Report on Suduko solver

- Name -Deepak kumar
- Branch - CSE-(AIML)
- Sec -B
- Roll num - 3
- Course - Artificial intelligence

- institution -KIET group of institution

INTRODUCTION :

Sudoku is a popular logic-based number placement puzzle that requires filling a 9×9 grid with numbers from 1 to 9, ensuring that each row, each column, and each 3×3subgrid (box) contains every number exactly once. The challenge lies in solving the puzzle efficiently while maintaining these constraints.

In this report, we present a Sudoku Solver implemented in Python using the backtracking algorithm. This algorithm is a depth-first search approach that systematically tries placing numbers in empty cells, checks for validity, and backtracks when a conflict is encountered. By recursively exploring possible solutions and undoing incorrect choices.

This report details the implementation of the Sudoku Solver, including:

The structure of the Sudoku board representation.
The working of the backtracking algorithm.
The performance and efficiency of the approach.

The goal of this project is to understand the application of recursion and backtracking in constraint-based problems .

Methodology for Sudoku Solver Report

1. Problem Definition

The objective of this project is to develop a Python-based Sudoku solver that can efficiently fill an incomplete 9×9 Sudoku grid while ensuring that each row, column, and 3×3 subgrid follows the standard Sudoku rules. The program should systematically attempt valid numbers in empty cells and backtrack when conflicts arise.

2. Approach & Algorithm

To solve the Sudoku puzzle, we use the backtracking algorithm, a recursive approach that explores possible solutions while reverting incorrect choices. The key steps of the methodology are as follows:

Step 1: Representing the Sudoku Board

The Sudoku board is represented as a 9×9 matrix (list of lists) in Python.
Empty cells are denoted by 0, while filled cells contain numbers from 1-9.

Step 2: Finding an Empty Cell

The program scans the board row by row, column by column to locate the first empty cell (0).
If no empty cell is found, the puzzle is considered solved.

Step 3: Validating a Number Placement

Before placing a number in an empty cell, the program checks if it is validusing three conditions:
The number must not already exist in the same row.
The number musThe Python implementation consists of the following core functions:
print_board(board) – Displays the Sudoku board.
find_empty(board) – Locates the next empty cell.
is_valid(board, num, pos) – Checks if a number can be placed in a given position.
solve_sudoku(board) – The main recursive function implementing the backtracking algorithm.

Here is my code of sudoko solver:

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

def find_empty(board):
    # Now it will finds the empty spaces
    for i in range(9):
        for j in range(9):
            # We will denote empty spaces by 0
            if board[i][j] == 0:
                return i, j
    # if empty space found it will return row and coulmn

    return None
    # if not found return NONE

def is_valid(board, num, tup):
    #  it will check the placing of num in tup
    row, col = tup

    # Check row and column
    if num in board[row] or num in [board[i][col] for i in range(9)]:
        return False

    # Check 3x3 box
    box_x, box_y = col // 3, row // 3
    for i in range(box_y * 3, box_y * 3 + 3):
        for j in range(box_x * 3, box_x * 3 + 3):
            if board[i][j] == num:
                return False

    return True

def solve_sudoku(board):
    """Solves the Sudoku using backtracking."""
    empty = find_empty(board)
    if not empty:
        return True  # Solved 😃 yey!
```

```
row, col = empty
for num in range(1, 10):  # try number in between 1-9
if is_valid(board, num, (row, col)):
board[row][col] = num  # Place the number

if solve_sudoku(board):
return True  #If solved return True

board[row][col] = 0  # If not solved then go back (recursion)---
# Repeat untill the board is solved

return False

# suduko board
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],
[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]
]

print("Original Sudoku Board is:")
print_board(sudoku_board)

if solve_sudoku(sudoku_board):
print("\nSolved Sudoku Board is :🥰💞")
print_board(sudoku_board)
else:
print("\nNo solution exists.")
```

OUTPUT:

```
Original 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
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

Solved Sudoku Board: 🥹 🇦🇹
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
```

image from google colab :