Documentação do Projeto 5: Problemas de Satisfação de Restrições (CSP)

1. Visão Geral do Projeto

Este projeto resolve um problema clássico da ciência da computação: a **Coloração de Mapas**. O objetivo é colorir um mapa político (neste caso, o mapa das Unidades Federativas do Brasil) usando um número limitado de cores, de forma que nenhuma duas regiões adjacentes (estados que fazem fronteira) tenham a mesma cor.

Este é um exemplo canônico de um **Problema de Satisfação de Restrições (CSP)**. O programa utiliza um algoritmo de **Backtracking** para encontrar uma solução válida e, ao final, exibe o grafo do mapa colorido.

2. Como Foi Desenvolvido

O projeto foi desenvolvido em **Python** e utiliza bibliotecas específicas para manipulação e visualização de grafos.

- **Linguagem:** Python.
- **Dados:** Um arquivo uf_neighbors.json que contém a estrutura do mapa. Ele define cada estado e sua lista de vizinhos.
- **Bibliotecas:**
    - json: Para carregar os dados do mapa a partir do arquivo.
    - networkx: A principal biblioteca do projeto. É usada para modelar o mapa como um **grafo**, onde cada estado (UF) é um **nó** (node) e cada fronteira é uma **aresta** (edge).
    - matplotlib.pyplot: Utilizada pelo networkx para desenhar e exibir uma representação visual do grafo final com a coloração aplicada.

3. Algoritmo Utilizado

Problema de Satisfação de Restrições (CSP)

O problema é formalmente definido como um CSP da seguinte maneira:

- **Variáveis:** As 27 Unidades Federativas (os nós do grafo).
- **Domínio:** O conjunto de cores disponíveis para cada variável (neste caso, {0: 'red', 1: 'green', 2: 'blue', 3: 'purple'}).
- **Restrições:** Para qualquer par de estados (U, V) que são vizinhos (conectados por uma aresta), a cor atribuída a U deve ser **diferente** da cor atribuída a V.

Solução com Backtracking

O algoritmo implementado (backtracking_coloring) é uma forma de **busca em profundidade (DFS)** com verificação de restrições. Ele funciona da seguinte forma:

1. **Escolhe um Nó:** Seleciona um estado (nó) para colorir (começando do primeiro).
2. **Tenta uma Cor:** Itera sobre o domínio de cores disponíveis.
3. **Verifica Restrições:** Para uma cor escolhida, verifica todos os vizinhos já coloridos desse nó. Se a cor *não* violar nenhuma restrição (ou seja, for diferente de todos os vizinhos), ela é considerada "segura".
4. **Atribui e Avança:** Se a cor for segura, ela é atribuída ao nó, e o algoritmo avança recursivamente para o próximo nó da lista.
5. **Backtrack (Volta Atrás):** Se a chamada recursiva falhar (o que significa que um nó futuro não pôde ser colorido), ou se nenhuma cor do domínio for "segura" para o nó atual, o algoritmo "volta atrás". Ele remove a atribuição de cor (define como None) e tenta a próxima cor no seu próprio loop.
6. **Sucesso/Falha:** Se o algoritmo conseguir atribuir uma cor a todos os nós, ele retorna True. Se ele tentar todas as combinações e falhar, retorna False.

4. Resultado Visual

A imagem abaixo é um exemplo da saída gerada pelo programa, mostrando o grafo das UFs do Brasil colorido com 4 cores, respeitando as restrições de fronteira.

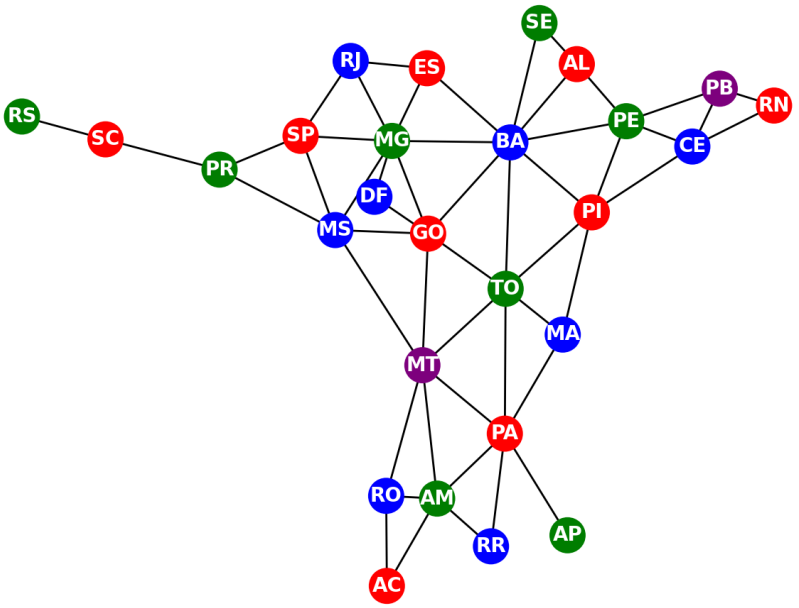

Figura 01: Visão do grafo colorido.