N Queen Problem

The N Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.

Introduction

The **N** Queen is the problem of placing **N** chess queens on an **N×N** chessboard so that no two queens attack each other. This problem represents a classic challenge in computer science, demonstrating algorithmic techniques like backtracking to find valid queen configurations.

Pros:

1. Enhances understanding of problem-solving algorithms.
2. Teaches constraint satisfaction techniques.
3. Valuable for educational purposes.
4. Used as a benchmark for algorithm evaluation.

Cons:

1. Simple nature may limit real-world relevance.
2. Direct applications might be limited.
3. Risk of dependence on standard algorithms.
4. Narrow focus on spatial constraints.

Working Procedure

1

Backtracking Algorithm Basics

- Initialize an NxN chessboard with empty spaces. Starting at the first row, check each column for safe queen placement using "`issafe`". If safe, mark with 'Q' and move to the next row. If not, backtrack, undo the last decision, and explore the next column. Repeat until all valid configurations are found.

2

Recursive Approach

Backtracking involves a recursive approach where the algorithm explores different branches of the solution space. It prunes the search tree whenever it realizes that a part of the arrangement will not lead to a solution.

3

Complexity Analysis

The backtracking algorithm for the N-Queens problem has a time complexity of O(N!), which grows rapidly with increasing N. Understanding the algorithm's complexity is crucial for assessing its scalability and performance impact.

Implementing the algorithm in code

Application of N Queen algorithm?

One area where the N-Queens problem's algorithms can be applied is in **optimization and constraint satisfaction problems**. These algorithms can be used in tasks such as scheduling, resource allocation, and logistics, where the goal is to find an optimal arrangement that satisfies certain constraints.

Dataset:

- **Input:** The size of the chessboard (N). In the provided Java code example, **int n=10 ;** sets the chessboard size to 10×10.
- **Output:** The program generates and prints all valid configurations of placing N queens on an NxN chessboard.

Backtracking Algorithm

Backtracking is a problem-solving method where decisions are made incrementally, exploring possible solutions. If a choice leads to an invalid solution, the algorithm backtracks to the last decision point. It's versatile, used for N-Queens, Sudoku, and maze solving, emphasizing efficiency by systematically searching through decision spaces.

Code of N-queen problem

```
public class Q_chess {
    public static boolean issafe(char board[][],int row,int col){
        //vertical up
        for(int i=row-1;i>-1;i--){
            char idx=board[i][col];
            if(idx=='Q'){
                return false;
            }
        }
        //digonal left up
        for(int i=row-1,j=col-1;i>=0 && j>=0;i--,j--){
            if(board[i][j]=='Q'){
                return false;
            }
        }
        //digonal right up
        for(int i=row-1,j=col+1;i>=0 && j< board.length;i--,j++){
            if(board[i][j]=='Q'){
                return false;
            }
        }
        return true;
    }
 public static void print(char board[][]){
        System.out.println("-------chess board-------");
        for(int i=0;i<board.length;i++){
            for(int j=0;j<board.length;j++){
                System.out.print(board[i][j]+"  ");
            }
            System.out.println();
```

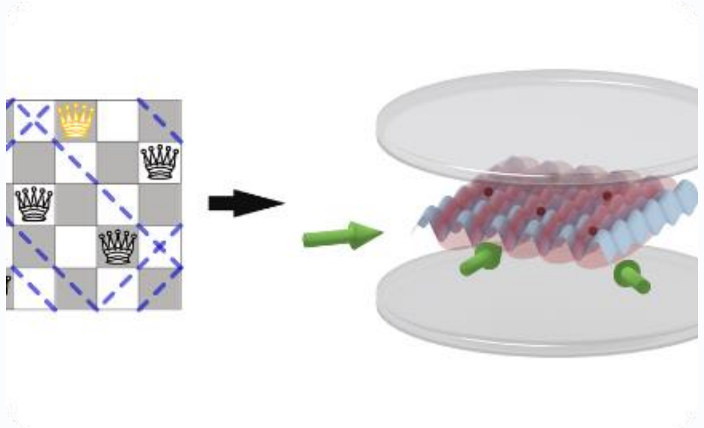
```
public static void nQueen(char board[][],int row){
        //base case
        if (row ==board.length){
            print(board);
            return;
        }
        for(int i=0;i<board.length;i++){
            if(issafe(board,row,i)) {
                board[row][i] = 'Q';
                nQueen(board, row + 1);
                board[row][i] = 'x';
            }
        }
    }
public static void main(String[] args) {
        int n=10;
        char board[][]=new char[n][n];
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                board[i][j]='x';
            }
        }
        nQueen(board,0);
    }
```

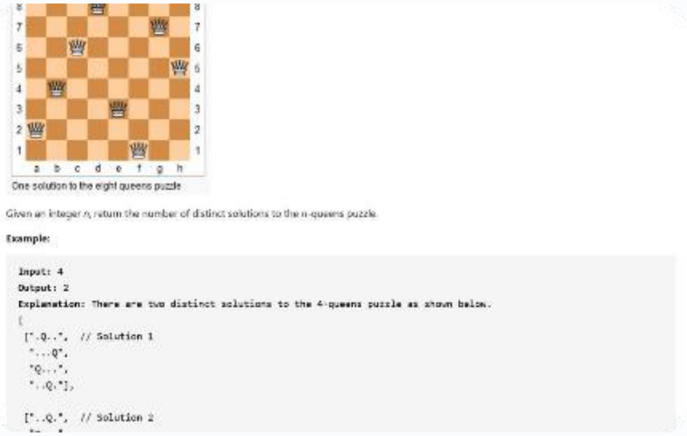
Output of 10×10 Chessboard

Q x x x x x x x x x
x x Q x x x x x x x
x x x x x Q x x x x
x x x x x x x Q x x
x x x x x x x x x Q
x x x x Q x x x x x
x x x x x x x x Q x
x Q x x x x x x x x
x x x Q x x x x x x
x x x x x x Q x x x

For more better understanding visit-
https://github.com/albertanon/AI-PROJECT

Visualizing the solution on an N*N chessboard





Classic 8x8 Chessboard

Visual representation of the N-Queens problem solution on a standard 8x8 chessboard, showcasing the arrangement of queens without conflicts.

Compact 4x4 Chessboard

An alternative visualization illustrating the N-Queens solution on a smaller 4x4 chessboard with reduced space, highlighting the adaptability of the algorithm.

Larger 10×10 Chessboard

Representation of the N-Queens puzzle solution on a larger 10×10 chessboard, demonstrating the scalability and effectiveness of the solver for bigger board sizes.

Conclusion

The N-Queens problem is like a puzzle where you place queens on a chessboard so they don't attack each other. People use a smart method called "backtracking" to try different placements, undoing if needed. It's an old but important puzzle that helps us learn how to solve problems step by step.

Future Work

In the future, the N-Queens problem might help make things work better and smarter in areas like organizing schedules or designing networks. It could also teach computers to recognize patterns and solve real-world challenges in robots or other technologies, making them more efficient.

Reference

1. Google
2. ChatGPT 3.5
3. Geeks for Geeks