Sudoku Solver

-Sujith(50),Aswathkrishna(09)
-Sriram(48),Yuvan(60)

Agenda

- Problem Statement
- Algorithm Used
- Modules
- Functions description
- Sample Output

Problem Statement

The problem is to solve a Sudoku puzzle. A Sudoku puzzle is a 9x9 grid divided into 9 subgrids (3x3), where some cells are already filled with numbers from 1 to 9. The objective is to fill in the remaining empty cells such that each row, each column, and each subgrid contains all numbers from 1 to 9 without repetition.

Algorithm used

Graph coloring is a fundamental concept in graph theory that involves assigning colors to the vertices of a graph such that no adjacent vertices share the same color. The objective is to color the graph using the fewest number of colors possible.

The graph coloring algorithm plays a crucial role in solving the Sudoku puzzle by ensuring that each number is placed in a valid position within the board, satisfying the constraints of the game.

Modules used

1. `**matplotlib**`: A plotting library used for visualizing the Sudoku board and the graph representation. It provides flexible and customizable plotting capabilities, making it suitable for displaying the Sudoku puzzle graphically.

2. `**networkx**`: A Python library for the creation, manipulation, and study of the structure, functions of complex networks. It is used in the code to create and handle the graph representation of the Sudoku board, allowing for efficient graph operations such as traversing vertices and edges.

3. `**numpy**`: A powerful numerical computing library used for working with arrays and matrices. It is used in the code to convert the Sudoku board into a NumPy array for easier manipulation and display.

Why those modules?

- `**matplotlib**`: It is a widely used plotting library that offers extensive functionality, allowing for the visualization of the Sudoku board and the graph representation in a visually appealing and informative manner.

- `**networkx**`: It provides comprehensive graph-related functionalities, making it convenient for creating and manipulating the Sudoku board as a graph. Its built-in algorithms and data structures are advantageous for graph-based operations required for solving the puzzle.

- `**numpy**`: It offers efficient numerical operations and array manipulation capabilities, which are useful for converting the Sudoku board into a NumPy array and performing array-based computations, making the code more concise and efficient.

Function Description

1. The code starts by importing necessary libraries: `**time**`, `**copy**`, `**matplotlib**`, `**matplotlib.pyplot**`, `**networkx**`, and `**numpy**`.

2. The `**generate_board**` function generates a Sudoku board with random numbers and empty cells. It uses a pattern to ensure the generated board is a valid solution.

3. The `**gen_sudoku_and_graph**` function generates a Sudoku board and creates a graph representation of the board using the `**make_graph_from_board**` function. It also creates a dictionary `B` to store the colors (numbers) of the vertices (cells) in the graph.

Function Description

4. The `**get_adjacents_from_board**` function takes a Sudoku board, row index (`lin`), and column index (`col`) as input and returns a list of adjacent cell coordinates for the given cell.

5. The `**make_graph_from_board**` function creates a graph `G` from the Sudoku board. Each cell in the board corresponds to a vertex in the graph, and the adjacent cells are connected by edges.

6. The `**get_colored_adjacents_number**` function calculates the number of colored (non-zero) adjacent vertices for each vertex in the graph. It returns a list of tuples containing the number of colored adjacent vertices and the vertex itself.

7. The `**is_ok**` function checks if it's possible to color a vertex with a given color without violating the Sudoku rules. It checks the colors of adjacent vertices and returns `True` if the color is valid and `False` otherwise.

Function Description

8. The `**count_number_of_possibilities**` function calculates the number of possible colors that can be assigned to a vertex by checking the colors of its adjacent vertices.

9. The `**get_vertex_with_least_number_of_possibilities**` function finds the vertex with the fewest number of possible colors to be assigned. It iterates through all vertices and returns the vertex with the least number of possibilities.

10. The **`RESOLUCAO2`** function is an alternative resolution algorithm that solves the Sudoku puzzle using a backtracking approach. It assigns colors to vertices one by one and backtracks when a color assignment leads to an invalid solution. This algorithm uses a heuristic by ordering the vertices based on the number of colored adjacent vertices.

Function Description

11. The `**RESOLUCAO**` function is the main resolution algorithm. It uses a recursive backtracking approach to solve the Sudoku puzzle. It assigns colors to vertices one by one, checks if the assignment is valid, and continues recursively until the puzzle is solved or a contradiction is reached.

12. The `**print_sudoku**` function prints the Sudoku board in a formatted manner.

13. The `**solve**` function is the entry point for solving the Sudoku puzzle. It takes the graph, color dictionary, and an algorithm choice as input and calls the respective resolution algorithm based on the choice.

14. The `**print_su**` function visualizes the solved Sudoku board using the `networkx` library.

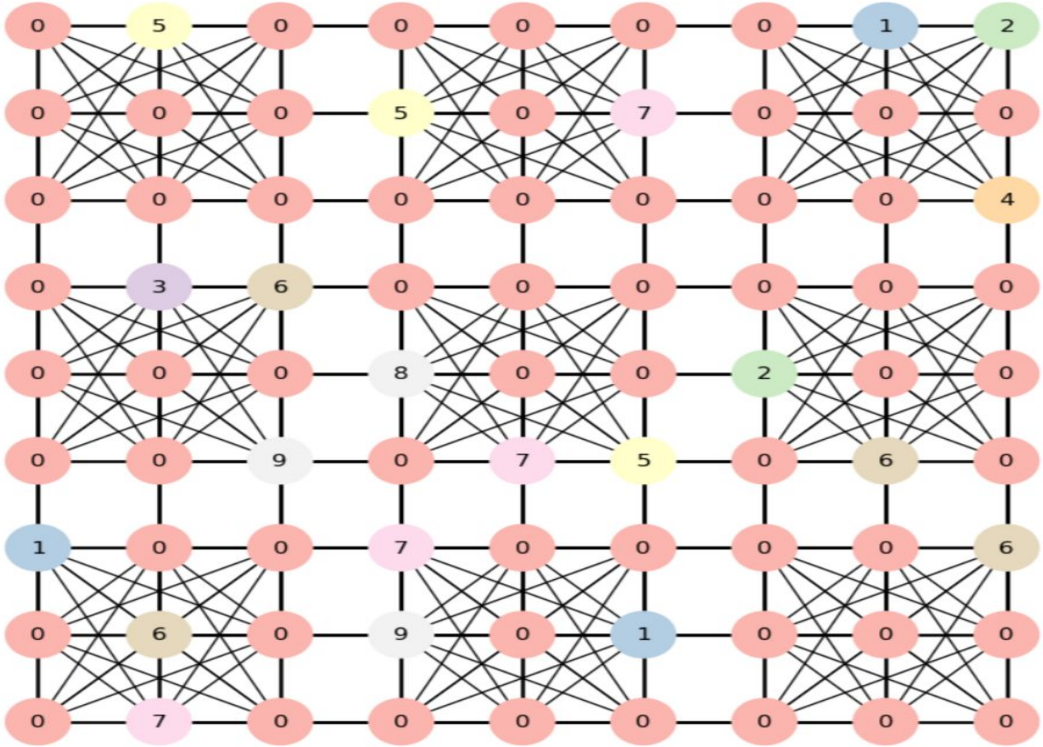
Function Description

15. The code generates a Sudoku board, prints the initial board, solves the puzzle using the `**RESOLUCAO**` algorithm, and prints the solved board.

16. The `**gen_sudoku_and_graph**` function and the resolution algorithms `**RESOLUCAO**` and `**RESOLUCAO2**` can be modified to explore different solving approaches or heuristics.

Output

Raw Board:



Output

Solved Board: