

Report on Sudoku Solver Using Backtracking

Algorithm

Introduction

Sudoku is a widely recognized logic-based number puzzle, which requires placing digits (1 through 9) in a 9x9 grid. The puzzle has specific rules that each row, each column, and each of the nine 3x3 subgrids (also called boxes) must contain the digits 1 to 9, without repetition. Some of the grid's cells are pre-filled, while others are empty, and the challenge is to correctly fill in these empty cells following the rules.

In this report, we explore a solution to this puzzle using a **backtracking algorithm**. The backtracking approach is a brute-force method that systematically tries all possibilities to find the solution. It works by filling one cell at a time and backtracking if a conflict arises. This method is efficient and guarantees finding the solution for valid Sudoku puzzles.

Code

```
#A function to check if a number is safe to place in a given position
def is_safe(board, row, col, num):
    # Check if the number exists in the same row
    if num in board[row]:
        return False

    # Check if the number exists in the same column
    for r in range(9):
        if board[r][col] == num:
            return False

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

    return True
```

```

# A function to apply the backtracking algorithm
def solve(board):
    for row in range(9):
        for col in range(9):
            # If the current cell is empty (0), try to fill it with a
            valid number
            if board[row][col] == 0:
                for num in range(1, 10):
                    if is_safe(board, row, col, num): # Check if num
is valid
                        board[row][col] = num # Place num in the cell
                        if solve(board): # Recursively attempt to
solve
                                return True
                        board[row][col] = 0 # Undo the move
(backtrack)
                return False # If no valid number is found, backtrack
    return True # Puzzle solved when no empty cells remain

```

```

# Function to print the Sudoku board in a readable format
def print_board(board):
    for row in board:
        print(" ".join(str(num) if num != 0 else '.' for num in
row)) # '.' for empty cells

```

```

#Function to take input from the user for the Sudoku board
def input_board():
    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 and all(0 <= num <= 9 for num in row):
                    board.append(row)
                    break
            else:
                print("Invalid row, please enter exactly 9 numbers
(0-9)!")
        except ValueError:

```

```

        print("Invalid input, please enter only integers!")
    return board

```

```

#Function to take input from the user for the Sudoku board
def input_board():
    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 and all(0 <= num <= 9 for num in row):
                    board.append(row)
                    break
            except ValueError:
                print("Invalid row, please enter exactly 9 numbers
(0-9)!")
                print("Invalid input, please enter only integers!")
    return board

```

Code Output:

The screenshot shows a Google Colab notebook interface. The main area displays the output of the code, which includes the initial Sudoku board, the solved board, and the time taken to complete the execution.

```

Row 2: 6 8 8 1 9 5 8 8 8
Row 3: 0 9 8 0 0 0 6 0
Row 4: 8 0 0 0 6 0 0 3
Row 5: 4 0 0 8 0 3 0 0 1
Row 6: 7 0 0 0 2 0 0 6
Row 7: 0 6 0 0 0 2 8 0
Row 8: 0 0 0 4 1 9 0 0 5
Row 9: 0 0 0 0 0 0 7 9

Initial Sudoku Board:
5 3 . . 7 . . .
6 . . 1 9 5 . . .
. 9 8 . . . 6 .
8 . . 6 . . . 3
4 . . 8 . 3 . . 1
7 . . 2 . . . 6
. 6 . . . 2 8 .
. . . 4 1 9 . . 5
. . . 8 . . 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

```

At the bottom of the notebook, it shows the execution time: 2m 0s, completed at 3:25 PM.

