ANUSHKA GAUR

202401100400053
ROLL NO-4
SUBJECT --AI
MSE 2

I. Introduction

Sudoku is a logic-based number placement puzzle where the objective is to fill a 9x9 grid so that each row, each column, and each 3x3 subgrid contains all digits from 1 to 9 without repetition. Solving Sudoku puzzles manually can be time-consuming, and thus, an automated solver using Python can significantly enhance efficiency. This report presents a Python-based Sudoku solver that employs the backtracking algorithm to solve puzzles efficiently.

c. Methodology

Methodology

Approach Used:

1. Input Handling: The Sudoku puzzle is represented as a 9x9 matrix, with empty cells denoted by 0.
2. Backtracking Algorithm:
    - Find an empty cell.
    - Try placing numbers 1 to 9 sequentially.
    - Check if placing a number violates Sudoku constraints.
    - If a valid number is found, proceed to the next empty cell.
    - If no valid number is found, backtrack to the previous step and try a different number.
3. Output Display: The completed Sudoku grid is displayed once a solution is found.

REFERENCE

- Python official documentation: https://docs.python.org/3/
- Backtracking algorithm concepts from various online resources.
- Sample Sudoku puzzle sourced from open datasets.

GitHub Repository

Ensure that the following files are uploaded to the GitHub repository:

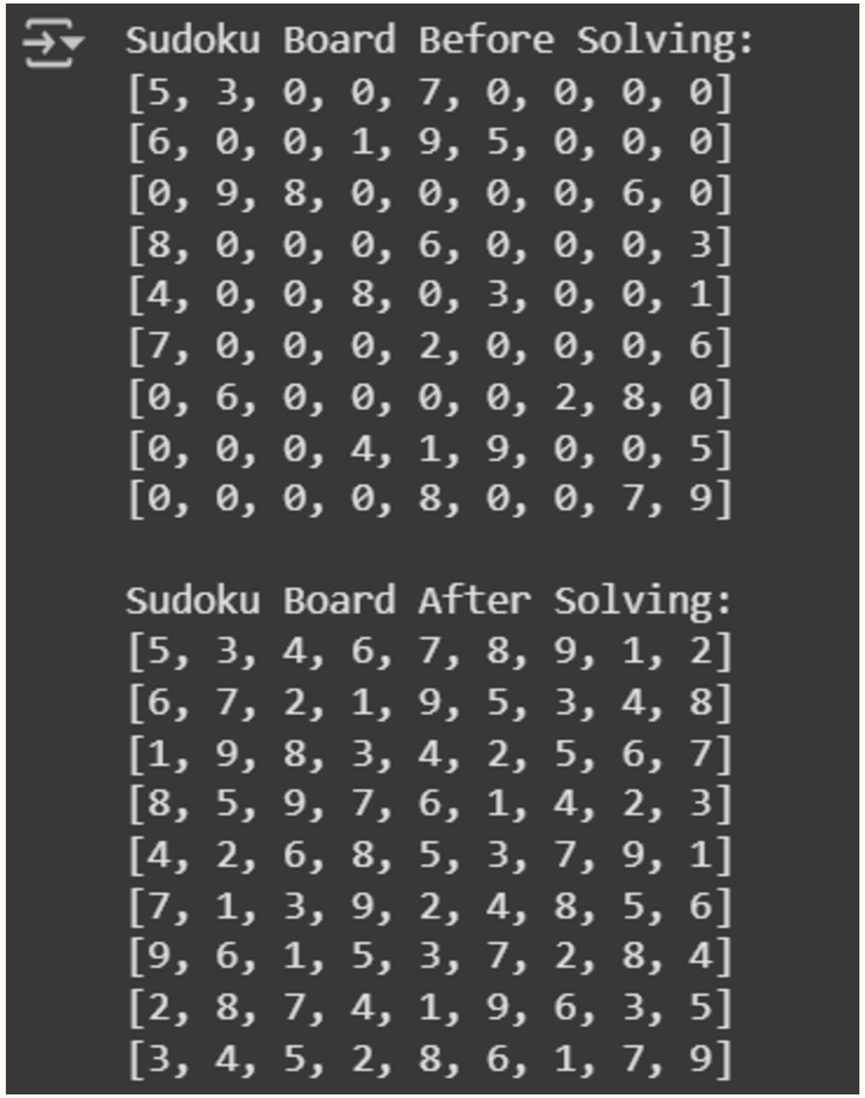
- Sudoku_Solver.ipynb (Jupyter Notebook containing the code)
- Report.pdf (This report in PDF format)
- README.md (Instructions on how to use the Sudoku solver)

```
def is_valid(board, row, col, num):
    # Check if the number is not already in the current row
    for i in range(9):
        if board[row][i] == num: # If the number is found in the row, return False
            return False
    # Check if the number is not already in the current column
    if board[i][col] == num: # If the number is found in the column, return False
        return False
    # Check if the number is not already in the 3x3 subgrid
    if board[3 * (row // 3) + i // 3][3 * (col // 3) + i % 3] == num:
        return False
    return True # Return True if the number is valid for the cell

# Function to solve the Sudoku puzzle using backtracking
def solve_sudoku(board):
    # Iterate over each cell in the grid
    for row in range(9):
        for col in range(9):
            if board[row][col] == 0: # Check if the cell is empty
                # Try all numbers from 1 to 9
```

Example Sudoku puzzle with 0 representing empty cells
sudoku_board = [
[5, 3, 0, 0, 7, 0, 0, 0, 0], # Row 1
[6, 0, 0, 1, 9, 5, 0, 0, 0], # Row 2
[0, 9, 8, 0, 0, 0, 0, 6, 0], # Row 3
[8, 0, 0, 0, 6, 0, 0, 0, 3], # Row 4
[4, 0, 0, 8, 0, 3, 0, 0, 1], # Row 5
[7, 0, 0, 0, 2, 0, 0, 0, 6], # Row 6
[0, 6, 0, 0, 0, 0, 2, 8, 0], # Row 7
[0, 0, 0, 4, 1, 9, 0, 0, 5], # Row 8
[0, 0, 0, 0, 8, 0, 0, 7, 9] # Row 9
]

Solve the puzzle and print the result
if solve_sudoku(sudoku_board): # Check if a solution exists
print("Solved Sudoku board:") # Print success message
print_board(sudoku_board) # Display the solved board
else:
print("No solution exists.") # Print failure message if no solution is found


Sudoku Board Before Solving:
[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]
[8, 0, 0, 0, 6, 0, 0, 0, 3]
[4, 0, 0, 8, 0, 3, 0, 0, 1]
[7, 0, 0, 0, 2, 0, 0, 0, 6]
[0, 6, 0, 0, 0, 0, 2, 8, 0]
[0, 0, 0, 4, 1, 9, 0, 0, 5]
[0, 0, 0, 0, 8, 0, 0, 7, 9]
Sudoku Board After Solving:
[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]

OUTPUT