TITLE PAGE

SUDOKO SOLVER

PROBLEM STATEMENT

"DEVELOP A PYTHON-BASED SUDOKU SOLVER THAT CAN HANDLE A GENERALIZED N×N SUDOKU GRID USING A BACKTRACKING ALGORITHM. THE PROGRAM SHOULD EFFICIENTLY FILL THE MISSING NUMBERS WHILE ENSURING THE SOLUTION ADHERES TO SUDOKU RULES. ADDITIONALLY, VISUALIZE THE SUDOKU GRID BEFORE AND AFTER SOLVING USING MATPLOTLIB."

NAME : AMAN UPADHYAY

UNIVERSITY ROLL NO:- 202401100300032

INTRODUCTION

INPUT: A PARTIALLY FILLED N×N SUDOKU GRID (E.G., 9×9, 16×16).

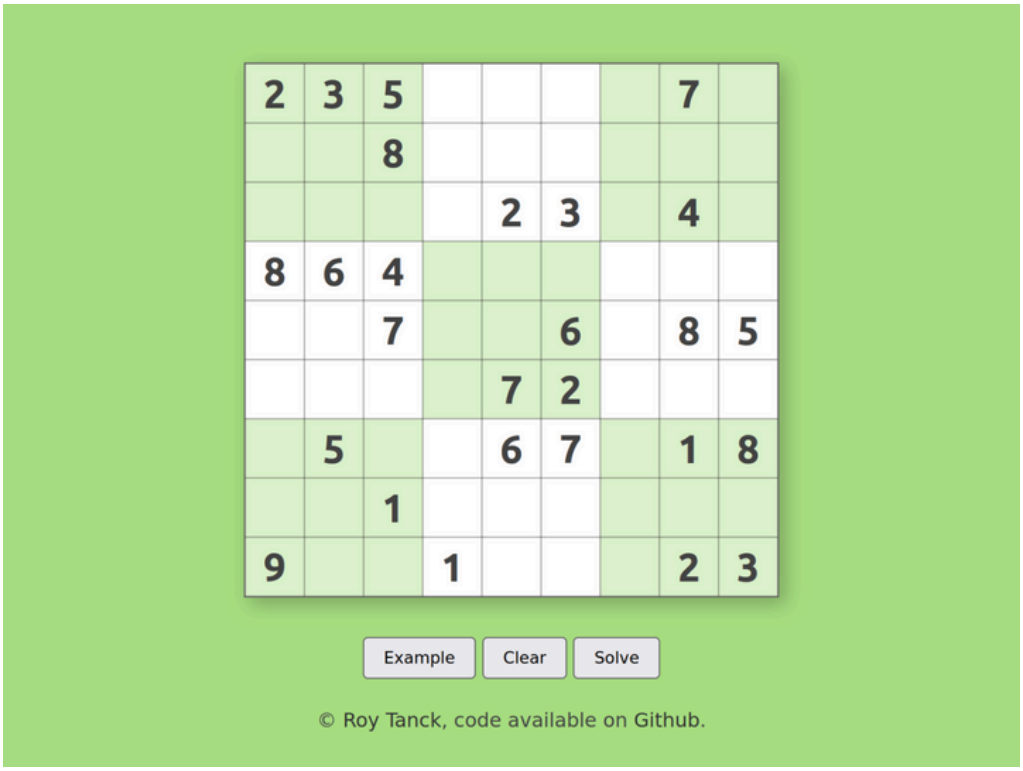
OUTPUT: A COMPLETELY SOLVED SUDOKU GRID FOLLOWING STANDARD RULES.

CONSTRAINTS: EACH ROW CONTAINS UNIQUE NUMBERS (1 TO N).

EACH COLUMN CONTAINS UNIQUE NUMBERS (1 TO N).
EACH $\sqrt{N} \times \sqrt{N}$ SUBGRID CONTAINS UNIQUE NUMBERS (FOR PERFECT SQUARES LIKE 9, 16, ETC.).

METHOD USED: BACKTRACKING ALGORITHM.

Visualization: The Sudoku grid is plotted using Matplotlib before and after solving.



METHODOLOGY

1. Understanding the Problem

A Sudoku puzzle consists of an N×N grid, where some cells are pre-filled with numbers.

The goal is to fill the empty cells while following these constraints

- Each row must contain unique numbers from 1 to N.
- Each column must contain unique numbers from 1 to N.
- Each √N × √N subgrid (for perfect squares like 9×9, 16×16) must contain unique numbers.

2. Approach Used: Backtracking Algorithm

Steps Followed in Backtracking:

1. Find an Empty Cell
   - The algorithm scans the grid to locate the next empty cell (a cell with 0).
2. Try Placing a Number (1 to N)
   - The algorithm attempts to place each number from 1 to N in the empty cell.
   - It checks if the number follows Sudoku rules (row, column, and subgrid uniqueness).
3. Check Validity
   - If the number satisfies Sudoku constraints, it is temporarily placed in the cell.
4. Recursively Solve the Remaining Grid
   - The function then moves to the next empty cell and repeats the process.
   - If a valid number is found, it continues filling the board.
5. Backtracking (Undo if Necessary)
   - If no valid number can be placed in a cell, the function backtracks (removes the last placed number) and tries the next possibility.
   - This ensures all possible solutions are explored efficiently.
6. Stop When the Grid is Fully Filled
   - Once all cells are filled correctly, the algorithm stops and returns the solved Sudoku grid.

CODE OF SUDOKO SOLVER

```python
import math
import numpy as np
import matplotlib.pyplot as plt

#importing necessary libraries for the project
```

```python
def is_valid(board, row, col, num, N):
    k = int(math.sqrt(N))
    for i in range(N):
        if board[row][i] == num or board[i][col] == num:
            return False
    start_row, start_col = (row // k) * k, (col // k) * k
    for i in range(k):
        for j in range(k):
            if board[start_row + i][start_col + j] == num:
                return False
    return True
```

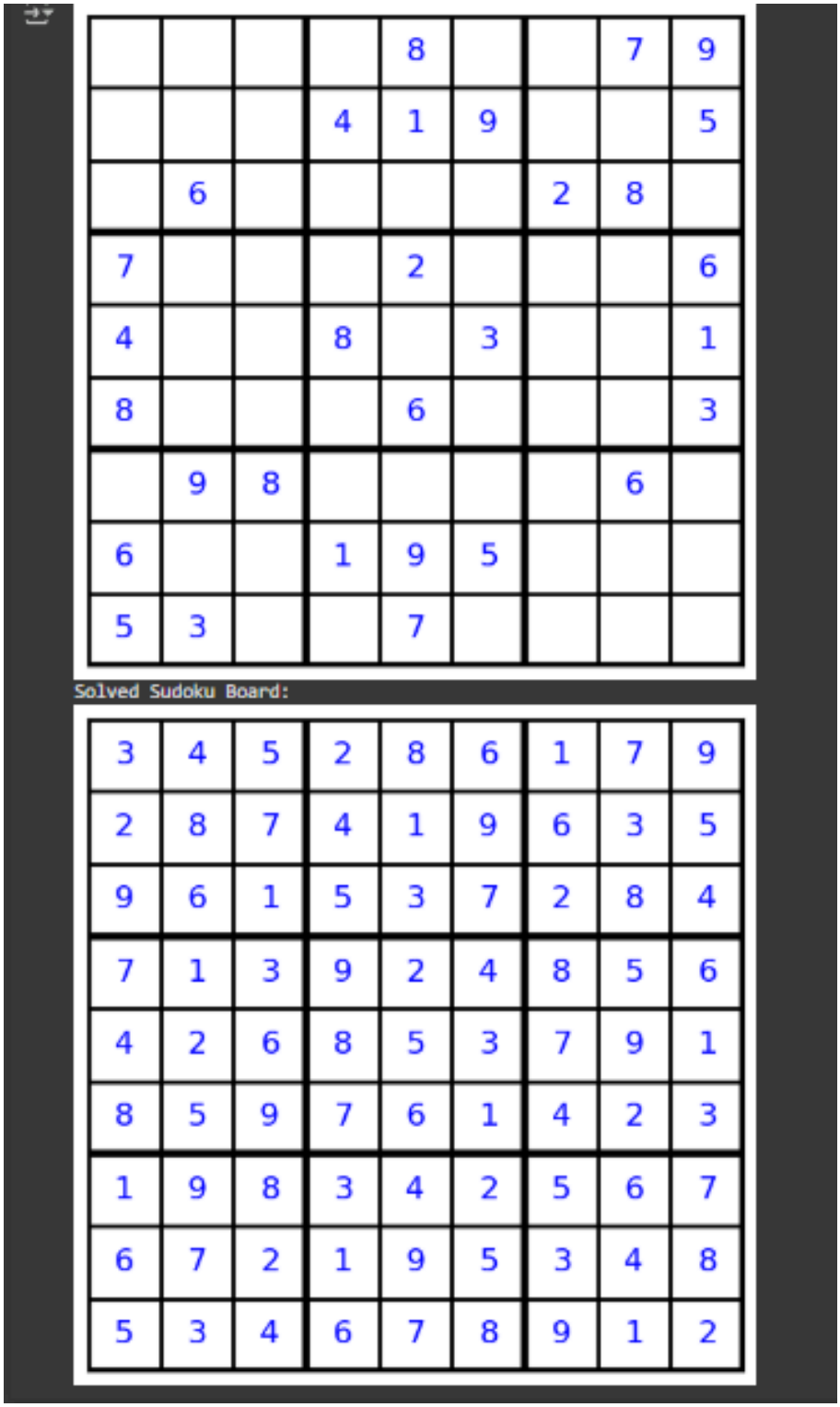
```python
def solve_sudoku(board, N):
    for row in range(N):
        for col in range(N):
            if board[row][col] == 0:
                for num in range(1, N + 1):
                    if is_valid(board, row, col, num, N):
                        board[row][col] = num
                        if solve_sudoku(board, N):
                            return True
                        board[row][col] = 0
                return False
    return True
```

```python
sudoku_board_9x9 = [
    [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]
]
```

```python
N_9 = 9

print("Initial Sudoku Board:")
draw_sudoku(sudoku_board_9x9)

if solve_sudoku(sudoku_board_9x9, N_9):
    print("Solved Sudoku Board:")
    draw_sudoku(sudoku_board_9x9)
else:
    print("No solution exists.")
```

```python
def draw_sudoku(board):
    N = len(board)
    k = int(np.sqrt(N))

    fig, ax = plt.subplots(figsize=(6, 6))
    ax.set_xticks(np.arange(N+1)-0.5, minor=True)
    ax.set_yticks(np.arange(N+1)-0.5, minor=True)

    ax.grid(which="minor", color="black", linestyle='-', linewidth=2)
    ax.tick_params(which="both", bottom=False, left=False, labelbottom=False, labelleft=False)

    for i in range(N):
        for j in range(N):
            if board[i][j] != 0:
                ax.text(j, i, str(board[i][j]), ha='center', va='center', fontsize=16, color='blue')

    for i in range(0, N+1, k):
        ax.axhline(i-0.5, color='black', linewidth=4)
        ax.axvline(i-0.5, color='black', linewidth=4)

    plt.show()
```

OUTPUT OF CODE



				8			7	9
			4	1	9			5
	6					2	8	
7				2				6
4			8		3			1
8				6				3
	9	8					6	
6			1	9	5			
5	3			7				

Solved Sudoku Board:

3	4	5	2	8	6	1	7	9
2	8	7	4	1	9	6	3	5
9	6	1	5	3	7	2	8	4
7	1	3	9	2	4	8	5	6
4	2	6	8	5	3	7	9	1
8	5	9	7	6	1	4	2	3
1	9	8	3	4	2	5	6	7
6	7	2	1	9	5	3	4	8
5	3	4	6	7	8	9	1	2

REFRENCE:

Sudoku Algorithm & Backtracking

GEEKS FOR GEEKS

https://www.geeksforgeeks.org/sudoku-backtracking-7/

LeetCode: "Sudoku Solver Problem"