

■ N-Queens

The ultimate backtracking problem — queens, constraints, and clever pruning

● Recursion

■ HARD

■ What You Will Learn

- Classic constraint-satisfaction backtracking
- How to check row, column, and diagonal conflicts efficiently
- Building a board state and pruning invalid placements
- The full backtracking lifecycle: place → recurse → remove
- Why this problem exemplifies all of backtracking at once

■ Intuition First

N queens on an $N \times N$ chessboard — no two queens can attack each other. Queens attack along rows, columns, and diagonals. Strategy: place one queen per row. For each row, try every column. If a column is safe (no conflict), place the queen and move to the next row. If you get stuck, remove the queen and try the next column.

■ Backtracking Core

Place queen in row 0, col 0 → move to row 1 → place in safe col → ... → if no safe col in current row → BACKTRACK to previous row, try next column. This 'try → fail → undo → try next' loop is backtracking.

■ Visual — N=4 Solution

N=4 board (. = empty, Q = queen):

Solution 1:

```
. Q . .  
. . . Q  
Q . . .  
. . Q .
```

Solution 2:

```
. . Q .  
Q . . .  
. . . Q  
. Q . .
```

Conflict check for placing at (row, col):

- ① Same column: any queen already in 'col'?
- ② Top-left diagonal: $\text{row} - \text{col} == \text{prevRow} - \text{prevCol}$?
- ③ Top-right diagonal: $\text{row} + \text{col} == \text{prevRow} + \text{prevCol}$?

Using sets:

cols set: stores used columns

diag1 set: stores (row-col) values

diag2 set: stores (row+col) values

■■ How It Works — Row by Row

1. Start at row 0. Try each column from 0 to N-1
2. For each column, check: is it safe? (not in cols, diag1, diag2 sets)
3. If safe: place queen (update sets + board), recurse on row+1
4. If row == N: all queens placed → add board to result
5. After recursion: remove queen (backtrack — remove from sets + board)
6. If no column works in this row: return (triggering backtrack above)

■ Dry Run — N=4, finding first solution

```
Row 0: try col=0 → safe → place Q at (0,0)
  Row 1: try col=0 → same col → skip
  Row 1: try col=1 → diag conflict (0-0==1-1) → skip
  Row 1: try col=2 → safe → place Q at (1,2)
    Row 2: try col=0 → diag conflict → skip
    Row 2: try col=1 → diag conflict → skip
    Row 2: try col=2 → same col → skip
    Row 2: try col=3 → diag conflict → skip
    Row 2: NO safe column → BACKTRACK
  Row 1: remove (1,2), try col=3 → safe → place Q at (1,3)
    Row 2: try col=0 → safe → place Q at (2,0)
      ... eventually (2,0) leads to dead end → BACKTRACK
    Row 2: try col=1 → safe → place Q at (2,1)
      Row 3: try col=0,1,2,3... col=2 safe!
        Row 4 == N → ADD SOLUTION: .Q../...Q/Q.../..Q. ✓
```

■ Java Code

```
1  class Solution {
2      public List<List<String>> solveNQueens(int n) {
3          List<List<String>> result = new ArrayList<>();
4          char[][] board = new char[n][n];
5          for (char[] row : board) Arrays.fill(row, '.');
6
7          Set<Integer> cols = new HashSet<>();
8          Set<Integer> diag1 = new HashSet<>(); // row - col
9          Set<Integer> diag2 = new HashSet<>(); // row + col
10
11         backtrack(board, 0, n, cols, diag1, diag2, result);
12         return result;
13     }
14
15     private void backtrack(char[][] board, int row, int n,
16         Set<Integer> cols, Set<Integer> diag1, Set<Integer> diag2,
17         List<List<String>> result) {
18         if (row == n) {
19             result.add(construct(board)); // All queens placed!
20             return;
21         }
```

```

21     }
22     for (int col = 0; col < n; col++) {
23         if (cols.contains(col)) continue;
24         if (diag1.contains(row - col)) continue;
25         if (diag2.contains(row + col)) continue;
26
27         // Place queen
28         board[row][col] = 'Q';
29         cols.add(col); diag1.add(row-col); diag2.add(row+col);
30
31         backtrack(board, row+1, n, cols, diag1, diag2, result);
32
33         // Remove queen (backtrack)
34         board[row][col] = '.';
35         cols.remove(col); diag1.remove(row-col); diag2.remove(row+col);
36     }
37 }
38
39 private List<String> construct(char[][] board) {
40     List<String> list = new ArrayList<>();
41     for (char[] row : board) list.add(new String(row));
42     return list;
43 }
44 }

```

■ ■ Time & Space Complexity

Type	Value	What It Means
Time	$O(n!)$	n choices for row 0, n-1 for row 1... = n! paths. Pruning cuts many.
Space	$O(n^2)$	Board = n^2 . Sets = $O(n)$. Stack = $O(n)$

■ ■ Common Mistakes

■ ■ Diagonal Formula

Cells on the same top-left diagonal have the same (row-col) value. Cells on the same top-right diagonal have the same (row+col) value. Mixing these up is the most common N-Queens bug.

■ ■ Not Removing from Sets

If you add to cols/diag1/diag2 but forget to remove after backtracking, future rows incorrectly think those positions are occupied.

■ Pro Tip

Use boolean arrays instead of HashSets: `boolean[] cols = new boolean[n]; boolean[] d1 = new boolean[2*n]; boolean[] d2 = new boolean[2*n]`. Arrays are faster than HashSets for this problem.

■ Practice Problems

#	Problem	Diff
1	N-Queens (LeetCode 51)	Hard
2	N-Queens II — count solutions (LeetCode 52)	Hard
3	Sudoku Solver (LeetCode 37)	Hard
4	Word Search (LeetCode 79)	Medium

■ Quick Revision

✓	One queen per row — try all columns for each row
✓	3 conflict checks: col set, diag1 (row-col), diag2 (row+col)
✓	Place queen → recurse → remove queen (backtrack)
✓	Base case: row == n → record the board
✓	Time $O(n!)$, Space $O(n^2)$
✓	N=4 has 2 solutions, N=8 has 92 solutions

■ Visual Thinking

Imagine an $N \times N$ city grid. You're placing N snipers — each must have a clear line of sight to their zone and CANNOT be seen by any other sniper (no shared row, column, or diagonal). You place the first sniper, then the second must avoid all attack lines of the first. If you hit a dead end, the last-placed sniper moves to their next possible position. This relentless 'try-and-retreat' is backtracking in its purest form.