PROJECT REPORT
(PROJECT TERM - SUMMER TERM)

N QUEEN VISUALIZER

SUBMITTED BY
KAVIYADHARSHINI M
12205433

SECTION 9SK01

COURSE -

CSES003

ROLL NO - 03

SCHOOL OF COMPUTER SCIENCE AND

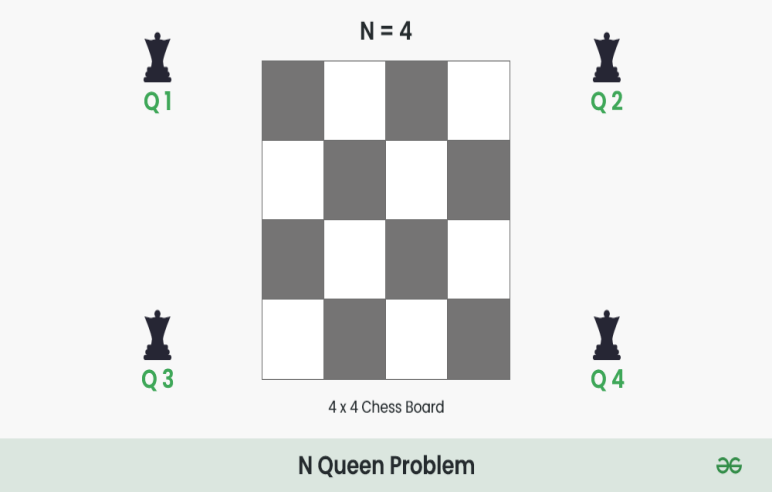
ENGINEERING

LOVELY
PROFESSIONAL
UNIVERSITY

TABLE OF CONTENTS

Introduction

The N-Queen problem is a classic example in computer science and algorithm design, where the objective is to place N queens on an N×N chessboard such that no two queens threaten each other. This project presents a graphical visualization of the N-Queen problem using Java Swing. The visualization helps in understanding the backtracking algorithm used to solve the problem by providing real-time feedback on the placement of queens and the conflicts encountered during the process.



Objective

Goal of my project are pointed below :

- The goal of my project is to place N queens on an N x N chessboard, in a way so that no queen is in conflict with the others.

- In my project I used backtracking algorithm. As we know a backtracking algorithm is a problem-solving algorithm that tries out all the possible solutions and chooses the best solutions.

- It removes the solutions that doesn't give rise to the solution of the problem based on the constraints given to solve the problem.

- The project will make us more confident towards to solve some specific types of problems such as Decision problem, Optimisation problem, Enumeration problem using Backtraking algorithm.

- In Backtracking, we first start with a partial sub-solution of the problem and then check if we can proceed further with this sub-solution or not. If not, then we just come back and change it. So, using this approach anyone can easily solve these types of problems.

Algorithm

Backtracking:
Backtracking technique is used to place queens in each row at right place. Backtracking constructs candidate solutions one component at a time and evaluates the partially constructed solutions. This approach makes it possible to solve some large instances of difficult combinatorial problems, though, in the worst case, we still face the same curse of exponential explosion encountered in exhaustive search.

Approach

Step 1: Start in the leftmost column

Step 2: If all queens are placed return true

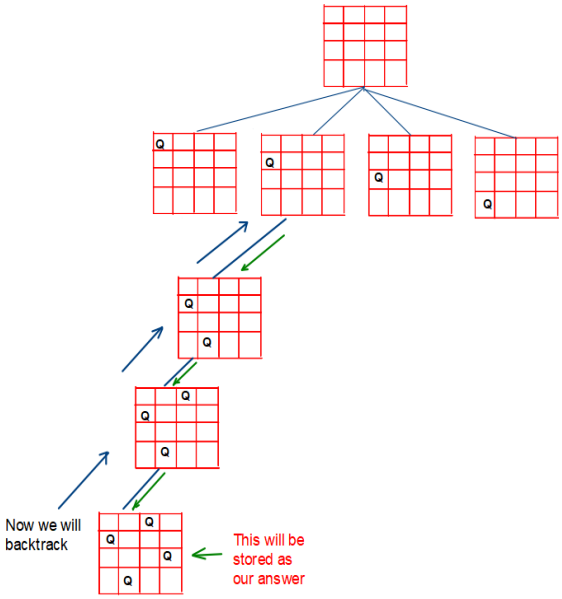
Step 3: Try all rows in the current column. Do following for every tried row.

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing the queen here leads to a solution.

b) If placing the queen in [row, column] leads to a solution then return true.

c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

Step 4: If all rows have been tried and nothing worked, return false to trigger backtracking.



Design

Visualization and User Experience:

The program consists of two main classes:

1. **NQueenVisualizer**
2. **NQueenPanel**

NQueenVisualizer

This class sets up the main frame for the visualization.

- **JFrame Setup:**
  - Creates a JFrame titled "N-Queen Visualizer".
  - Configures the frame to exit the application when closed.
  - Sets the size of the frame to 800x800 pixels.

- **Adding the NQueenPanel:**
  - Instantiates an `NQueenPanel` with a specified board size (e.g., 5).
  - Adds the panel to the frame.
- **Thread for Placing Queens:**
  - Starts a new thread to execute the `placeQueens` method from `NQueenPanel` to ensure the GUI remains responsive.

The visualization process is designed to be interactive and informative:

- **Chessboard Pattern:**
  - Alternating colors (white and gray) provide a clear view of the board.
- **Queen Placement:**
  - Queens are represented by black ovals.
- **Conflict Highlighting:**
  - Unsafe positions are highlighted in red and orange to indicate conflicts.
  - Delays are added to allow users to see the backtracking process in action.
- **Real-time Updates:**
  - The panel is repainted after each significant step to show the current state of the board.

NQueenPanel

This class extends `JPanel` and handles the visualization and logic for solving the N-Queen problem.

- **Fields:**
  - `N`: The size of the board.
  - `board`: An array storing the column positions of queens for each row.
  - `cellColors`: A 2D array representing the colors of each cell on the board.

    - `currentRow` and `currentCol`: Track the current position being processed.
- **Constructor:**
    - Initializes the board and cellColors arrays.
    - Sets the preferred panel size.
    - Alternates cell colors to create a chessboard pattern.
- **paintComponent Method:**
    - Overrides `paintComponent` to draw the chessboard and queens.
    - Calls `drawBoard` for the actual drawing.
- **drawBoard Method:**
    - Calculates cell size based on panel width.
    - Iterates through each cell, setting its color and drawing it.
    - Draws a black oval for queens.
- **placeQueens Method:**
    - Calls the `placeQueensOnRow` method starting from the first row.
- **placeQueensOnRow Method:**
    - Implements the recursive backtracking algorithm to place queens.
    - Attempts to place a queen in each column of the current row.
    - Updates `currentRow` and `currentCol` for visualization.
    - Repaints the panel and adds a delay for visualization.
- **isSafe Method:**
    - Checks if placing a queen at a specific position is safe.
    - Ensures no other queen is in the same column or diagonal.

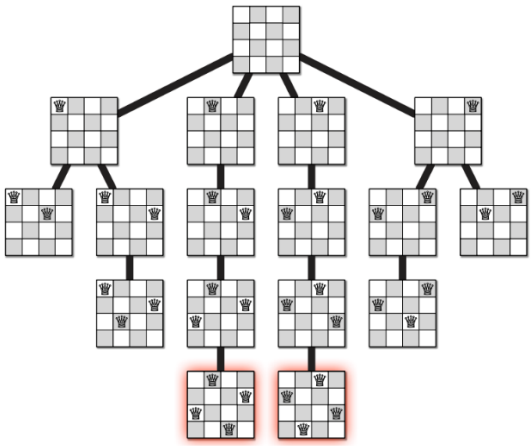
Backtracking:

The core of the solution is the backtracking algorithm implemented in the `placeQueensOnRow` method. Here's a breakdown of how it works:

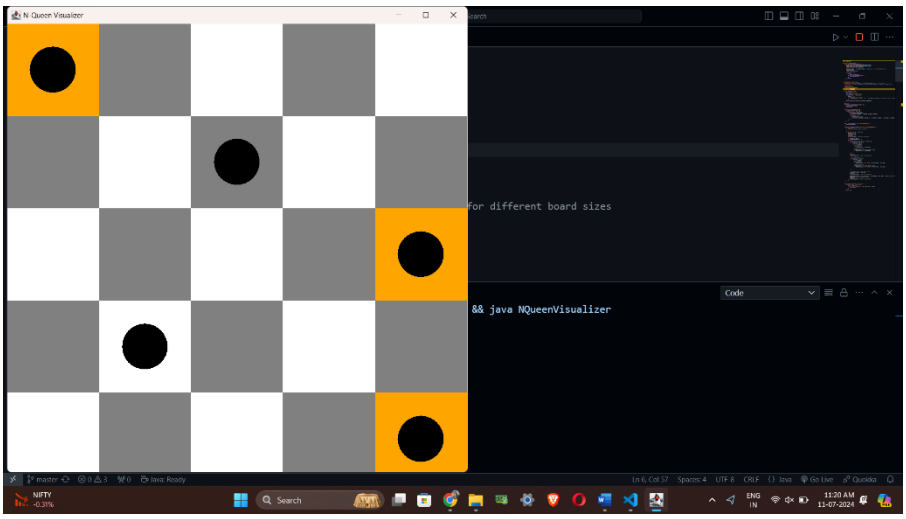
1. **Base Case:**
    - If all queens are placed successfully (row == N), the algorithm returns.
2. **Recursive Case:**
    - For each column in the current row:
        - Place a queen at (row, col).

- Update the current position and repaint the panel.
- Add a delay to visualize the placement.
- If the position is safe (checked by `isSafe`), recursively place queens in the next row.
- If not safe:
  - Highlight conflicts.
  - Revert cell colors and remove the queen.

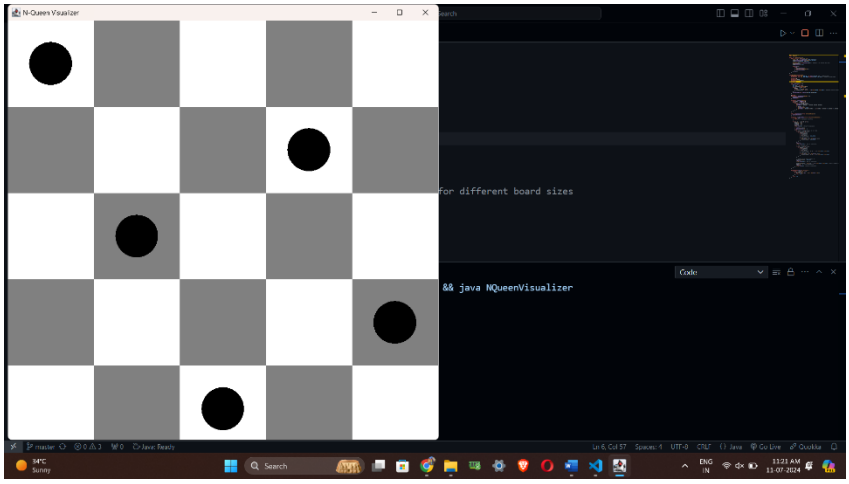
3. **isSafe Method:**
   - Checks each previously placed queen to ensure no two queens threaten each other.
   - Validates column and diagonal positions



Output



Shows the queens that are making it unsafe in orange



One of the solution that is found

Conclusion

The N-Queen Visualizer is a practical tool for understanding the complexities of the N-Queen problem and the backtracking algorithm used to solve it. By providing a clear and interactive graphical representation, it allows users to see the step-by-step process of placing queens, detecting conflicts, and backtracking to find a solution. This visualization enhances learning and comprehension of algorithm design and problem-solving techniques in computer science.