**UNIVERSIDADE FEDERAL DE RORAIMA**

**CENTRO DE CIÊNCIA E TECNOLOGIA**

**DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO**

**DCC606– ANALISE DE ALGORITMOS– 2025**

**PROF. DR. HEBERT OLIVEIRA ROCHA**

**CLEILLYSON OSMAR SOUZA DINIZ DE ALMEIDA**

**SARAH EVELYN DO VALE DA SILVA**

**Geração de Cronogramas com Coloração Aproximada de Grafos**

**BOA VISTA, RR**

**2025**

**CLEILLYSON OSMAR SOUZA DINIZ DE ALMEIDA**

**SARAH EVELYN DO VALE DA SILVA**

**Geração de Cronogramas com Coloração Aproximada de Grafos**

Trabalho da disciplina de Análise de Algoritmos do ano de 2025.1 apresentado à Universidade Federal de Roraima do curso de Bacharelado em ciência da computação.

Docente: Prof. Dr. Hebert O. Rocha

**BOA VISTA, RR**

**2025**

LISTA DE ABREVIATURAS E SIGLAS

DSATUR *Degree of saturation*

SUMÁRIO

[1. Introdução 5](#_Toc444681825)

[2. Algoritmos 5](#_Toc444681826)

[2.1. Algoritmo Guloso 5](#_Toc444681827)

[2.1.1. Pseudocódigo 5](#_Toc444681828)

[2.2. Algoritmo Dsatur 6](#_Toc444681829)

[2.2.1. Pseudocódigo 6](#_Toc444681830)

[3. Avaliação Experimental 8](#_Toc444681831)

# Introdução

Coloração de grafos é uma técnica que consiste em aplicar cores aos vértices seguindo a regra de que dois vértices ligados por uma aresta não podem ter a mesma cor, a tarefa é usar a menor quantidade de cores, coloração de grafos pode ser usada em várias áreas mas o trabalho vai focar em resolver os problemas de conflitos em gerações de cronogramas.

# Algoritmos

Foram implementados 2 algoritmos, um algoritmo guloso e um algoritmo DSatur os dois tem a mesma entrada. A primeira entrada é N que é o numero de vertices, as proximas entradas são U e V e são as arestas.

# Algoritmo Guloso

Seu funcionamento é simples ele apenas checa a menor cor disponível, colore o vértice com essa cor e passa pro próximo, por percorrer ordenadamente o os vértices seus resultados são dependentes da ordem de inseção ele não vai garantir uma solução ótima mas é eficiente.

Sua complexidade é O(V2 + E) V é a quantidade de vértices e E é a quantidade de arestas.

# Pseudocódigo

ALGORITMO ColoracaoGulosa(G: Grafo)

ENTRADA: G - grafo não direcionado com V vértices

SAÍDA: resultado[] - array com a cor de cada vértice

INÍCIO

// Inicialização

resultado ← array de tamanho G.numVertices inicializado com -1

disponivel ← array de tamanho G.numVertices de booleanos

// Colorir o primeiro vértice com cor 0

resultado[0] ← 0

// Colorir os vértices restantes

PARA u ← 1 ATÉ G.numVertices-1 FAÇA

// Marcar todas as cores como disponíveis

PARA i ← 0 ATÉ G.numVertices-1 FAÇA

disponivel[i] ← VERDADEIRO

FIM PARA

// Marcar cores dos vizinhos como indisponíveis

atual ← G.listaAdjacencia[u]

ENQUANTO atual ≠ NULO FAÇA

vizinho ← atual.vertice

SE resultado[vizinho] ≠ -1 ENTÃO

disponivel[resultado[vizinho]] ← FALSO

FIM SE

atual ← atual.proximo

FIM ENQUANTO

// Encontrar primeira cor disponível

cor ← 0

ENQUANTO cor < G.numVertices E disponivel[cor] = FALSO FAÇA

cor ← cor + 1

FIM ENQUANTO

// Atribuir a cor ao vértice

resultado[u] ← cor

FIM PARA

RETORNAR resultado

FIM

# Algoritmo Dsatur

Esse algoritmo também tem uma abordagem guloso mas é considerado um pouco mais sofistica pois além de usar apenas o grau ou a posição do vértice, também usa o conceito de grau de saturação. O grau de saturação é definido como o numero de cores diferentes atribuídas aos seus vizinhos.

O primeiro vértice que ele escolhe é o de maior grau e atribui a ele a primeira cor disponível. Depois vai buscar sempre o que tiver o maior grau de saturação e usa o grau do vértice como critério de desempate. E já que é um algoritmo guloso ele sempre vai atribuir a menor cor disponível.

# Pseudocódigo

ALGORITMO DSatur\_Coloracao\_Grafo(G)

ENTRADA: Grafo G = (V, E) com n vértices

SAÍDA: Coloração válida do grafo usando o menor número de cores possível

INÍCIO

// Inicialização

cores[1..n] ← -1 // Nenhum vértice colorido inicialmente

colorido[1..n] ← FALSO // Marca vértices já coloridos

num\_cores ← 0 // Contador de cores utilizadas

// PASSO 1: Escolher vértice inicial (maior grau)

v\_inicial ← vértice com maior grau em G

cores[v\_inicial] ← 0

colorido[v\_inicial] ← VERDADEIRO

num\_cores ← 1

vertices\_restantes ← n - 1

// PASSO 2: Colorir vértices restantes usando critério DSatur

ENQUANTO vertices\_restantes > 0 FAÇA

// Encontrar próximo vértice pelo critério DSatur

max\_saturacao ← -1

max\_grau ← -1

proximo\_vertice ← -1

PARA CADA v ∈ V FAÇA

SE colorido[v] = FALSO ENTÃO

sat ← Calcular\_Grau\_Saturacao(v, G, cores)

grau ← Calcular\_Grau(v, G)

SE (sat > max\_saturacao) OU

(sat = max\_saturacao E grau > max\_grau) ENTÃO

max\_saturacao ← sat

max\_grau ← grau

proximo\_vertice ← v

FIM SE

FIM SE

FIM PARA

// Encontrar menor cor disponível para o vértice escolhido

cores\_usadas[0..n] ← FALSO

PARA CADA u adjacente a proximo\_vertice FAÇA

SE cores[u] ≠ -1 ENTÃO

cores\_usadas[cores[u]] ← VERDADEIRO

FIM SE

FIM PARA

cor\_escolhida ← 0

ENQUANTO cores\_usadas[cor\_escolhida] = VERDADEIRO FAÇA

cor\_escolhida ← cor\_escolhida + 1

FIM ENQUANTO

// Colorir o vértice

cores[proximo\_vertice] ← cor\_escolhida

colorido[proximo\_vertice] ← VERDADEIRO

vertices\_restantes ← vertices\_restantes - 1

SE cor\_escolhida ≥ num\_cores ENTÃO

num\_cores ← cor\_escolhida + 1

FIM SE

FIM ENQUANTO

RETORNAR (cores, num\_cores)

FIM

FUNÇÃO Calcular\_Grau\_Saturacao(v, G, cores)

ENTRADA: Vértice v, Grafo G, Array de cores

SAÍDA: Grau de saturação do vértice v

INÍCIO

cores\_distintas ← conjunto vazio

PARA CADA u adjacente a v FAÇA

SE cores[u] ≠ -1 ENTÃO

cores\_distintas ← cores\_distintas ∪ {cores[u]}

FIM SE

FIM PARA

RETORNAR |cores\_distintas|

FIM

FUNÇÃO Calcular\_Grau(v, G)

ENTRADA: Vértice v, Grafo G

SAÍDA: Grau do vértice v

INÍCIO

grau ← 0

PARA CADA u ∈ V FAÇA

SE (v,u) ∈ E ENTÃO

grau ← grau + 1

FIM SE

FIM PARA

RETORNAR grau

FIM

# Avaliação Experimental

Foi usado como teste o a grade curricular do semestre de 2025.2 do curso de ciências da computação. Foram 16 matérias obrigatárias ofertadas no semestre, modelamos e colocamos para os algoritmos calcularem os horários. Ambos os algorítimos chegaram a 5 cores, mas o algoritmo guloso conseguiu agrupar melhor os resultados.

# CONCLUSÃO

Os dois algoritmos não entregam a opção ótima mas ao menos o DSatur consegue entregar constantemente uma resposta aproximada de ótimo. A principal limitação é a entrada de dados, pois é preciso que as arestas sejam feitas previamente. Isso dificulta para usuários comuns.

**REFERÊNCIAS**

FRANKNBERGER, F. F.; BRANDL, M.; LEITE, M. ESTUDO SOBRE POSSÍVEIS SOLUÇÕES PARA O PROBLEMA DA COLORAÇÃO DE GRAFOS. Anais da Feira do Conhecimento Tecnológico e Científico , [S. l.], v. 1, n. 25, 2024. Disponível em: https://publicacoes.ifc.edu.br/index.php/fetec/article/view/6165

Karger, D., Motwani, R., Sudan, M. (1998). Approximate Graph Coloring by Semidefinite Programming.