🗺️ Roteiro Conceitual para o Desenvolvimento do Algoritmo DSATUR (Web API - Flask)

Este roteiro é uma adaptação para implementar o algoritmo **DSATUR** dentro de uma **estrutura de aplicação web baseada em Flask**, focada no Item 1 do "Trabalho 3" (Coloração em Grafos).

A aplicação usará a matriz de adjacência do grafo (Figura 1, Tabela 1) como entrada estática para o cálculo do DSATUR e exibirá os resultados em uma interface web, seguindo a estrutura do projeto anexo.

---

1. Visão Geral do Projeto

O objetivo é encapsular a lógica do algoritmo **DSATUR** em uma aplicação web usando o *framework* **Flask**.

1. **Backend (Python/Flask):** Implementar o algoritmo DSATUR e configurar uma rota API para receber requisições e retornar o resultado da coloração.
2. **Frontend (HTML/CSS/JS):** Utilizar o index.html e styles.css fornecidos como template. Será necessário modificar o *frontend* para apresentar a tarefa de coloração (DSATUR) e exibir o resultado.
3. **Entrada de Dados:** O grafo será codificado como uma matriz de adjacência diretamente no *backend* (ou lido de um arquivo, se preferir), conforme a Tabela 1 do documento de trabalho.

---

2. Estrutura do Projeto

A estrutura de diretórios seguirá o modelo do README.md anexo, adaptada para o foco no DSATUR, removendo os algoritmos não solicitados (Prim, Bellman-Ford, Floyd-Warshall) e a pasta /graphs (já que o grafo de entrada é estático).

trabalho_dsatur_web/

├── app.py # Servidor API da aplicação (Flask) [cite: 5]

├── devserver.sh # Script para execução em Linux [cite: 5]

├── devserver.bat # Script para execução em Windows [cite: 5]

├── README.md # Descrição do projeto e instruções de uso [cite: 5]

├── requirements.txt # Lista de dependências do projeto [cite: 5]

|

```
├── /src
|   ├── __init__.py
|   ├── graph_dsatur.py # Módulo para representação do grafo (Matriz Adj.)
|   └── dsatur_algorithm.py # Implementação do Algoritmo de DSATUR
|
├── /static          # Pasta padrão do Flask para arquivos estáticos [cite: 5]
|   ├── /css
|   |   └── styles.css  # Estilos CSS
|   └── /js
|       └── main.js    # Script para interagir com a API do DSATUR
|
└── /templates
    └── index.html     # Página inicial (Interface do DSATUR)
```

---

3. Ferramentas e Tecnologias

- **Linguagem de Programação: Python**.
- **Framework Web: Flask** (para o servidor API).
- **Scripts de Execução:** devserver.bat e devserver.sh para inicialização do ambiente virtual e do servidor Flask na porta 8080.
- **Frontend: HTML5**, **CSS3** (usando styles.css e Bootstrap), e **JavaScript** para comunicação assíncrona com o servidor.

---

4. Roteiro de Implementação

4.1. Configuração do Ambiente e Execução

Os scripts devserver.bat e devserver.sh devem ser usados para:

1. Garantir a existência e ativação do ambiente virtual (venv).
2. Instalar as dependências do requirements.txt (que deve incluir **Flask**).
3. Executar o servidor Flask (python -u -m flask --app app run -p $PORT --debug ou o equivalente em Windows).

4.2. Módulo de Lógica (DSATUR)

A. Representação do Grafo (graph_dsatur.py)

- Definir a **matriz de adjacência** do grafo da Figura 1 (Tabela 1) como uma variável Python (ex: ADJACENCY_MATRIX).
- Implementar funções auxiliares, como:
    - get_neighbors(vertex_index): Retorna a lista de vizinhos de um vértice.
    - calculate_degree(vertex_index): Retorna o grau de um vértice.

B. Algoritmo DSATUR (dsatur_algorithm.py)

- Implementar a função principal dsatur_coloring() que recebe a matriz de adjacência e retorna a coloração final (um dicionário ou lista de tuplas: (vértice, cor)).
- Esta função deve seguir a lógica gulosa descrita no roteiro anterior, com especial atenção à:
    - **Priorização:** Maior Grau de Saturação.
    - **Desempate:** Maior Grau Original.
    - **Busca por cor:** Menor cor positiva disponível.

4.3. Servidor API (app.py)

O servidor Flask será o ponto de integração.

1. **Rota Principal:** Definir a rota / que renderiza o template index.html .
2. **Rota de API:** Criar a rota /run-dsatur (ou similar) que:
    - Chama o módulo graph_dsatur.py para carregar a matriz de adjacência.
    - Chama a função dsatur_coloring() de dsatur_algorithm.py.
    - Formata o resultado da coloração (vértice: cor) em **JSON**.
    - Retorna a resposta JSON (ex: {'coloring': {'1': 1, '2': 2, ...}}).

4.4. Interface Web (index.html e main.js)

O arquivo index.html e o novo script main.js (substituindo form-logic.js e api.js por simplificação) gerenciarão a interface.

- **HTML (index.html):** O formulário original deve ser simplificado para focar apenas no DSATUR, talvez com um único botão "Executar DSATUR" e um

bloco para exibir os resultados. As seções "Seleção de Algoritmo," "Tipo de Grafo," "Escolha o Grafo," e "Vértice Inicial" devem ser removidas ou adaptadas, já que a entrada é fixa.

- **JavaScript (main.js):**
    1. Adicionar um *event listener* ao botão de execução.
    2. Ao clicar, fazer uma requisição fetch (GET ou POST) para a rota /run-dsatur no servidor Flask.
    3. Receber o resultado em JSON.
    4. Manipular o DOM para exibir a coloração final em uma tabela ou lista clara, indicando o número total de cores utilizadas.