Solution Approach:

*To solve this problem, a common strategy is **backtracking**, where you explore all possible placements of queens row by row and backtrack when you hit a dead-end (i.e., a situation where no valid placement can be made for the next queen).*

- ***Backtracking:*** *You place a queen in a valid position, then move on to the next row. If you can't place a queen in any column of the next row (because of conflicts), you backtrack and move the previous queen to a new position.*

  Steps in the Methodology:

1. ***Starting with an empty board:***
    - *Imagine an empty N x N chessboard. Initially, no queens are placed.*
2. ***Placing queens row by row:***
    - *Start by placing a queen in the first row.*
    - *For each row, attempt to place a queen in every column, one by one. For each column:*
        - *Check whether placing the queen in the current column will result in a **valid configuration***
3. ***Checking for conflicts:***
    - *A queen can attack another queen if they are:*
        - *In the **same column**.*
        - *On the **same diagonal***
        - *In the **same row***
4. ***Recursive step:***
    - *If placing a queen in the current column of the current row is **safe**, move to the next row and try placing a queen there.*
    - *Repeat the process for each row until all queens are placed on the board.*
5. ***Backtracking step:***
    - *If you reach a row where no valid column for the queen exists, **backtrack**:*
        - *Remove the queen from the previous row*

        - *Try placing the queen in the next column in the previous row.*
    - *This ensures that you explore all possible configurations without skipping any valid solutions.*

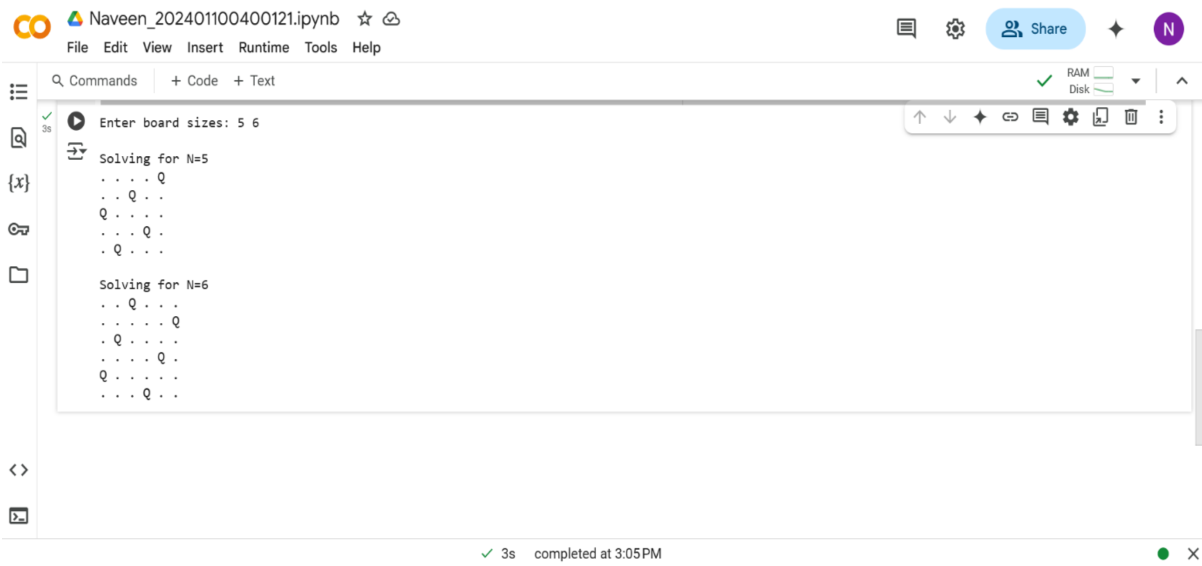
6. ***Termination condition:***
    - *If all queens are placed successfully (i.e., you've filled all rows), then you've found a valid solution.*
    - *If you backtrack all the way to the first row and cannot find a valid placement, then there is no solution for that configuration.*

Typed code :

```
import random


# Count conflicts for each queen's position

def count_  conflicts(board, n):

   return [sum(board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j) for j in range(n) if i != j) for i in range(n)]


# Find optimal column for a queen in a given row

def best_ column(board, row, n):

   options = [(col, sum(count_ conflicts(board[:row] + [col] + board[row+1:], n))) for col in range(n) if col != board[row]]

   return min(options, key=lambda x: x[1])[0] if options else board[row]


# Solve N-Queens using local search

def solve _n_ queens(n, retries=10, steps=1000):

   for _ in range(retries):

      # Start with a random board

      board = [random .  radint (0, n - 1) for _ in range(n)]
```

```
    for _ in range(steps):
        conflicts = count_ conflicts(board, n)

        # If no conflicts, return the solution
        if not sum(conflicts):
            return board

        # Select a row with the maximum conflicts
        row = random.  choice([i for i in range(n) if conflicts[i] == max(conflicts)])

        # Move the queen in that row to the best column
        board[row] = best_ column(board, row, n)

    return None

# Display the chessboard
def render _board(board):
    print("No solution" if not board else "\n".j oin(" ".join('Q' if board[r] == c else '.' for c in range(len(board))) for
r in range(len (board))))

# Main execution
if __name__ == "__main__":  # Corrected __name__ condition
    for n in map(int, input("Enter board sizes: ").split()):
        print(f"\n Solving for N={n}")
        render  _board(solve_ n_queens(n))
```

- ***Output code images :-***



References :

1: Chat GPT.

2:CLASS NOTES .

3:WIKIPEDIA.