N-Queens Problem: A Backtracking Approach

Varun Kumar

July 8, 2025

Contents

Listings

List of Algorithms

1 Introduction

The N-Queens problem is a classic backtracking puzzle where the goal is to place N queens on an $N \times N$ chessboard such that no two queens threaten each other.

Constraints:

- No two queens can be in the same row.
- No two queens can be in the same column.
- No two queens can be on the same diagonal.

2 Problem Statement

Place N queens on an $N \times N$ board so that:

- No queen can attack another.
- Return all distinct solutions (or any one).

3 Approach: Backtracking

- Start with the first row.
- Try placing a queen in each column.
- If placing is safe, move to the next row.
- If not, backtrack and try the next column.

Key Conditions

To check if placing a queen at (row, col) is safe:

- No other queen is in the same column.
- No other queen on upper-left diagonal.
- No other queen on upper-right diagonal.

4 Pseudocode

Algorithm 1 Solve N-Queens

procedure SOLVENQUEENS(n)
    $board \leftarrow$ empty $n \times n$ grid
    $result \leftarrow$ empty list
    BACKTRACK(0)
    return $result$
end procedure
procedure BACKTRACK(row)
    if $row == n$ **then**
        Add current board to $result$
        return
    end if
    for $col = 0$ to $n - 1$ **do**
        if Safe to place at (row, col) **then**
            Place queen at (row, col)
            BACKTRACK($row + 1$)
            Remove queen from (row, col)
        end if
    end for
end procedure

5 Python Implementation

```python
def solve_n_queens(n):
    board = [['.' for _ in range(n)] for _ in range(n)]
    result = []

    def is_safe(row, col):
        for i in range(row):
            if board[i][col] == 'Q': return False
            if col - (row - i) >= 0 and board[i][col - (row - i)] == 'Q': return False
            if col + (row - i) < n and board[i][col + (row - i)] == 'Q': return False
        return True
```

```python

    def backtrack(row):
        if row == n:
            result.append([''.join(r) for r in board])
            return
        for col in range(n):
            if is_safe(row, col):
                board[row][col] = 'Q'
                backtrack(row + 1)
                board[row][col] = '.'

    backtrack(0)
    return result
```

Listing 1: N-Queens in Python

6 C++ Implementation

```cpp
#include <iostream>
#include <vector>
using namespace std;

bool isSafe(int row, int col, vector<string> &board, int n) {
    for (int i = 0; i < row; ++i) {
        if (board[i][col] == 'Q') return false;
        if (col - (row - i) >= 0 && board[i][col - (row - i)] == 'Q') return false;
        if (col + (row - i) < n && board[i][col + (row - i)] == 'Q') return false;
    }
    return true;
}

void solve(int row, vector<string> &board, vector<vector<string>> &result, int n) {
    if (row == n) {
        result.push_back(board);
        return;
    }
    for (int col = 0; col < n; ++col) {
        if (isSafe(row, col, board, n)) {
            board[row][col] = 'Q';
            solve(row + 1, board, result, n);
            board[row][col] = '.';
        }
    }
}

vector<vector<string>> solveNQueens(int n) {
    vector<vector<string>> result;
    vector<string> board(n, string(n, '.'));
    solve(0, board, result, n);
    return result;
}
```

Listing 2: N-Queens in C++

7 Key Points to Remember

1. N-Queens is a classic backtracking problem.
2. A solution is valid if no two queens threaten each other.
3. We only place one queen per row.
4. Use diagonals and column checks to prune invalid states.
5. Base case: all n queens are placed $\rightarrow$ store result.
6. Recursive case: try placing a queen in all columns.
7. If placing fails $\rightarrow$ backtrack and try other positions.
8. Total solutions vary: 92 for N=8.
9. Very useful in solving constraint satisfaction problems.
10. Can be extended to optimization and parallel problems.

8 Real-World Applications

- **Constraint Solvers:** Used in scheduling and resource allocation.
- **Circuit Design:** Placing non-overlapping components.
- **Parallel Processing:** Task assignments without interference.
- **Sudoku Solvers:** Similar combinatorial strategy.
- **AI Problem Solving:** CSP-based agents use similar logic.

9 Conclusion

The N-Queens problem demonstrates a powerful application of backtracking for constraint satisfaction. It helps build strong recursive reasoning, efficient state pruning, and is foundational for problems in AI, search, and optimization domains.