Sudoku Solver

Report

By:

Divyanshi Tyagi

[20240010030010]

Course:

B. Tech - Computer Science and Engineering with Artificial Intelligence

---

Institution:

KIET Group of Institutions

Date:

11 March 2025

Introduction:

Sudoku is a popular and challenging puzzle game that has been widely recognized for its potential to enhance cognitive skills and problem-solving abilities. The objective of Sudoku is to fill a 9x9 grid with digits from 1 to 9 in such a way that each column, each row, and each of the nine 3x3 sub grids, also known as regions or blocks, contain all of the digits from 1 to 9 exactly once.

The puzzle starts with some cells already filled with digits, and the remaining cells need to be filled according to the Sudoku rules. While solving the puzzle manually can be a fun and engaging activity, it becomes increasingly difficult as the level of complexity increases. As such, the need for an automated method to solve Sudoku puzzles efficiently has led to the development of various algorithms.

Methodology

The Sudoku Solver uses the **backtracking algorithm**, which is a recursive approach to solve constraint satisfaction problems like Sudoku.

Steps:

1. **Representation**: The Sudoku puzzle is represented as a 9x9 grid (2D list), where empty cells are denoted by 0, and filled cells contain numbers from 1 to 9.
2. **Backtracking Process**:
    - Start at the first empty cell (represented by 0).
    - For each empty cell, attempt to place digits from 1 to 9.
    - For each number, check if it satisfies Sudoku constraints:
        - The number must not repeat in the current row, column, or 3x3 subgrid.
    - If a valid number is found, move to the next empty cell.
    - If no valid number can be placed, backtrack by resetting the cell to 0 and trying another number in the previous cells.
3. **Constraint Checking**:
    - Ensure the number placed is valid by checking:
        - **Row**: No duplicates in the same row.
        - **Column**: No duplicates in the same column.
        - **Subgrid**: No duplicates in the 3x3 subgrid.
4. **Termination**:
    - The process continues until the puzzle is solved or all possibilities are exhausted.
    - If the puzzle is solved, the grid is printed; otherwise, the algorithm reports that no solution exists.

Efficiency:

Backtracking efficiently narrows down possibilities by pruning invalid choices early, making it suitable for solving standard Sudoku p

CODE

```
def is_valid(board, row, col, num):
    # Check if the number is not repeated in the current row, column, or 3x3 grid
    for i in range(9):
        if board[row][i] == num or board[i][col] == num:
            return False

    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(start_row, start_row + 3):
        for j in range(start_col, start_col + 3):
            if board[i][j] == num:
                return False

    return True

def solve_sudoku(board):
    # Try to find an empty space in the board
    for row in range(9):
        for col in range(9):
            if board[row][col] == 0:  # 0 represents an empty space
                for num in range(1, 10):  # Try numbers from 1 to 9
                    if is_valid(board, row, col, num):
                        board[row][col] = num  # Place the number
                        if solve_sudoku(board):  # Recursively try to solve
                            return True
                        board[row][col] = 0  # Backtrack if not valid
                return False  # If no number fits, return False
    return True  # Puzzle solved
```

```
def print_board(board):
    for row in board:
        print(" ".join(str(num) for num in row))


def get_input():
    board = []
    print("Enter the Sudoku puzzle row by row (use 0 for empty cells):")
    for i in range(9):
        while True:
            try:
                row = list(map(int, input(f"Row {i + 1}: ").strip().split()))
                if len(row) != 9:
                    print("Each row must contain exactly 9 numbers.")
                else:
                    board.append(row)
                    break
            except ValueError:
                print("Please enter valid integers.")
    return board

# Get user input for the Sudoku board
board = get_input()


if solve_sudoku(board):
    print("\nSudoku solved successfully!")
    print_board(board)
else:
    print("\nNo solution exists")
```

Output

```
Enter the Sudoku puzzle row by row (use 0 for empty cells):
Row 1: 5 3 0 0 7 0 0 0 0
Row 2: 6 0 0 1 9 5 0 0 0
Row 3: 0 9 8 0 0 0 0 6 0
Row 4: 8 0 0 0 6 0 0 0 3
Row 5: 4 0 0 8 0 3 0 0 1
Row 6: 7 0 0 0 2 0 0 0 6
Row 7: 0 6 0 0 0 0 2 8 0
Row 8: 0 0 0 4 1 9 0 0 5
Row 9: 0 0 0 0 8 0 0 7 9

Sudoku solved successfully!
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```