SUDOKU-SOLVER

NAME- ABDUL RAFEY

CLASS- CSE(AIML)-A

U.NO- 202401100400003

Introduction:

Sudoku is a popular number puzzle that requires filling a 9x9 grid with numbers such that each row, each column, and each of the nine 3x3 sub grids contain all digits from 1 to 9 without repetition. Solving a Sudoku puzzle programmatically involves using algorithms to determine the correct placement of numbers in the empty cells.

Methodology:

The solution follows a backtracking approach, which systematically fills the empty cells while ensuring that Sudoku rules are followed.

Algorithm Steps:

1. Identify an empty cell in the Sudoku grid.
2. Try placing numbers from 1 to 9 in the empty cell.
3. Check if the number placement is valid (i.e., it does not violate row, column, or sub grid constraints).
4. If valid, place the number and move to the next empty cell.
5. If an invalid placement is encountered, backtrack to the previous cell and try another number.
6. Repeat the process until the entire board is filled correctly.

Implementation The Sudoku solver is implemented in Python using the following functions:

- **is_valid(board, row, col, num):** Checks whether placing a given number at a specific position is valid.
- **solve_sudoku(board):** Implements the backtracking algorithm to solve the Sudoku puzzle.
- **print_board(board):** Prints the Sudoku board in a readable format.

CODE:

```
def is_valid(board, row, col, num):
    """Check if it's valid to place num in board[row][col]"""
    for i in range(9):
        # Check if num exists in the same row or column
        if board[row][i] == num or board[i][col] == num:
            return False

    # Check if num exists in the 3x3 subgrid
    start_row, start_col = 3 * (row // 3), 3 * (col // 3)
    for i in range(3):
        for j in range(3):
            if board[start_row + i][start_col + j] == num:
                return False

    return True



def solve_sudoku(board):
    """Solve the Sudoku puzzle using backtracking"""
    for row in range(9):
```

```
        for col in range(9):
            # Find an empty cell
            if board[row][col] == 0:
                for num in range(1, 10):  # Try numbers 1-9
                    if is_valid(board, row, col, num):
                        board[row][col] = num  # Place the number
                        if solve_sudoku(board):  # Recur to solve rest
of the board
                            return True
                        board[row][col] = 0  # Backtrack if not solvable
                return False
# No valid number found, trigger backtracking
    return True
 # Puzzle solved
```

```
def print_board(board):
    """Print the Sudoku board in a readable format"""
    for row in board:
        print(" ".join(str(num) if num != 0 else '.' for num in
row))

# Example Sudoku puzzle (0 represents empty cells)
sudoku_board = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
...     print("Solved Sudoku:")
    print_board(sudoku_board)
else:
    print("No solution exists")
```

OUTPUT:

```
Solved Sudoku:
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
```