Project Report

N-Queens Problem Visualization in C++

Submitted by:

Name : Chandan Singh Rajput
Reg No: 12220324
Course Code: CSES003
Section: 9SK02

Introduction:

The *N*-Queens problem originates from a question relating to chess, The 8-Queens problem. Chess is played on an 8 × 8 grid, with each piece taking up one cell. A queen is a piece in chess that, in any given move, can move any distance vertically, horizontally, or diagonally. However, the queen cannot move more than one direction per turn. It can only move one direction per turn. So a question one might ask themselves is whether or not you can place 8 queens on a chessboard so that none of the queens can kill each other in one move (i.e. There is no way for a queen to in one cell to reach a queen in another cell in one move)?
The answer is yes!

Background and Problem Statement:

The N-Queens problem is a classical chess puzzle that involves placing N queens on an N×N chessboard such that no two queens threaten each other. A queen can attack any piece in its same row, column, or diagonal. This problem, first introduced as the 8-Queens problem for an 8×8 chessboard, has since been generalized to boards of any size, known as the N-Queens problem.

The objective is to find all possible arrangements of N queens on the board where no two queens can attack each other. This constraint ensures that each queen is placed in a position that is safe from any other queen's potential move. The problem is not merely a puzzle but also has practical applications in computer science, algorithm design, and constraint satisfaction.

Key challenges include the exponential growth of solutions as N increases and the computational complexity involved in verifying and finding these solutions. Despite its seemingly straightforward rules, the problem's solution requires careful consideration of board symmetries, efficient search algorithms (like backtracking), and combinatorial techniques such as Latin squares and pandiagonal Latin squares.

Understanding and solving the N-Queens problem not only showcase fundamental concepts in chess and combinatorial mathematics but also provide insights into algorithmic efficiency and problem-solving strategies essential in various fields of computer science and beyond.

Methodology:

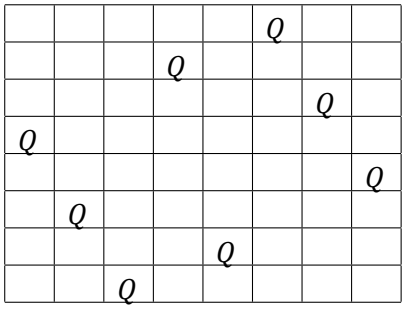
To tackle the N-Queens problem, various methodologies and strategies have been developed, each aimed at efficiently placing N queens on an N×N chessboard without them threatening each other. The primary approaches include:

Backtracking Algorithm:

- The most common method is the backtracking algorithm, which systematically explores potential placements of queens on the board. Starting from the first row, it attempts to place a queen in each column of the current row, ensuring that no two queens share the same row, column, or diagonal. If a conflict arises (i.e., a queen threatens another), it backtracks to explore alternative placements until a valid configuration or all possibilities are exhausted.

Each methodology has its strengths and weaknesses, depending on the specific problem constraints, board size, and computational resources available. Experimentation with different approaches continues to refine our understanding and efficiency in solving the N-Queens problem across various domains of research and application.

Q = Queen



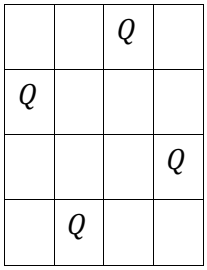
There are 12 unique solutions to this problem. Two solutions are not unique if you can ”mirror” one solution to find the other, or if you can rotate the board to find the other solution, or a combination of the two moves.

We can generalize the 8-Queens problem to be the N-Queens problem.

Given an $n \times n$ chessboard, can we place n queens on the chessboard so that none of the queens can kill each other in one move?

Example.

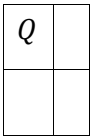
n= 4:



One may think we can find solutions for all values of n, upon trying a few small values of n, we find that no solutions exist for n = 2 or n = 3. Here we see why:

n = 2:

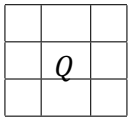
The only option we have is to place a queen in the corner of board.



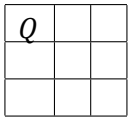
No matter where we put the first queen, it can always move to any other spot on the board in one move. Thus, we cannot place a second queen to satisfy the *N*-Queens problem.

$n = 3$:

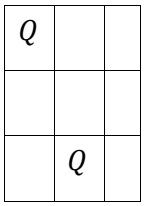
This doesn't work for a reason similar to why $n = 2$ doesn't work. If we place a queen in the center of the board, then it can always move to any other spot on the board in one move.



So placing a queen at the center of the board does not let us place a second queen. So we can try placing a queen in the corner of the board.

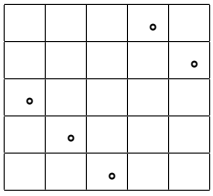


Now we have to cells that the queen cannot reach in one move. So we can now place a second queen.

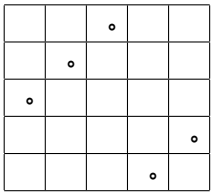


However, putting a queen in any of those two cells makes it so that both queens can reach any other cell on the board. Therefore we do not have the option to place a third queen. Any place we put the first queen leaves (at most) two other places to put queens without conflict with the first queen, but the 2nd and 3rd queens will always be able to reach each other in one move, so $n = 3$ cannot satisfy the *N*-Queens problem.

The *N*-Queens problem seems inherently similar to concepts of Latin Squares. It turns out Latin Squares are in fact useful when working with the *N*-Queens problem!



Definition. A **broken left diagonal** is the same as a broken right diagonal, except we wrap around to the left instead of the right. **Example.**



The concept of broken diagonals will give us the special type of Latin square that will help us find solutions to the *n*-queens problem.

Results:

The exploration of the N-Queens problem has yielded significant insights and solutions across various board sizes. For instance, solutions exist for all N ≥ 4, with the number of distinct solutions growing exponentially as N increases. Extensive computational efforts have enumerated solutions up to N = 26, revealing precise counts of different configurations where N queens can be placed on an N×N chessboard without threatening each other.

Discussion:

The exponential growth in the number of solutions highlights the increasing complexity of the problem as N expands. For example, while there are only 2 distinct solutions for N = 4, the number escalates to 92 for N = 8 and continues to increase dramatically beyond. Interestingly, some board sizes exhibit unexpected behaviors, such as N = 5 having more solutions than N = 6, emphasizing the non-linear nature of the solution space.

Various methodologies, including backtracking algorithms, constraint satisfaction techniques, and combinatorial structures like Latin squares, have been instrumental in exploring and validating these solutions. Techniques to handle symmetries and optimize computational efficiency play crucial roles in exhaustively searching or efficiently pruning the solution space.

Conclusion:

In conclusion, the N-Queens problem serves as a fundamental challenge in combinatorial mathematics and computer science. Its solutions not only demonstrate the intricate interplay between constraints and arrangements but also underscore the application of algorithmic techniques in solving complex puzzles. While solutions are well-documented for smaller board sizes, the quest for solutions and insights continues for larger boards, where computational limits and the exponential growth of possibilities pose ongoing challenges.

Further research may explore heuristic approaches, parallel computing strategies, and deeper theoretical insights into the structure of solution spaces to expand our understanding and capabilities in solving combinatorial problems like the N-Queens problem. As computational resources evolve, so too will our ability to explore and apply these solutions in practical domains such as scheduling, optimization, and constraint satisfaction in real-world applications.

This discussion and conclusion highlight the enduring relevance and complexity of the N-Queens problem, paving the way for continued exploration and innovation in algorithmic design and mathematical theory.