The backtracking algorithm is implemented in the **SudokuSolver.java** file. The backtracking algorithm is implemented in the following lines of the SudokuSolver.java file:

**Line 16: public static boolean solve(int[][] puzzle) {…**

Entry point for the solver method.

**Line 17: return solve(puzzle, 0, 0);**

The recursive solve method is called, starting from the top-left corner of the puzzle (row 0, column 0).

**Line 32: private static boolean solve(int[][] puzzle, int row, int col) {…**

The private recursive method that handles the backtracking logic.

**Line 39: return solve(puzzle, row + 1, 0);**

If the end of a row is reached, the algorithm proceeds to the next row.

**Line 43: return solve(puzzle, row, col + 1);**

If the current cell is already filled, the algorithm moves to the next column in the same row.

**Line 49: if (solve(puzzle, row, col + 1)) {…**

Recursively attempts to solve the puzzle by trying the next cell after placing a number in the current cell.

**Algorithm Mathematical Expression**

Given a 9×9 Sudoku grid 𝑃 where some cells are pre-filled with numbers (from 1 to 9) and others are empty (denoted as 0), the goal is to fill in all the empty cells such that each row, each column, and each 3×3 sub-grid contains all the numbers from 1 to 9 without repetition.

**Variables and Functions**

**Puzzle Grid:** 𝑃 is a 9×9 grid where 𝑃[𝑖][𝑗] represents the value at row 𝑖 and column 𝑗.

**Row and Column:** The algorithm iterates over the grid using row and col indices.

**Valid Placement Function:** isValid(𝑃,𝑖,𝑗,𝑣) checks if placing a value 𝑣 at 𝑃[𝑖][𝑗] follows Sudoku rules.

**Backtracking Algorithm**

The backtracking algorithm can be expressed mathematically as follows:

**Base Case:**

**Recursive Step:**

**Validation Function:** The isValid function ensures that placing a value 𝑣 at 𝑃[𝑖][𝑗] is consistent with Sudoku rules: