BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE UM SISTEMA WEB PARA VISUALIZAÇÃO DE COLORAÇÃO DE ARESTAS EM GRAFOS DE CAYLEY $H_{l,p}$

GABRIEL MEDEIROS ALVES

Rio Verde, GO

2025

INSTITUTO FEDERAL GOIANO - CAMPUS RIO VERDE
BACHAREL EM CIÊNCIA DA COMPUTAÇÃO

IMPLEMENTAÇÃO DE UM SISTEMA WEB PARA VISUALIZAÇÃO DE COLORAÇÃO DE ARESTAS EM GRAFOS DE CAYLEY $H_{l,p}$

GABRIEL MEDEIROS ALVES

Trabalho de Conclusão de Curso apresentado ao Instituto Federal Goiano - Campus Rio Verde, como requisito parcial para a obtenção do Grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. André Da Cunha Ribeiro

Rio Verde, GO

Setembro, 2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema Integrado de Bibliotecas do IF Goiano - SIBi

Alves, Gabriel
A474 Implementação de um sistema web para visualização de coloração de arestas em grafos de Cayley Hlp / Gabriel Alves. Rio Verde 2025.

51f. il.

Orientador: Prof. Dr. André da Cunha Ribeiro.
Tcc (Bacharel) - Instituto Federal Goiano, curso de 0219201 - Bacharelado em Ciência da Computação - Integral - Rio Verde (Campus Rio Verde).
1. Coloração. 2. Arestas. 3. Grafos de Cayley. 4. Grafo Hlp. I. Título.

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO

PARA DISPONIBILIZAR PRODUÇÕES TÉCNICO-CIENTÍFICAS NO REPOSITÓRIO INSTITUCIONAL DO IF GOIANO

Com base no disposto na Lei Federal nº 9.610, de 19 de fevereiro de 1998, AUTORIZO o Instituto Federal de Educação, Ciência e Tecnologia Goiano a disponibilizar gratuitamente o documento em formato digital no Repositório Institucional do IF Goiano (RIIF Goiano), sem ressarcimento de direitos autorais, conforme permissão assinada abaixo, para fins de leitura, download e impressão, a título de divulgação da produção técnico-científica no IF Goiano.

IDENTIFICAÇÃO DA PRODUÇÃO TÉCNICO-CIENTÍFICA

- ☐ Tese (doutorado)
- ☐ Dissertação (mestrado)
- ☐ Monografia (especialização)
- ☑ TCC (graduação)
- ☐ Artigo científico
- ☐ Capítulo de livro
- ☐ Livro
- ☐ Trabalho apresentado em evento
- ☐ Produto técnico e educacional - Tipo:

Nome completo do autor:
Gabriel Medeiros Alves

Matrícula:
2017102201910196

Título do trabalho:
Implementação de um sistema web para visualização de coloração de arestas em grafos de Cayley Hlp

RESTRIÇÕES DE ACESSO AO DOCUMENTO

Documento confidencial: ☑ Não ☐ Sim, justifique:

Informe a data que poderá ser disponibilizado no RIIF Goiano: 31 / 10 / 2025

O documento está sujeito a registro de patente? ☐ Sim ☑ Não

O documento pode vir a ser publicado como livro? ☐ Sim ☑ Não

DECLARAÇÃO DE DISTRIBUIÇÃO NÃO-EXCLUSIVA

O(a) referido(a) autor(a) declara:

• Que o documento é seu trabalho original, detém os direitos autorais da produção técnico-científica e não infringe os direitos de qualquer outra pessoa ou entidade;

• Que obteve autorização de quaisquer materiais inclusos no documento do qual não detém os direitos de autoria, para conceder ao Instituto Federal de Educação, Ciência e Tecnologia Goiano os direitos requeridos e que este material cujos direitos autorais são de terceiros, estão claramente identificados e reconhecidos no texto ou conteúdo do documento entregue;

• Que cumpriu quaisquer obrigações exigidas por contrato ou acordo, caso o documento entregue seja baseado em trabalho financiado ou apoiado por outra instituição que não o Instituto Federal de Educação, Ciência e Tecnologia Goiano.

Rio Verde - GO
Local

07 / 10 / 2025
Data

Assinatura do autor e/ou detentor dos direitos autorais

Ciente e de acordo:

Assinatura do(a) orientador(a)

gov.br
Documento assinado digitalmente
ANDRE DA CUNHA RIBEIRO
Data: 07/10/2025 16:52:02-0300
Verifique em https://validar.iti.gov.br

Regulamento de Trabalho de Curso (TC) – IF Goiano - Campus Rio Verde

ANEXO V - ATA DE DEFESA DE TRABALHO DE CURSO

Aos **24** dias do mês de **setembro** de **dois mil e vinte e cinco**, às 15 horas, reuniu-se a Banca Examinadora composta por: Prof. **André da Cunha Ribeiro** (orientador), Prof. **Douglas Cedrim Oliveira** e Prof. **Márcio da Silva Vilela**, para examinar o Trabalho de Curso (TC) intitulado **COLORAÇÃO DE ARESTAS NOS GRAFOS DE CAYLEY HI,p**, de **Gabriel Medeiros Alves**, estudante do curso de **Bacharelado em Ciência da Computação** do IF Goiano – Campus Rio Verde, sob Matrícula nº **2017102201910196**. A palavra foi concedida ao estudante para a apresentação oral do TC, em seguida houve arguição do candidato pelos membros da Banca Examinadora. Após tal etapa, a Banca Examinadora decidiu pela **APROVAÇÃO** do estudante. A banca decidiu a mudança do título do trabalho para: **Implementação de um sistema web para visualização de coloração de arestas em grafos de Cayley HI,p.** Ao final da sessão pública de defesa foi lavrada a presente ata, que, foi assinada pelos membros da Banca Examinadora e Mediador de TC.

Rio Verde, 24 de setembro de 2025.

André da Cunha Ribeiro

Orientador

Douglas Cedrim Oliveira

Membro da Banca Examinadora

Márcio da Silva Vilela

Membro da Banca Examinadora

Douglas Cedrim Oliveira

Mediador de TC

Documento assinado eletronicamente por:

- **Andre da Cunha Ribeiro**, **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 24/09/2025 17:50:16.
- **Douglas Cedrim Oliveira**, **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 24/09/2025 17:53:13.
- **Marcio da Silva Vilela**, **PROFESSOR ENS BASICO TECN TECNOLOGICO** , em 24/09/2025 18:22:39.

Este documento foi emitido pelo SUAP em 23/09/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 747189
Código de Autenticação: 6b8e2052b2

AGRADECIMENTOS

Primeiramente, agradeço a Deus pela saúde, força e perseverança que me acompanharam ao longo desta caminhada acadêmica.

Aos meus pais, pelo amor incondicional, pela educação, incentivo e pelos valores que me transmitiram e que sempre foram a base para minhas conquistas.

À minha esposa, pelo carinho, paciência e compreensão nos momentos de ausência, além do apoio constante que foi fundamental para a realização deste trabalho.

Ao meu orientador, Prof. Dr. André da Cunha Ribeiro, pela orientação dedicada, pela paciência, pelas valiosas sugestões e pelo incentivo ao desenvolvimento acadêmico e profissional.

A todos que, de alguma forma, contribuíram para a concretização deste trabalho, deixo aqui minha mais sincera gratidão.

RESUMO

ALVES, Gabriel. **Implementação de um sistema web para visualização de coloração de arestas em grafos de Cayley $H_{l,p}$.** Setembro, 2025. 42 f. Trabalho de Conclusão de Curso – (Curso de Bacharel em Ciência da Computação), Instituto Federal Goiano - Campus Rio Verde. Rio Verde, GO.

Este trabalho de conclusão de curso aborda o problema da coloração de arestas na família de grafos de Cayley $H_{l,p}$, analisando as estratégias algorítmicas e implementando uma ferramenta de visualização interativa. A abordagem teórica e prática baseia-se na distinção fundamental da paridade do parâmetro p. Para o caso em que p é par, explora-se a propriedade estrutural do grafo que permite sua decomposição em ciclos de comprimento par. Isso viabiliza a aplicação de um algoritmo de coloração por ciclos alternantes que alcança o índice cromático ótimo de Δ cores com uma complexidade de tempo linear, $O(|E|)$. Quando p é ímpar, a presença de ciclos ímpares torna essa abordagem inviável, exigindo o uso do algoritmo de Vizing (implementado como Misra-Gries), que garante uma coloração própria com no máximo $\Delta + 1$ cores, com uma complexidade de $O(|V||E|)$. Para validar e demonstrar esses resultados, foi desenvolvida uma aplicação web em Python, utilizando as bibliotecas Dash, NetworkX e Dash Cytoscape. A ferramenta permite a geração de instâncias do grafo $H_{l,p}$, aplica o algoritmo de coloração correspondente e exibe o resultado visualmente, servindo como um recurso prático para o estudo das propriedades desses grafos.

Palavras-chave: Coloração, Arestas, Grafos de Cayley, Grafo $H_{l,p}$.

ABSTRACT

ALVES, Gabriel. Implementation of a Web System for Visualizing Edge Coloring in Cayley Graphs $H_{l,p}$. Setembro, 2025. 42 f. Trabalho de Conclusão de Curso – Bacharel em Ciência da Computação, Instituto Federal Goiano - Campus Rio Verde. Rio Verde, GO, Setembro, 2025.

This final year project addresses the edge coloring problem in the Cayley graph family $H_{l,p}$, analyzing algorithmic strategies and implementing an interactive visualization tool. The theoretical and practical approach is based on the fundamental distinction of the parity of the parameter p. For the case where p is even, the structural property of the graph that allows its decomposition into even-length cycles is explored. This enables the application of an alternating cycle coloring algorithm that achieves the optimal chromatic index of Δ colors with a linear time complexity of $O(|E|)$. When p is odd, the presence of odd cycles makes this approach unfeasible, requiring the use of the Vizing algorithm (implemented as Misra-Gries), which guarantees a proper coloring with at most $\Delta + 1$ colors, with a complexity of $O(|V||E|)$. To validate and demonstrate these results, a web application was developed in Python, using the Dash, NetworkX, and Dash Cytoscape libraries. The tool allows for the generation of instances of the $H_{l,p}$ graph, applies the corresponding coloring algorithm, and visually displays the result, serving as a practical resource for studying the properties of these graphs.

Keywords: Coloring, Edges, Cayley Graphs, Graph $H_{l,p}$.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

Muitas situações do cotidiano podem ser representadas eficientemente por diagramas formados por um conjunto de elementos, pontos que são interligados por linhas. Estes pontos podem representar pessoas, e as linhas podem representar relações de amizade entre elas; ou esses pontos podem representar os diversos centros de comunicação, com as linhas simbolizando as conexões de comunicação. Nestes diagramas, o aspecto mais importante está na existência ou ausência de uma linha entre dois pontos. Este tipo de modelagem abstrata, que simplifica e generaliza diversas relações reais, dá origem ao conceito de grafo (BONDY; MURTY, 2008).

De acordo com Gross, Yellen e Zhang (2013), um grafo, denotado por $G = (V, E)$, é uma estrutura composta por um par de conjuntos ordenados: o conjunto V, cujos elementos são chamados de vértices, e o conjunto E, cujos elementos são chamados de arestas, que conectam esses vértices. Dentro desta área, a teoria de coloração de grafos se destaca, pois aborda o problema fundamental de separar um conjunto de objetos em classes a partir de regras específicas, consistindo na atribuição de cores aos elementos do grafo, como vértices e arestas, de modo que dois elementos adjacentes não recebam a mesma cor (ALVES et al., 2015).

Este tema é de grande relevância, pois pertence à classe dos problemas NP-difíceis, o que desafia os pesquisadores a desenvolver algoritmos cada vez mais eficientes para lidar com sua complexidade (GAREY; JOHNSON, 1979). Suas aplicações são diversas, entre elas a tradicional coloração de mapas, a otimização de horários e o transporte de produtos químicos (ALVES et al., 2015). Para tipos específicos de grafos, como os grafos de Cayley $H_{l,p}$, a coloração de arestas pode ser abordada com técnicas adaptadas aos parâmetros envolvidos.

Neste trabalho, apresentamos uma aplicação voltada à visualização da coloração de arestas em grafos de Cayley $H_{l,p}$, utilizando duas abordagens distintas. Quando o parâmetro p é par, utilizamos um algoritmo que colore as arestas do grafo com base nos ciclos do grafo, garantindo uma coloração adequada das arestas. Quando o parâmetro p é ímpar, aplicamos o método de Vizing, que impõe limites ao número cromático das arestas, oferecendo uma solução eficiente para a coloração (VIZING, 1968). A visualização dessas técnicas permite uma análise mais profunda da estrutura e das propriedades dos grafos de Cayley, facilitando a compreensão dos padrões de coloração e suas implicações tanto teóricas quanto práticas.

O trabalho está estruturado da seguinte forma: no Capítulo 2, apresentamos uma visão geral da teoria dos grafos, com ênfase no problema da coloração de arestas. Em seguida, abordamos os grafos de Cayley, destacando suas propriedades e estruturas, e exploramos o grafo $H_{l,p}$, analisando suas características específicas e sua importância no

contexto da coloração de arestas.

No Capítulo 3, discutimos a coloração de arestas nos grafos $H_{l,p}$, detalhando os algoritmos utilizados para diferentes valores de p. Este Capítulo está dividido em seções que explicam o Algoritmo PAR, aplicado quando p é par, e o Algoritmo ÍMPAR, para o caso em que p é ímpar.

Por fim, o Capítulo 4 apresenta as conclusões do trabalho, discutindo as contribuições da aplicação desenvolvida e sugerindo direções para estudos futuros.

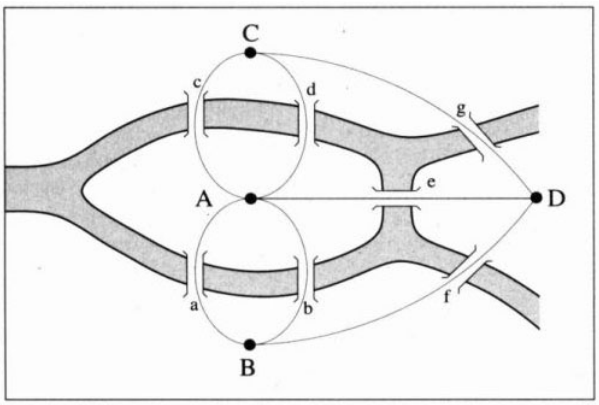
2 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo, apresentamos uma visão geral da teoria dos grafos, destacando conceitos relevantes e, particularmente, o problema de coloração. Em seguida, abordamos os grafos de Cayley, introduzindo suas propriedades e estruturas e, por fim, exploramos o grafo $H_{l,p}$, analisando suas características específicas e a importância no contexto da coloração de arestas.

2.1 Conceitos Fundamentais em Teoria de Grafos

A teoria dos grafos, ramo da matemática e da ciência da computação, dedica-se ao estudo de estruturas compostas por vértices (ou nós) conectados por arestas (ou arcos) e suas diversas aplicações. Essa área foi formalmente introduzida por Leonhard Euler no século XVIII, ao resolver o famoso Problema das Sete Pontes de Königsberg, ilustrado na Figura 1, Euler demonstrou que não era possível percorrer todas as pontes da cidade de Königsberg cruzando cada uma delas exatamente uma vez, que inaugurou o estudo das conexões entre elementos de um conjunto através de grafos (GROSS; YELLEN; ZHANG, 2013). Desde então, o campo expandiu-se significativamente e tem sido aplicado em áreas como redes de comunicação, biologia molecular, logística e redes sociais.

Figura 1 – Pontes de Königsberg.



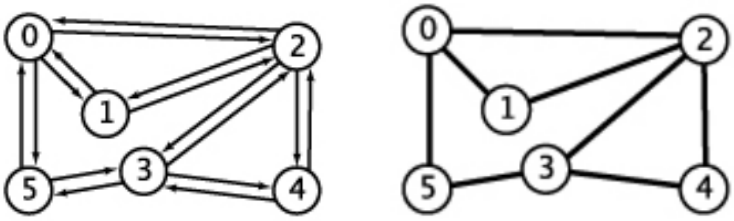
Fonte: Baseado em Dantas (2010).

Um grafo $G = G(V,E)$ é definido por um conjunto V de vértices e um conjunto E de arestas que conectam pares de vértices. Cada vértice em V representa um ponto ou uma entidade, enquanto cada aresta em E indica uma relação ou conexão entre dois vértices. Grafos podem ser direcionados ou não direcionados, dependendo de se as arestas possuem uma orientação específica (ou direção) que liga um vértice a outro. A Figura 2a mostra um exemplo de grafo direcionado, no qual as arestas são pares ordenados de

vértices, enquanto a Figura 2b mostra um grafo não direcionado, em que as arestas são conjuntos não ordenados (WEST, 2001).

Figura 2 – Representação de um Grafo Direcionado e Não Direcionado.

(a) Direcionado. (b) Não Direcionado.



Fonte: Baseado em Feofiloff (2017).

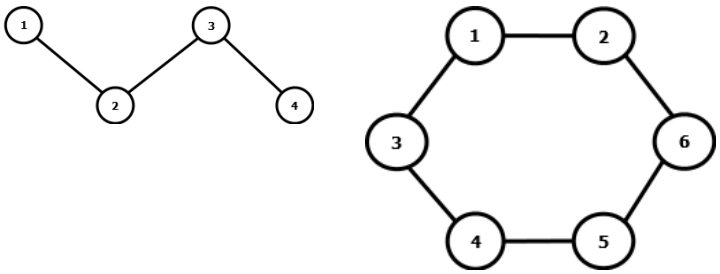
A classificação de grafos em famílias com base em propriedades estruturais compartilhadas é um pilar da teoria dos grafos moderna (DIESTEL, 2017); ela é fundamental para a compreensão da complexidade computacional, visto que muitos problemas intratáveis em grafos gerais tornam-se solucionáveis quando restritos a classes específicas (GAREY; JOHNSON, 1979). Trabalhos em teoria de grafos demonstraram que, ao explorar as características que definem classes como as de grafos perfeitos ou grafos planares, é possível projetar algoritmos altamente eficientes e especializados para problemas que, de outra forma, seriam difíceis (GOLUMBIC, 2004). O estudo dessas classes, extensivamente analisado por Brandstädt, Le e Spinrad (1999), fornece, portanto, a base teórica para a resolução de problemas complexos baseados em redes.

Outro conceito central na teoria dos grafos é o grau de um vértice, que se refere ao número de arestas incidentes a ele. Em um grafo direcionado, distinguimos o grau de entrada e o grau de saída de um vértice, que são, respectivamente, o número de arestas que chegam e saem desse vértice. Essas métricas permitem classificar os vértices e entender a importância relativa de cada um dentro da estrutura do grafo (BONDY; MURTY, 2008). Esse tipo de análise é particularmente útil em redes de comunicação e redes sociais, onde é necessário identificar vértices centrais para otimizar o fluxo de informações.

O conceito de caminho e ciclo também é fundamental na teoria dos grafos. Um caminho é uma sequência de vértices na qual cada vértice é adjacente ao próximo, conforme demonstrado na Figura 3a; enquanto um ciclo é um caminho que começa e termina no mesmo vértice, sem repetir outras arestas ou vértices no percurso, exemplificado na Figura 3b. A análise de caminhos e ciclos é essencial em problemas de otimização e roteamento, como os encontrados em redes de transporte e circuitos eletrônicos (GROSS; YELLEN; ZHANG, 2013). Em muitos casos, a presença de ciclos em grafos está diretamente relacionada à eficiência e robustez das redes.

Figura 3 – Representação de um Grafo Caminho e Ciclo.

(a) Grafo Caminho. (b) Grafo Ciclo.

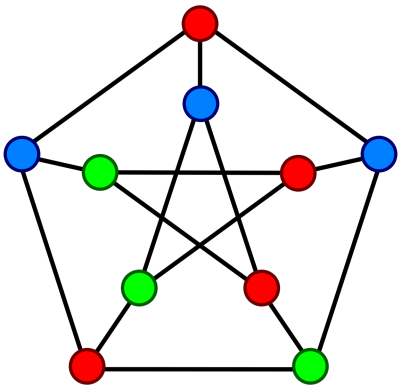


Fonte: Baseado em Dantas (2015).

Entre os tópicos mais estudados, está o conceito de coloração de grafos, que consiste em atribuir cores aos vértices ou arestas de um grafo, de modo que vértices ou arestas adjacentes não compartilhem a mesma cor. A coloração de vértices e de arestas tem aplicações em agendamento, alocação de recursos e problemas de roteamento. A coloração de grafos é conhecida por ser um problema NP-difícil, tornando sua resolução computacionalmente desafiadora para grafos de grande escala (GAREY; JOHNSON, 1979).

A coloração de vértices, por exemplo, busca atribuir cores distintas a vértices vizinhos, evitando conflitos diretos. Um exemplo ilustrativo é mostrado na Figura 4, onde um grafo simples recebe cores nos vértices de modo que nenhum par de vértices adjacentes compartilhe a mesma cor. Este tipo de coloração é aplicado em problemas de alocação de frequências em redes de comunicação e na organização de tabelas de horários, situações em que se deseja minimizar sobreposições.

Figura 4 – Exemplo de Coloração de Vértices.

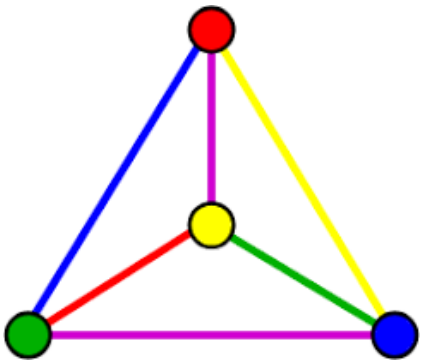


Fonte: Baseado em Wikipedia (2022).

Além das colorações de vértices e arestas, existe a coloração total, que colore simultaneamente vértices e arestas, respeitando as restrições de adjacência. Esse tipo de coloração é fundamental em aplicações que exigem uma diferenciação completa entre os elementos do grafo, como em problemas de design de redes e na minimização de interferências em redes de sensores.

A Figura 5 apresenta um exemplo de coloração total, onde tanto vértices quanto arestas recebem cores, garantindo que nenhum par de elementos adjacentes conflite. Por sua complexidade, a coloração total permanece como uma área ativa de pesquisa, sendo considerada um problema de difícil resolução para grafos gerais (BONDY; MURTY, 2008).

Figura 5 – Exemplo de Coloração Total.



Fonte: Baseado em Almeida (2018).

Nos últimos anos, os avanços na computação têm permitido uma aplicação mais ampla da teoria dos grafos em áreas interdisciplinares. Com o aumento do poder computacional e a criação de algoritmos mais eficientes, problemas complexos de grafos que antes eram impraticáveis podem agora ser resolvidos em tempo hábil, promovendo o desenvolvimento de tecnologias em áreas como inteligência artificial, segurança cibernética e ciência de dados. Ferramentas computacionais de visualização, como a desenvolvida neste trabalho, são especialmente úteis, pois permitem observar as características dos grafos e compreender o comportamento de algoritmos aplicados a problemas específicos (DIESTEL, 2017).

Portanto, a teoria dos grafos oferece um ferramental poderoso para representar e resolver uma ampla variedade de problemas. Sua aplicabilidade prática é amplamente documentada na literatura, e a contínua expansão de novas tecnologias computacionais tende a aumentar ainda mais seu potencial. Neste trabalho, a teoria dos grafos serve como base para o desenvolvimento de uma ferramenta de visualização para coloração de arestas, permitindo estudar algoritmos específicos aplicados a uma classe de grafos conhecida como $H_{l,p}$, que apresentaremos em detalhes na seção subsequente.

2.2 Grafos de Cayley

Os grafos de Cayley, introduzidos por Arthur Cayley em 1878, representam uma ponte fundamental entre a teoria dos grupos e a teoria dos grafos. Eles foram concebidos com o objetivo de associar as estruturas algébricas dos grupos com as estruturas discretas dos grafos, modelando as relações entre os objetos de um determinado conjunto. Em um grafo de Cayley, os vértices estão diretamente associados aos elementos de um grupo,

enquanto as arestas são definidas pelos elementos de um conjunto gerador desse grupo. Esses grafos possuem propriedades intrínsecas que os tornam de grande interesse em diversas áreas da matemática e da computação. São notavelmente conexos, regulares e vértice-transitivos. A sua capacidade de ter um diâmetro logarítmico em relação ao número de vértices torna-os particularmente úteis para a criação de redes de interconexão (RIBEIRO, 2013).

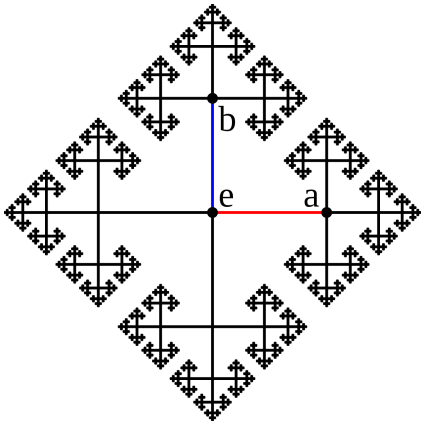
2.2.1 Construção Formal e Propriedades

Um grafo de Cayley, denotado por $\Gamma(G, S)$, é formalmente definido sobre um grupo finito G e um subconjunto de seus geradores $S \subseteq G$. Nesta estrutura, o conjunto de vértices V do grafo é o próprio conjunto de elementos do grupo, de modo que $V = G$. O conjunto de arestas E é estabelecido pela ação do conjunto gerador sobre os elementos do grupo, onde uma aresta direcionada é traçada de um vértice g a um vértice h se, e somente se, $h = gs$ para algum $s \in S$. Assim, o conjunto de arestas é formalmente dado por:

$$E = \{(g, gs) \mid g \in G, s \in S\}$$

A Figura 6 ilustra um exemplo de grafo de Cayley, evidenciando a simetria e a regularidade que caracterizam essa classe de grafos. A topologia do grafo $\Gamma(G, S)$ é intrinsecamente dependente das propriedades do conjunto gerador S. Para a exclusão de laços $-$ e arestas da forma (g, g) $-$, impõe-se a condição de que o elemento identidade do grupo não pertença a S ($\iota \notin S$). Adicionalmente, para que o grafo seja não direcionado, uma condição necessária e suficiente é que S seja simétrico, isto é, fechado sob a operação de inversão ($s \in S \implies s^{-1} \in S$). Essa simetria garante que, para cada aresta (g, gs), sua aresta recíproca (gs, g) também exista em E, visto que $(gs)s^{-1} = g$ (RIBEIRO, 2013).

Figura 6 – Representação do Grafo de Cayley.



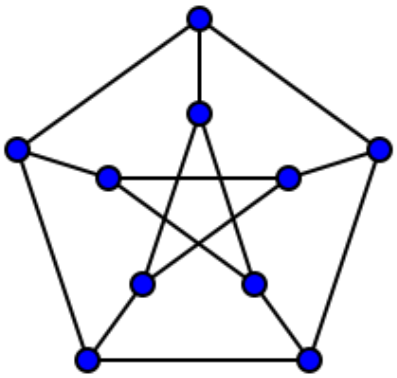
Fonte: Baseado em Cayley (2021).

Grafos de Cayley não direcionados e sem laços exibem um conjunto notável de propriedades estruturais. Primeiramente, a conectividade do grafo é uma consequência

direta da definição de S como um conjunto gerador de G, o que assegura a existência de um caminho entre quaisquer dois vértices. Adicionalmente, trata-se de uma estrutura regular, na qual todo vértice possui grau $k = |S|$; esta uniformidade decorre do fato de que cada elemento $g \in G$ está conectado a um número de vizinhos precisamente igual à cardinalidade do conjunto gerador. Contudo, sua propriedade mais distintiva é a vértice-transitividade, que confere ao grafo uma forte simetria. Isto implica que o grafo é estruturalmente homogêneo: para quaisquer vértices $u, v \in V$, existe um automorfismo ϕ de $\Gamma(G, S)$ tal que $\phi(u) = v$. Essa simetria é uma manifestação direta da ação regular do grupo G sobre si mesmo por multiplicação (RIBEIRO, 2013).

É crucial estabelecer, entretanto, que a recíproca não é verdadeira: nem todo grafo vértice-transitivo é um grafo de Cayley. O exemplo canônico que ilustra essa distinção é o Grafo de Petersen (Figura 7). Embora seja vértice-transitivo, sua estrutura de parâmetros (ordem 10, grau 3) não é compatível com a de nenhum grafo de Cayley que possa ser gerado por um grupo de ordem 10 (RIBEIRO, 2013).

Figura 7 – Representação do Grafo de Petersen.



Fonte: Baseado em Wikipedia (2021).

2.3 Grafo $\mathbf{H_{l,p}}$

O grafo $H_{l,p}$ é um tipo especial de grafo utilizado no estudo de particionamento de arestas. Esse grafo foi inicialmente definido como uma estrutura auxiliar para resolver problemas complexos de coloração e particionamento, como proposto por Holyer (1981). De acordo com Ribeiro (2013):

> "o grafo $H_{l,p}$ é formado por vértices contendo l coordenadas com valores entre 0 e $p-1$, de modo que a soma das l coordenadas seja equivalente a $0 \mod p$, com $p \in \mathbb{Z}^+$. Existe uma aresta entre dois vértices sempre que houver um par de coordenadas correspondentes com valores que diferem por uma unidade".

,

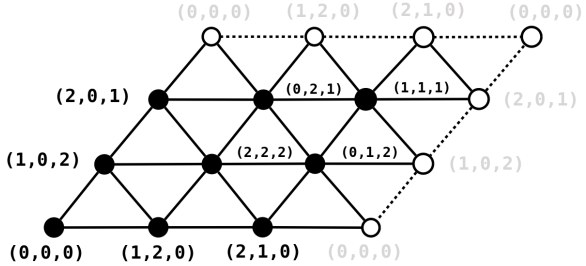
Para compreender melhor essa definição, considere o caso ilustrado na Figura 8a, que apresenta o grafo $H_{3,3}$. Como $l = 3$, cada vértice é uma tripla ordenada (x_1, x_2, x_3), na qual cada coordenada assume valores no conjunto $\{0, 1, \ldots, p-1\}$. No exemplo, temos $p = 3$, de modo que as coordenadas podem assumir os valores 0, 1 ou 2. Entretanto, uma tripla só é considerada um vértice válido se a soma de suas coordenadas for congruente a zero módulo 3, isto é, se a soma for múltipla de 3. Formalmente, o conjunto de vértices é definido como:

$$V_{l,p} = \{(x_1, x_2, \ldots, x_l) \in \mathbb{Z}_p^l \mid \sum_{i=1}^{l} x_i \equiv 0 \pmod{p}\}.$$

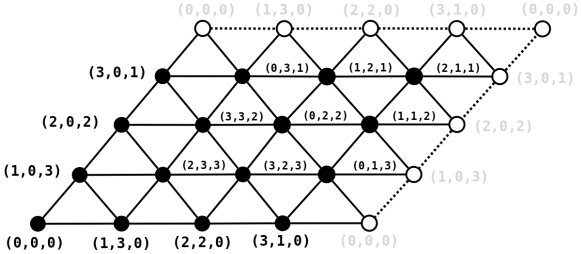
Assim, no grafo $H_{3,3}$, o vértice (0,0,0) pertence ao conjunto $V_{3,3}$, enquanto (1,0,0) não, pois $1 + 0 + 0 \equiv 1 \pmod 3$.

Figura 8 – Representação dos Grafos $H_{l,p}$

(a) Grafo $H_{3,3}$ com repetição de Vértices.



(b) Grafo $H_{3,4}$ com repetição de Vértices.



Fonte: Baseado em Ribeiro (2013).

A relação de adjacência entre os vértices também segue uma regra bem definida: dois vértices $x = (x_1, \ldots, x_l)$ e $y = (y_1, \ldots, y_l)$ são conectados por uma aresta se diferirem exatamente em duas coordenadas, de modo que uma dessas coordenadas aumenta em uma unidade e a outra diminui em uma unidade. Dessa forma, mantém-se a soma total congruente a zero módulo p. Formalmente, o conjunto de arestas é dado por:

$$E_{l,p} = \{(x,y) : \text{existem } i, j \text{ tais que } y_k \equiv x_k \text{ para todo } k \neq \{i, j\},$$
$$\text{e } (y_i \equiv x_i + 1, y_j \equiv x_j - 1) \text{ ou } (y_i \equiv x_i - 1, y_j \equiv x_j + 1)\}.$$

Essa definição permite interpretar $H_{l,p}$ como um grafo de Cayley, em que o conjunto gerador é formado pelos deslocamentos possíveis entre as coordenadas:

$$S = \{e_i - e_j \mid i,j \in \{1, \ldots, l\},\ i \neq j\},$$

onde e_i representa o vetor unitário com 1 na posição i e 0 nas demais. Aplicando os deslocamentos definidos por S a qualquer vértice $x \in V_{l,p}$, obtemos todos os vértices adjacentes a ele, garantindo que cada vértice tenha o mesmo grau:

$$\Delta = l(l-1).$$

Voltando ao exemplo da Figura 8a, o vértice (0,0,0) conecta-se a (1,2,0) através do gerador $(1,-1,0)$, e a (0,1,2) pelo gerador $(0,1,-1)$. Essas operações respeitam o módulo 3 e mantêm a soma das coordenadas igual a zero módulo 3. De modo análogo, na Figura 8b, o grafo $H_{3,4}$ segue as mesmas regras, mas com coordenadas variando de 0 a 3. Assim, o vértice (0,0,0) conecta-se, por exemplo, a (1,3,0), (0,1,3) e (3,0,1), calculadas módulo 4.

Observa-se que o número de vértices cresce rapidamente com o aumento de p e l. Enquanto $H_{3,3}$ possui $p^{l-1} = 3^2 = 9$ vértices, o grafo $H_{3,4}$ já conta com $p^{l-1} = 4^2 = 16$ vértices, ilustrando o crescimento exponencial dessa estrutura em relação aos parâmetros escolhidos.

O grafo $H_{l,p}$ apresenta um conjunto de propriedades formais que o tornam particularmente útil em estudos teóricos de coloração e simetria. De acordo com Ribeiro (2013) e Teixeira (2023), suas principais características são:

- O número total de vértices é $n = p^{l-1}$;
- O grau de cada vértice é $l(l-1)$;
- O número total de arestas é $\binom{l}{2} p^{l-1}$;
- $H_{l,p}$ é conexo e vértice-transitivo, o que significa que todos os vértices possuem papéis equivalentes na estrutura do grafo.
- O grafo cresce exponencialmente com l e p, o que implica um aumento significativo no custo computacional de geração e manipulação.

2.3.1 Coloração de Arestas nos Grafos $H_{l,p}$

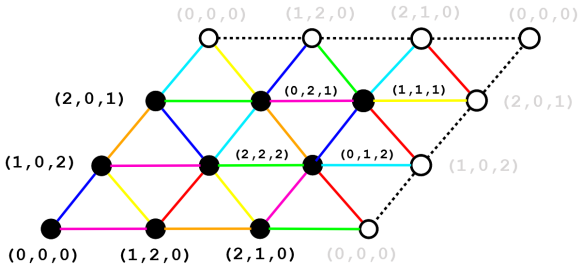
No estudo da coloração de arestas dos grafos $H_{l,p}$, o comportamento difere de acordo com a paridade do parâmetro p. Quando p é par, exploramos uma propriedade estrutural fundamental: cada elemento do conjunto gerador de $H_{l,p}$, quando aplicado sucessivamente p vezes a um vértice, retorna ao vértice inicial, formando um ciclo de comprimento p. Como p é par, esses ciclos são necessariamente pares, permitindo que suas arestas sejam coloridas de maneira alternada com apenas duas cores, sem conflitos. Dessa forma, a coloração global do grafo é obtida pela união das colorações de todos esses ciclos. Em consequência, para o grafo $H_{l,p}$ com p pares, a coloração das arestas exige exatamente $\Delta(H_{l,p})$ cores, onde $\Delta(H_{l,p})$ representa o grau máximo do grafo.

Essa característica pode ser visualizada na Figura 9b, que mostra a coloração das arestas do grafo $H_{3,4}$. Observa-se que, como $p = 4$ é par, as arestas foram coloridas alternadamente dentro de cada ciclo, produzindo uma distribuição simétrica e sem conflitos de cores. Já na Figura 9a, para $p = 3$, temos um cenário distinto: como o parâmetro é ímpar, os ciclos formados possuem comprimento ímpar, o que impede a coloração alternada sem repetição de cores adjacentes.

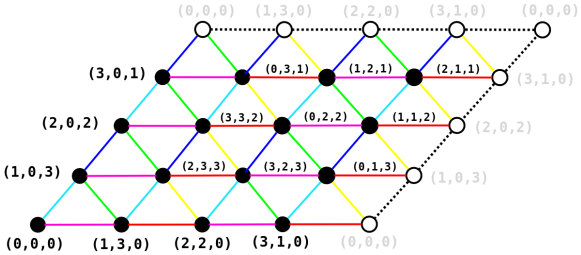
Quando o parâmetro p é ímpar, o método baseado na coloração de ciclos não pode ser aplicado. O motivo é que ciclos ímpares não podem ser coloridos alternadamente com apenas duas cores sem gerar conflitos. Nesse caso, utiliza-se o algoritmo de Vizing

Figura 9 – Coloração de Arestas no Grafo $H_{l,p}$.

(a) Grafo $H_{3,3}$ com p ímpar.



(b) Grafo $H_{3,4}$ com p par.



Fonte: Baseado em Teixeira (2023).

(também conhecido como algoritmo de Misra–Gries), o qual estabelece limites superiores e inferiores para o número de cores necessárias para colorir adequadamente as arestas de qualquer grafo simples.

Teorema 1 (Teorema de Vizing) *Seja G um grafo simples com grau máximo Δ. Então o índice cromático $\chi'(G)$ satisfaz:*

$$\Delta \leq \chi'(G) \leq \Delta + 1.$$

Este resultado fundamental de Vizing (1964) mostra que o índice cromático de qualquer grafo simples está restrito a apenas dois valores possíveis. Essa forte restrição deu origem a um problema clássico conhecido como o Problema da Classificação, que consiste em determinar a qual das duas categorias um grafo pertence. Os grafos são, portanto, classificados da seguinte forma:

- **Classe 1:** se seu índice cromático é igual ao seu grau máximo, ou seja, $\chi'(G) = \Delta(G)$.
- **Classe 2:** se seu índice cromático é $\chi'(G) = \Delta(G) + 1$.

Embora a escolha seja limitada a apenas duas opções, determinar a qual classe um grafo pertence é um problema NP-completo, como demonstrado por Holyer (1981).

Prova. O limite inferior $\chi'(G) \geq \Delta$ é imediato, pois cada vértice de grau máximo Δ possui Δ arestas incidentes, que devem receber cores distintas.

Para o limite superior, o algoritmo de Misra–Gries constrói iterativamente uma coloração válida das arestas.

Em cada passo, uma nova aresta é considerada, e, através da construção de um *fan* maximal e da inversão de caminhos coloridos, garante-se que sempre existe uma cor disponível entre $\Delta + 1$ possibilidades. Logo, nunca são necessárias mais do que $\Delta + 1$ cores. □

O conceito de *fan maximal* (ou leque maximal) desempenha um papel central na execução do algoritmo de Misra–Gries. Dado um vértice x e um conjunto de arestas incidentes a ele, um *fan* é uma sequência ordenada de vértices adjacentes $F = (v_1, v_2, \ldots, v_k)$ tal que:

1. cada v_i é adjacente a x, isto é, $(x, v_i) \in E(G)$;
2. para cada $i > 1$, a cor atribuída à aresta (x, v_{i-1}) está livre em v_i.

Um *fan* é dito *maximal* quando não é possível adicionar novos vértices à sequência sem violar essas propriedades.

Durante a execução do algoritmo, o *fan maximal* é utilizado para rearranjar cores localmente ao redor de um vértice x, liberando uma cor disponível para uma nova aresta que ainda não pôde ser colorida. Esse rearranjo é realizado por meio de duas operações fundamentais: a inversão de caminhos coloridos e a rotação do *fan*, garantindo que sempre haja uma cor disponível entre as $\Delta + 1$ cores possíveis.

2.4 Algoritmo ÍMPAR

No caso em que o parâmetro p é ímpar, a estratégia de coloração por ciclos alternantes não é aplicável, conforme discutido na Seção Coloração de Arestas nos Grafos $H_{l,p}$, uma vez que ciclos de comprimento ímpar não admitem uma coloração alternada com duas cores. Para esses casos, aplica-se o algoritmo de Misra–Gries (também referido como algoritmo de Vizing), que garante uma coloração própria utilizando no máximo $\Delta + 1$ cores.

2.4.1 Pseudocódigo do Algoritmo de Misra–Gries

Algoritmo 1: Algoritmo de Misra–Gries para coloração de arestas

Entrada: Grafo $H_{l,p}$ com p ímpar
Saída: Coloração própria das arestas utilizando no máximo $\Delta + 1$ cores

$U \leftarrow E(H_{l,p})$
enquanto $U \neq \emptyset$ **faça**
  selecione uma aresta $(x,v) \in U$
  construa um *fan* $F = (v_1, v_2, \ldots, v_k)$ em torno de x, com $F[1] = v$
  seja c uma cor livre em x, e d uma cor livre em $F[k]$
  inverta o caminho cdx (se existir)
  encontre $w \in \{1, \ldots, k\}$ tal que $F' = (v_1, \ldots, v_w)$ é um fan e d é livre em $F[w]$
  rotacione F'
  atribua a cor d à aresta $(x,F[w])$
  $U \leftarrow U - \{(x,v)\}$
fim

Fonte: Adaptado de Wikipedia (2024).

Esse procedimento garante que, em cada iteração, uma nova aresta é colorida de forma consistente, até que todas as arestas tenham sido processadas.

2.4.2 Corretude do Algoritmo

Teorema 2 *Seja $H_{l,p}$ com p ímpar. Então a coloração obtida pelo algoritmo de Misra–Gries é própria e utiliza no máximo $\Delta(H_{l,p}) + 1$ cores.*

Prova. O algoritmo mantém a propriedade de que, em cada vértice, as arestas incidentes possuem cores distintas. Quando uma aresta não pode receber imediatamente uma cor, o procedimento de *inversão de caminho alternado* reorganiza as cores de um subconjunto de arestas, liberando uma cor disponível.

Além disso, pela teoria de Vizing, sempre existe uma cor livre entre $\Delta + 1$ possibilidades. Portanto, o algoritmo nunca falha e a coloração final é própria, respeitando o limite superior de $\Delta + 1$ cores. □

2.4.3 Complexidade do Algoritmo

Teorema 3 *O algoritmo de Misra–Gries para $H_{l,p}$ possui complexidade de tempo $O(|V||E|)$.*

Prova. Cada aresta é processada exatamente uma vez no laço principal. O custo de cada iteração pode envolver a construção de um *fan* e operações de inversão de caminhos, cujo custo é no máximo linear no número de vértices. Assim, o custo total é limitado por:

$$O(|E| \cdot |V|).$$

No caso de $H_{l,p}$, temos:

$$|V| = p^{l-1}, \qquad |E| = \binom{l}{2} \cdot p^{l-1}.$$

Logo, a complexidade é:

$$O\left(p^{2(l-1)} \cdot l^2\right),$$

o que, embora maior que o caso par, ainda é polinomial e viável para valores moderados de l e p. □

2.5 Algoritmo PAR

Quando o parâmetro p é par, o grafo $H_{l,p}$ admite uma coloração de arestas baseada em ciclos alternantes. Cada vetor do conjunto gerador de $H_{l,p}$, quando aplicado sucessivamente a um vértice, percorre um ciclo de comprimento p. Como p é par, esses ciclos podem ser coloridos alternadamente com duas cores distintas, garantindo uma coloração própria.

2.5.1 Pseudocódigo do Algoritmo de Ciclos Alternantes

Algoritmo 2: Coloração de arestas por ciclos alternantes

Entrada: Grafo $H_{l,p}$ com p par

Saída: Coloração própria das arestas com exatamente Δ cores

```
1 U ← E(H_{l,p})
2 enquanto U ≠ ∅ faça
3 |   selecione uma aresta (u,v) ∈ U
4 |   identifique o gerador g que leva u a v
5 |   construa o ciclo C gerado por g, de comprimento p
6 |   atribua cores alternadas c_1, c_2 às arestas de C
7 |   U ← U − E(C)
8 fim
```

Fonte: Adaptado de Araújo (2024).

Esse procedimento colore todos os ciclos associados aos geradores, removendo-os da lista de arestas ainda não processadas. Como cada aresta pertence a exatamente um desses ciclos, o algoritmo termina com todas as arestas coloridas.

2.5.2 Corretude do Algoritmo

Teorema 4 *Seja $H_{l,p}$ com p par. Então a coloração obtida pelo algoritmo de ciclos alternantes é própria e utiliza exatamente $\Delta(H_{l,p})$ cores.*

Prova. Cada vetor gerador de $H_{l,p}$, quando aplicado sucessivas vezes a um vértice inicial, gera um ciclo de comprimento p. Como p é par, esse ciclo pode ser 2-colorido alternadamente sem conflitos. Ao processar todos os geradores, cobrimos todas as arestas do grafo com colorações próprias.

Além disso, como cada vértice possui grau máximo $\Delta(H_{l,p})$, o número de cores distintas utilizadas é exatamente $\Delta(H_{l,p})$, que é o menor possível. Assim, a coloração obtida é ótima.

□

2.5.3 Complexidade do Algoritmo

Teorema 5 *O algoritmo de ciclos alternantes para $H_{l,p}$ possui complexidade de tempo $O(|E|)$.*

Prova. Cada aresta pertence a exatamente um ciclo construído pelo algoritmo. Uma vez que um ciclo é processado, todas as suas arestas são removidas do conjunto U. Logo, cada aresta é visitada e colorida apenas uma vez.

Assim, o custo total é linear no número de arestas:

$$O(|E|).$$

No caso de $H_{l,p}$, temos que $|E| = \binom{l}{2} \cdot p^{l-1}$. Portanto, a complexidade assintótica permanece

$$O\left(\binom{l}{2} \cdot p^{l-1}\right),$$

que é linear em relação ao tamanho do grafo e, portanto, eficiente. □

3 TRABALHOS RELACIONADOS

Este Capítulo contextualiza a presente pesquisa, analisando trabalhos teóricos sobre a família de grafos $H_{l,p}$ e ferramentas de software para visualização de grafos em geral. O objetivo é demonstrar como este projeto se baseia em estudos anteriores e qual é o seu diferencial em relação às soluções existentes.

3.1 Estudos sobre a Estrutura e Coloração dos Grafos $H_{l,p}$

A base teórica para o estudo da família de grafos $H_{l,p}$ foi consolidada por Ribeiro (2013), que os identificou formalmente como uma classe de grafos de Cayley e estabeleceu suas propriedades fundamentais. Essa fundamentação é o ponto de partida para investigações mais aprofundadas sobre suas características cromáticas.

O estudo mais diretamente relacionado a este Trabalho é o de Araújo (2024), apresentado no 11th Latin American Workshop on Cliques in Graphs (LAWCG). Em seu trabalho, os autores investigam a coloração de arestas dos grafos $H_{l,p}$ e, de forma análoga a este TCC, dividem o problema com base na paridade do parâmetro p. Eles concluem que o índice cromático é Δ para p par e $\Delta + 1$ para p ímpar, resultados que validam a abordagem teórica e os algoritmos implementados na ferramenta desenvolvida neste projeto.

Além da coloração de arestas, a família $H_{l,p}$ tem sido objeto de estudo para outras variantes do problema. Teixeira (2023) investigou a coloração total nesses mesmos grafos, que envolve a atribuição de cores a vértices e arestas simultaneamente. A pesquisa de Teixeira confirma que a Conjectura da Coloração Total também se aplica a essa classe de grafos, demonstrando a relevância e a complexidade da família $H_{l,p}$ para a teoria dos grafos.

3.2 Ferramentas para Visualização de Grafos

A visualização de grafos é uma área consolidada, com diversas ferramentas disponíveis. Softwares de desktop como Gephi e Graphviz são amplamente utilizados para análise de redes complexas e geração de diagramas precisos. No âmbito das plataformas web, soluções como Graph Online Editor, VisuAlgo e CS Academy Graph Editor oferecem ambientes interativos para desenhar, editar e aplicar algoritmos em grafos diretamente no navegador, sendo recursos valiosos, especialmente no contexto educacional.

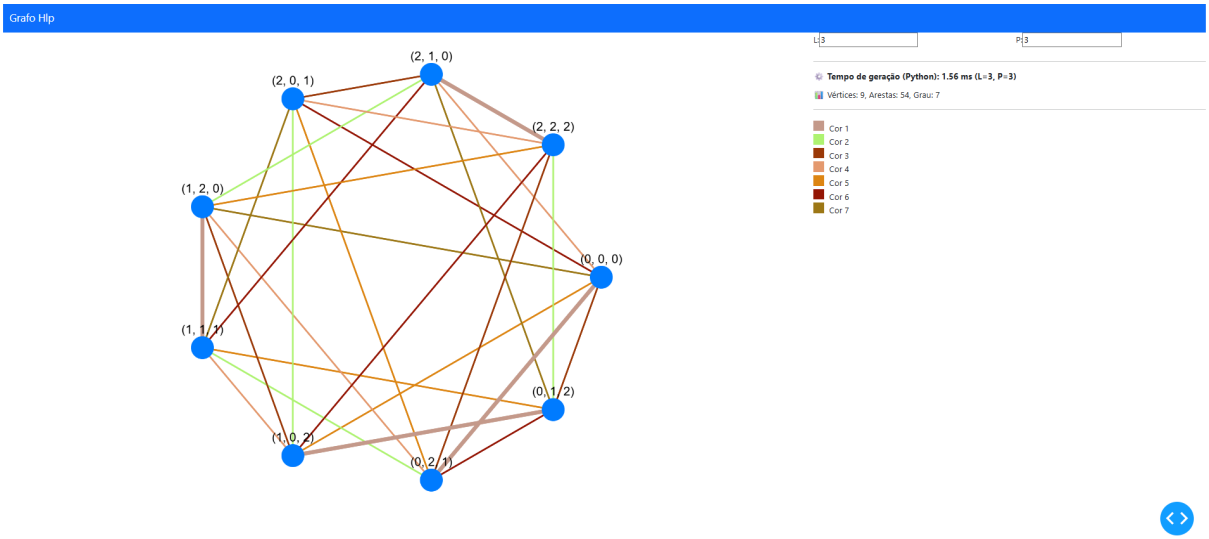
A ferramenta desenvolvida neste TCC se diferencia desses trabalhos pela sua especialização. Enquanto as soluções mencionadas são de propósito geral, a aplicação aqui proposta é um sistema focado exclusivamente na família $H_{l,p}$. Seu objetivo não é apenas renderizar um grafo, mas sim servir como um laboratório interativo para um problema

teórico específico: a coloração de arestas baseada na paridade de p. A aplicação automatiza a geração da estrutura complexa do $H_{l,p}$, implementa os dois algoritmos de coloração distintos (por ciclos alternantes e Misra-Gries) e permite a análise visual dos resultados. Dessa forma, ela preenche uma lacuna ao oferecer um recurso didático e investigativo que não é encontrado em visualizadores genéricos.

4 Implementação da Ferramenta de Visualização

Neste Capítulo, detalhamos o processo de desenvolvimento da ferramenta computacional criada para a visualização da coloração de arestas nos grafos de Cayley $H_{l,p}$. O objetivo foi construir uma aplicação web interativa que permitisse aos usuários gerar instâncias do grafo, aplicar os algoritmos de coloração discutidos no Capítulo anterior e analisar visualmente os resultados.

Figura 10 – Interface da Ferramenta.



Fonte: Autoria própria.

4.1 Arquitetura e Tecnologias Utilizadas

A ferramenta foi desenvolvida em linguagem **Python (versão 3.11.8),** devido à sua robustez e ao vasto ecossistema de bibliotecas para análise de dados e desenvolvimento web. O ambiente de desenvolvimento utilizado foi o Google Firebase Studio, que forneceu uma infraestrutura virtual com CPU Intel(R) Xeon(R) @ 2.20 GHz e 8 GB de memória RAM.

As principais bibliotecas que compõem a arquitetura da aplicação são:

- **Dash (versão 2.18.2)**: Utilizado como o framework principal para a construção da interface web interativa. O Dash permite criar aplicações analíticas complexas com interfaces reativas, escritas puramente em Python.
- **Dash Bootstrap Components (versão 1.6.0)**: Empregado para estilizar a interface da aplicação, garantindo um layout responsivo e intuitivo para o usuário.
- **NetworkX (versão 3.4.2)**: Biblioteca fundamental para a modelagem e manipulação das estruturas do grafo. Foi utilizada para gerar os vértices e as arestas do grafo $H_{l,p}$ com base nos parâmetros l e p fornecidos pelo usuário.

- **Dash Cytoscape (versão 1.0.2)**: Responsável pela renderização gráfica e interativa dos grafos na interface web. Esta biblioteca permite a aplicação de estilos visuais, como cores e larguras de arestas, que são essenciais para a visualização das colorações.

4.2 Estrutura do Código-fonte

O código-fonte da aplicação, apresentado no **Apêndice A**, foi organizado em módulos funcionais para garantir clareza e manutenibilidade. A estrutura principal pode ser dividida nas seguintes etapas:

4.2.1 Geração do Grafo $H_{l,p}$

A geração dos vértices é realizada pela função `gerar_vertices_Hlp(L, P)`, que cria todas as triplas $(x_1, \ldots, x_l)$ pertencentes a $\mathbb{Z}_p^l$ cuja soma das coordenadas é congruente a zero módulo p. As arestas são obtidas pela função `gerar_arestas_Hlp(vertices, gerador, P)`, que utiliza o conjunto gerador

$$S = \{e_i - e_j \mid i,j \in \{1, \ldots, l\},\ i \neq j\},$$

conectando vértices que diferem em exatamente duas coordenadas (uma incrementada e outra decrementada).

Os vértices são posicionados de forma radial, em torno de um ponto central, com coordenadas calculadas por:

$$x = c_x + r \cdot \cos\left(\frac{2\pi i}{n}\right), \quad y = c_y + r \cdot \sin\left(\frac{2\pi i}{n}\right),$$

em que $n = |V_{l,p}|$, r é o raio da circunferência, e (c_x, c_y) define o centro da visualização. Esse método favorece a percepção global das conexões e mantém o grafo legível, especialmente em instâncias pequenas.

4.2.2 Aplicação dos Algoritmos de Coloração

Após a geração do grafo, a aplicação seleciona o algoritmo de coloração apropriado com base na paridade do parâmetro p:

- **Se p for par**, a função `colorir_arestas` é invocada. Esta função implementa o Algoritmo 2, que identifica os ciclos gerados por cada gerador e os colore de forma alternada, resultando em uma coloração com Δ cores.
- **Se p for ímpar**, a função `vizing` é chamada. Ela implementa o Algoritmo 1, que garante uma coloração válida com no máximo $\Delta + 1$ cores.

Ambas as funções retornam um dicionário que mapeia cada aresta a um número inteiro representando sua cor.

Internamente, a aplicação gera uma paleta com $\Delta + 1$ cores, onde $\Delta = l(l-1)$ é o grau máximo do grafo $H_{l,p}$. Essa escolha garante a compatibilidade com o Teorema de Vizing e evita erros de indexação, já que os índices de cor utilizados são iniciados em 1.

Apesar de o sistema gerar corretamente as colorações de acordo com os algoritmos implementados, a percepção visual das cores apresenta limitações práticas. As cores das arestas são atribuídas de forma pseudoaleatória, o que pode resultar em tonalidades visualmente semelhantes, especialmente em grafos que utilizam um número elevado de cores. Essa limitação não compromete a validade dos algoritmos, mas pode dificultar a distinção perceptiva entre arestas adjacentes quando o número de cores se aproxima de $\Delta(H_{l,p}) + 1$.

Além disso, o aumento da densidade do grafo faz com que múltiplas arestas se sobreponham na renderização, o que reduz a clareza da visualização. Trata-se de uma restrição inerente ao meio de exibição — a tela do navegador e a capacidade perceptiva do usuário — e não à implementação dos métodos de coloração. Como possível melhoria, recomenda-se empregar paletas perceptualmente otimizadas ou escalas de contraste adaptativo em trabalhos futuros.

Para contornar este problema, foi criada uma função de destaque de arestas, que visualmente deixa a aresta mais grossa para distingui-las das demais. Assim, mesmo com uma paleta de cores visualmente semelhante, a função garante discernir as arestas desejadas.

4.2.3 Visualização e Interatividade

A etapa final é a renderização do grafo colorido na interface.

1. **Estrutura de Dados para Cytoscape**: Os dados do grafo (vértices e arestas coloridas) são convertidos para o formato JSON, que é o padrão exigido pelo Dash Cytoscape. Cada aresta no JSON resultante contém um atributo `color`, que define sua cor na visualização.
2. **Layout e Callbacks**: O layout da aplicação é definido usando componentes do Dash Bootstrap. A interatividade é gerenciada por *callbacks* do Dash. O principal *callback*, a função `update_graph`, é acionado sempre que o usuário altera os valores de l ou p. Ele reexecuta todo o pipeline: geração do grafo, coloração e atualização dos elementos visuais na tela.
3. **Recursos de Interação**: A ferramenta também inclui funcionalidades para destacar arestas. O usuário pode clicar em um vértice para destacar todas as arestas incidentes a ele, ou clicar em uma cor na legenda para destacar todas as arestas daquela cor. Isso é implementado por um segundo *callback* que modifica dinamicamente a folha de estilos (stylesheet) do Cytoscape para aumentar a largura das arestas selecionadas.

Dessa forma, a implementação combina o rigor dos algoritmos matemáticos com uma interface gráfica acessível, permitindo que a teoria da coloração de arestas nos grafos

$H_{l,p}$ seja visualizada e explorada de forma computacional e interativa.

Para manter o desempenho e a legibilidade, a visualização é limitada a instâncias com $3 \leq l \leq 5$ e $3 \leq p \leq 5$. Valores superiores resultam em uma mensagem informando que o grafo é muito grande para ser renderizado no navegador. Tal limitação reflete a complexidade combinatória de $H_{l,p}$, cujo número de vértices cresce segundo p^{l-1}.

O tempo de geração mostrado na interface corresponde apenas ao tempo de execução das funções em Python (geração dos vértices, arestas e coloração), não incluindo o tempo de renderização no navegador. Essa medição é apresentada no formato:

$$\text{Tempo de geração (Python): } t = (t_{\text{final}} - t_{\text{inicial}}) \times 1000 \text{ ms.}$$

5 Conclusão

5.1 Resultados Experimentais

Para avaliar o desempenho prático dos algoritmos implementados, foram realizados testes variando os parâmetros l e p. Os resultados do tempo de geração em Python para diferentes instâncias do grafo $H_{l,p}$ estão apresentados na Tabela 1.

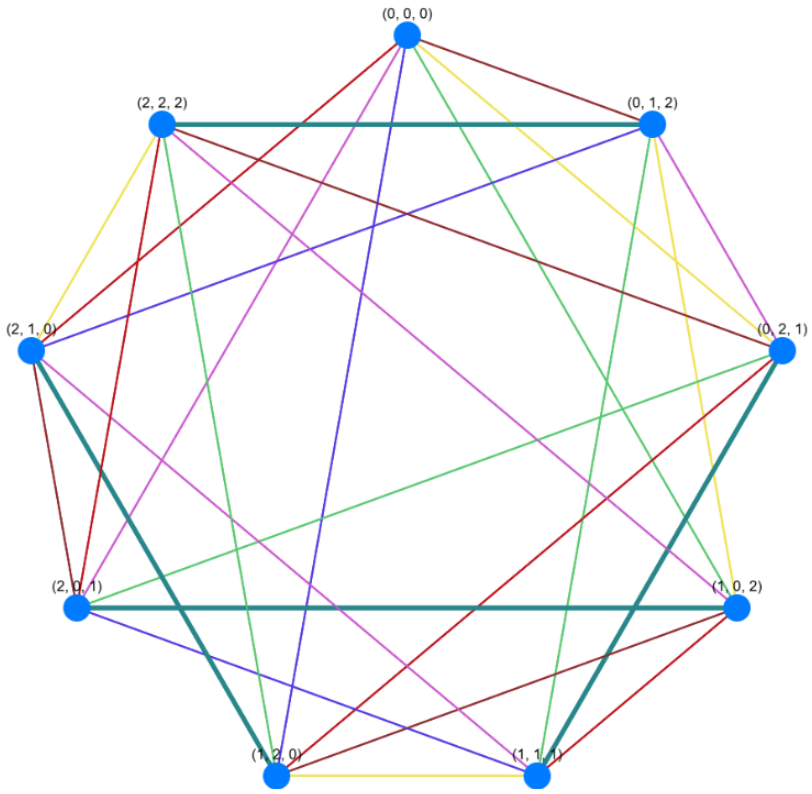
Tabela 1 – Tempo de geração dos grafos $H_{l,p}$ em Python.

l	p	Tempo de geração (ms)	Nº de V	Nº de E	Δ
3	3	1.48	9	27	6
3	4	2.36	16	48	6
3	5	2.63	25	75	6
3	6	2.95	36	108	6
4	3	7.32	27	162	12
4	4	10.59	64	384	12
4	5	17.26	125	750	12
4	6	39.23	216	1.296	12
5	3	15.35	81	810	20
5	4	284.34	256	2.560	20
5	5	1019.13	625	6.250	20
5	6	1968.43	1.296	12.960	20
6	3	109.72	243	3.645	30
6	4	1669.10	1.024	15.360	30
6	5	12761.56	3.125	46.875	30
6	6	99116.57	7.776	116.640	30

Fonte: Autoria própria.

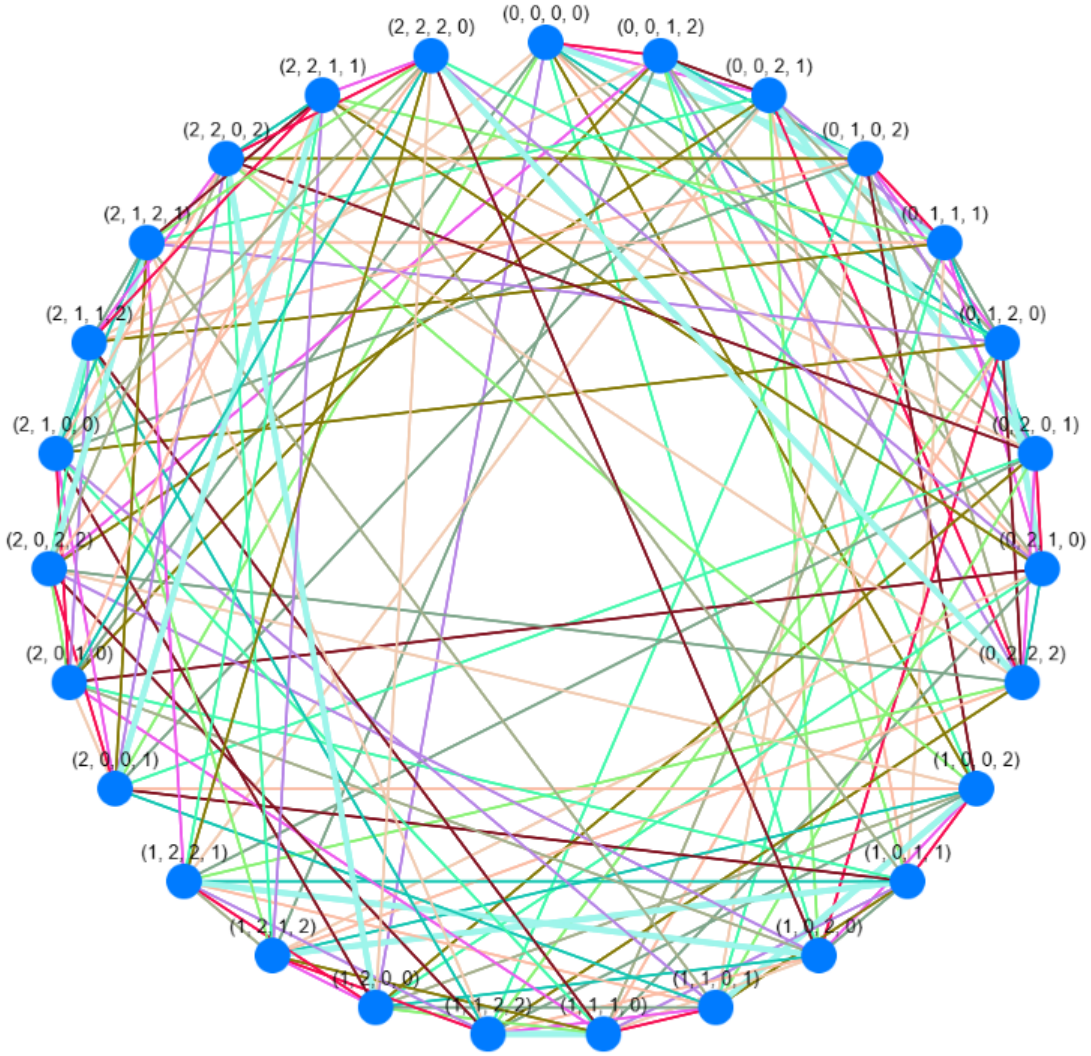
Observa-se que o crescimento dos tempos segue diretamente a complexidade teórica dos algoritmos. Para instâncias menores, como $(l,p) = (3,3)$ representado na ferramenta na Figura 11 ou (4,3) representado na ferramenta na Figura 12, o tempo de geração é praticamente instantâneo (da ordem de milissegundos). No entanto, para instâncias maiores, o custo cresce rapidamente: em (5,5), o tempo de geração já ultrapassa um segundo, enquanto para (6,6), o processamento ultrapassou 99 segundos apenas para a geração no backend Python.

Figura 11 – Grafo $H_{3,3}$ visualizado na ferramenta.



Fonte: Autoria própria.

Figura 12 – Grafo $H_{4,3}$ visualizado na ferramenta.

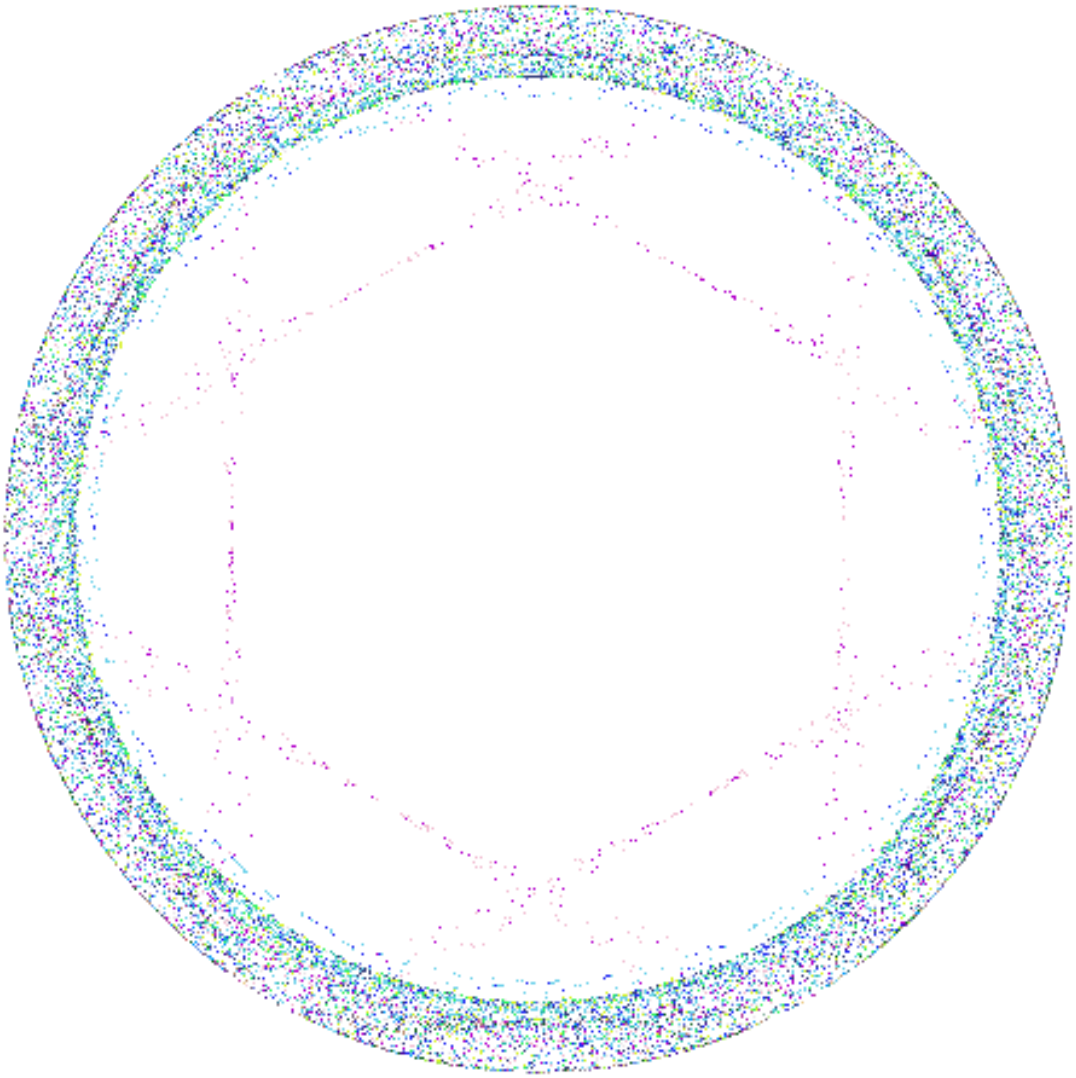


Fonte: Autoria própria.

É importante destacar que a etapa de renderização no navegador também cresce de forma significativa. Por exemplo, a instância (6,6) representada na ferramenta na Figura 13 demandou aproximadamente 2 minutos adicionais para que o grafo pudesse ser visualizado de forma interativa, além de a representação do mesmo ser ilegível na ferramenta, o que causou lentidão na manipulação do grafo. Essa limitação prática justifica o corte aplicado na ferramenta, restringindo a visualização a instâncias com $l, p < 6$.

Esses resultados confirmam, portanto, a boa escalabilidade do algoritmo baseado em ciclos (caso par), mas também evidenciam a complexidade superior do algoritmo de Misra–Gries (caso ímpar), cuja execução para parâmetros elevados torna-se computacionalmente custosa.

Figura 13 – Grafo $H_{6,6}$ visualizado na ferramenta.



Fonte: Autoria própria.

5.2 Síntese e Discussão dos Resultados

O presente trabalho teve como objetivo o estudo da coloração de arestas em grafos de Cayley $H_{l,p}$ e a implementação de uma ferramenta de visualização interativa que permite explorar diferentes instâncias desses grafos. Foram consideradas duas abordagens distintas: a coloração baseada em ciclos, utilizada quando p é par, e o algoritmo de Vizing/Misra–Gries, aplicado quando p é ímpar.

No caso em que p é par, mostrou-se que a estrutura do grafo permite a decomposição em ciclos de comprimento par. Essa propriedade garante a possibilidade de uma coloração alternada com exatamente $\Delta = l(l-1)$ cores, o que corresponde ao índice cromático mínimo do grafo. O algoritmo implementado possui complexidade $O(|E|)$, uma vez que cada aresta é visitada uma única vez no processo de coloração. Dessa forma, além de

teoricamente ótimo, o método é computacionalmente eficiente e escalável.

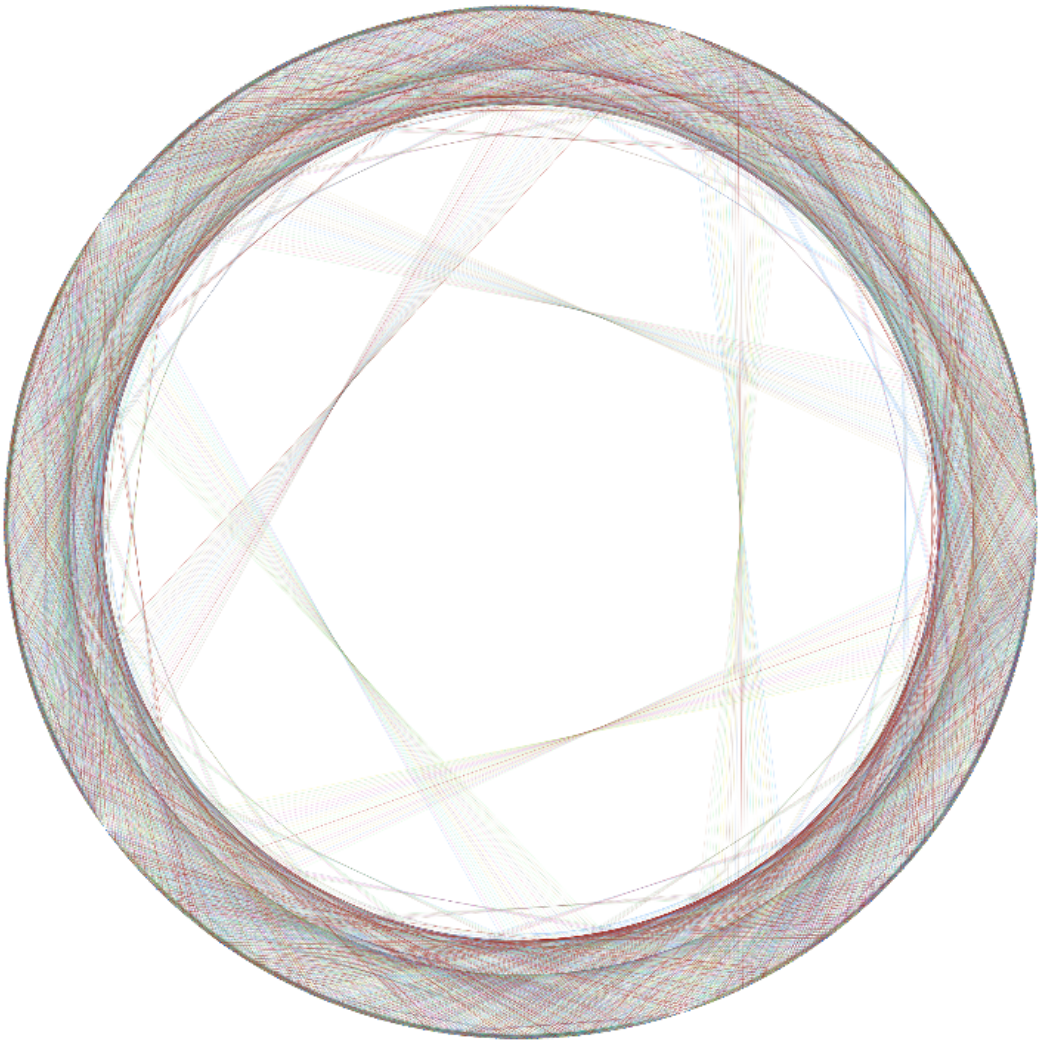
Por outro lado, quando p é ímpar, a presença de ciclos ímpares inviabiliza o procedimento anterior, exigindo o uso do algoritmo de Vizing/Misra–Gries. Nessa situação, o limite superior garantido é $\Delta+1$, de acordo com o Teorema de Vizing. Embora este algoritmo assegure uma coloração própria, sua complexidade é maior, da ordem de $O(|V||E|)$. Isso implica que, para valores elevados de l e p, a execução pode tornar-se significativamente mais custosa. Ressalta-se que, até o momento, não há procedimento conhecido que resolva o caso ímpar de forma mais eficiente que o algoritmo de Vizing, sendo essa uma limitação intrínseca ao problema.

A análise experimental mostrou que, enquanto instâncias pequenas, como $(l,p) = (3,3)$ ou $(4,3)$, são processadas em milissegundos, o tempo de execução cresce de forma acentuada para instâncias maiores. O caso $(6,6)$, por exemplo, levou cerca de 99 segundos apenas na etapa de geração em Python, além de aproximadamente 2 minutos adicionais para renderização no navegador. Esses resultados reforçam a escalabilidade do algoritmo para o caso par, mas também confirmam os limites práticos da abordagem para o caso ímpar em instâncias de grande porte.

A ferramenta desenvolvida cumpre seu propósito de ilustrar e validar os resultados teóricos, oferecendo uma interface visual que permite compreender, de forma intuitiva, as propriedades estruturais dos grafos $H_{l,p}$. Contudo, a escalabilidade constitui um fator limitante, especialmente em instâncias maiores. Na prática, a visualização foi restringida a valores de $p < 6$ e $l < 6$, de modo a preservar a clareza na representação e evitar sobrecarga computacional, uma vez que o número de vértices cresce exponencialmente com esses parâmetros.

Além disso, observou-se uma limitação perceptual: como as cores das arestas são atribuídas de forma pseudoaleatória, tonalidades visualmente semelhantes podem ocorrer, tornando difícil distinguir certas arestas quando muitas cores são utilizadas, como na Figura 14. Essa limitação não afeta a correção dos algoritmos, mas pode comprometer a percepção do usuário na análise visual de grafos densos.

Figura 14 – Grafo $H_{5,5}$ visualizado na ferramenta.



Fonte: Autoria própria.

5.3 Trabalhos Futuros

Este trabalho abre espaço para novas investigações. Entre as principais perspectivas, destacam-se:

- Estender a ferramenta para incluir a coloração de vértices, permitindo analisar a relação entre a coloração de vértices e de arestas em $H_{l,p}$;
- Investigar a coloração total em grafos de Cayley $H_{l,p}$, que combina simultaneamente a coloração de vértices e arestas;
- Explorar melhorias de desempenho no algoritmo de Vizing por meio de otimizações específicas para a estrutura de $H_{l,p}$;
- Ampliar o escopo da ferramenta de visualização para lidar com instâncias maiores, incorporando técnicas de paralelismo e otimização gráfica.

5.4 Considerações Finais

Em suma, o estudo realizado evidencia a riqueza estrutural dos grafos $H_{l,p}$ e mostra que a interação entre teoria e implementação prática é fundamental para o avanço do conhecimento na área. A distinção entre os casos par e ímpar revela tanto a elegância quanto os desafios do problema de coloração de arestas, ao mesmo tempo em que abre caminhos promissores para pesquisas futuras.

Referências

ALMEIDA, S. M. de. *Problemas de Coloração em Grafos*. 2018. Último acesso em: 01/10/2025. Disponível em: <https://sheilaalmeida.pro.br/coloracao.html>. Citado na página 6.

ALVES, R. P. et al. *Coloração de grafos e aplicações*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2015. Citado na página 1.

ARAúJO, A. J. *Edge Coloring of the Graph Hl,p*. 2024. Acesso em: 06 out. 2025. Disponível em: <https://www.lawcg.mat.br/lawcg24/LAWCG2024-annals.pdf>. Citado 2 vezes nas páginas 14 e 16.

BONDY, J. A.; MURTY, U. S. R. *Graph Theory*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008. ISBN 1846289696. Citado 3 vezes nas páginas 1, 4 e 6.

BRANDSTäDT, A.; LE, V. B.; SPINRAD, J. P. *Graph Classes: A Survey*. [S.l.]: Society for Industrial and Applied Mathematics, 1999. Citado na página 4.

CAYLEY. *Cayley*. 2021. Último acesso em: 01/10/2025. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Grafo_de_Cayley&oldid=62003202>. Citado na página 7.

DANTAS, A. *Pontes e Navegador de Posicionamento Global por Satélite*. 2010. Último acesso em: 01/10/2025. Disponível em: <https://www.autoentusiastasclassic.com.br/2010/08/folguedos-pontes-e-gps.html?m=0v>. Citado na página 3.

DANTAS, N. M. *Sobre o Problema da Lista Coloração em Grafos*. Tese (Doutorado) — Universidade Federal do Amazonas, 2015. Citado na página 5.

DIESTEL, R. *Graph Theory*. [S.l.]: Springer, 2017. Citado 2 vezes nas páginas 4 e 6.

FEOFILOFF, P. *Algoritmos para Grafos via Sedgewicks*. 2017. Último acesso em: 01/10/2025. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/graphs.html>. Citado na página 4.

GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. [S.l.]: Freeman, 1979. Citado 3 vezes nas páginas 1, 4 e 5.

GOLUMBIC, M. C. Algorithmic graph theory and perfect graphs. *Annals of discrete mathematics*, 2004. Citado na página 4.

GROSS, J.; YELLEN, J.; ZHANG, P. *Handbook of Graph Theory, Second Edition*. [S.l.]: Chapman and Hall/CRC, 2013. Citado 3 vezes nas páginas 1, 3 e 4.

HOLYER, I. The np-completeness of some edge-partition problems. *SIAM Journal on Computing*, v. 10, n. 4, p. 713–717, 1981. Último acesso em: 01/10/2025. Disponível em: <https://doi.org/10.1137/0210054>. Citado 2 vezes nas páginas 8 e 11.

RIBEIRO, C. *Estruturas e Aplicações em Teoria dos Grafos*. Rio de Janeiro, Brasil: Editora Acadêmica, 2013. Citado 5 vezes nas páginas 7, 8, 9, 10 e 16.

TEIXEIRA, T. d. S. *Coloração Total nos Grafos de Cayley H(l,p)*. 2023. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação). Disponível em: <https://repositorio.ifgoiano.edu.br/bitstream/prefix/3881/1/tcc_Thaynara%20dos%20Santos.pdf>. Citado 3 vezes nas páginas 10, 11 e 16.

VIZING, V. G. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz*, v. 3, p. 25–30, 1964. Citado na página 11.

VIZING, V. G. Some unsolved problems in graph theory. *Russian Mathematical Surveys*, v. 23, p. 125–141, 1968. Citado na página 1.

WEST, D. B. *Introduction to Graph Theory*. [S.l.]: Prentice Hall, 2001. Citado na página 4.

WIKIPEDIA. *PETERSEN*. 2021. Último acesso em: 01/10/2025. Disponível em: <https://pt.wikipedia.org/wiki/Grafo_de_Petersen>. Citado na página 8.

WIKIPEDIA. *COLORAÇÃO*. 2022. Último acesso em: 01/10/2025. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Colora%C3%A7%C3%A3o_de_grafos&oldid=64061099>. Citado na página 5.

WIKIPEDIA, M.-G. *Misra–Gries summary — Wikipedia, The Free Encyclopedia*. 2024. Acesso em: 06 out. 2025. Disponível em: <https://en.wikipedia.org/wiki/Misra%E2%80%93Gries_summary>. Citado na página 12.

Apêndices

APÊNDICE A – Código-fonte da aplicação em *Dash*

Este apêndice apresenta o código-fonte desenvolvido em *Python* (versão 3.11.8), utilizando o framework *Dash* (versão 2.18.2) e as bibliotecas *Dash Bootstrap Components* (versão 1.6.0), *NetworkX* (versão 3.4.2) e *Dash Cytoscape* (versão 1.0.2), para a construção da aplicação de visualização e coloração das arestas do grafo de Cayley $H_{l,p}$.

```
import dash
import json
import time
from dash import dcc, html
import networkx as nx
import dash_cytoscape as cyto
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output, State, ALL
from itertools import combinations, product
from random import randint
from math import cos, sin, pi


def conjunto_gerador(L, P):
    elementos_geradores = []
    for i, j in combinations(range(L), 2):
        vetor = [0] * L
        vetor[i] = 1
        vetor[j] = -1
        elementos_geradores.append(tuple(vetor))
        vetor = [0] * L
        vetor[i] = -1
        vetor[j] = 1
        elementos_geradores.append(tuple(vetor))
    return elementos_geradores

def gerar_vertices_Hlp(L, P):
    lista = list(product(range(P), repeat=L-1))
    vertices = []
    for v in lista:
        soma = 0
        for i in range(L-1):
            soma = soma + v[i]
        nova_coordenada = tuple(v) + (((P-(soma%P))%P),)
        vertices.append(nova_coordenada)
    return vertices

def gerar_arestas_Hlp(vertices, gerador, P):
    arestas = []
    i = 0
    for v in vertices:
        for g in gerador:
            nova_coordenada = tuple((v[i] + g[i]) % P for i in
```

```
                range(len(v)))
            for j in range(len(vertices)):
                if vertices[j] == nova_coordenada:
                    arestas.append((i+1, j+1))
                    break
        i += 1
    return arestas

def clear(N):
    C = [[0] * (N + 1) for _ in range(N + 1)]
    G = [[0] * (N + 1) for _ in range(N + 1)]
    return C, G

def update(X, C, u):
    X[u] = 1
    while C[u][X[u]]:
        X[u] += 1
    return X

def color(C, G, X, u, v, c):
    p = G[u][v]
    G[u][v] = G[v][u] = c
    C[u][c] = v
    C[v][c] = u
    C[u][p] = C[v][p] = 0
    if p:
        X[u] = X[v] = p
    else:
        X = update(X, C, u)
        X = update(X, C, v)
    return C, G, X, p

def flip(C, G, X, u, c1, c2):
    p = C[u][c1]
    C[u][c1], C[u][c2] = C[u][c2], C[u][c1]
    if p:
        G[u][p] = G[p][u] = c2
    if not C[u][c1]:
        X[u] = c1
    if not C[u][c2]:
        X[u] = c2
    return C, G, X, p

def vizing(C, G, X, E, N, M):
    for i in range(1, N + 1):
        X[i] = 1
    t = 0
    while t < len(E):
        u, v0 = E[t]
        v, c0 = v0, X[u]
        c = c0
        d = 0
```

```
95          L = []
96          vst = [0] * (N+1)
97          while not G[u][v0]:
98              L.append((v, d := X[v]))
99              if not C[v][c]:
100                 for a in range(len(L) - 1, -1, -1):
101                     C, G, X, c = color(C, G, X, u, L[a][0], c)
102             elif not C[u][d]:
103                 for a in range(len(L) - 1, -1, -1):
104                     C, G, X, h = color(C, G, X, u, L[a][0], L[a
                            ][1])
105             elif vst[d]:
106                 break
107             else:
108                 vst[d] = 1
109                 v = C[u][d]
110         if not G[u][v0]:
111             while v:
112                 C, G, X, v = flip(C, G, X, v, c, d)
113                 c, d = d, c
114             if C[u][c0]:
115                 a = 0
116                 for b in range(len(L) - 2, -1, -1):
117                     if L[b][1] == c:
118                         break
119                     a = b
120                 while(a>=0):
121                     C, G, X, h = color(C, G, X, u, L[a][0], L[a
                            ][1])
122                     a-=1
123             else:
124                 t -= 1
125         t += 1
126
127     dic = {}
128     for i in range(1, N + 1):
129         for j in range(1, N + 1):
130             if G[i][j] != 0:
131                 dic[(i, j)] = G[i][j]
132     return dic
133
134 def gerar_ciclo(u, e, vertices, P):
135     ciclo = [u]
136     v = tuple((u[i] + e[i]) % P for i in range(len(u)))
137     while v != u:
138         ciclo.append(v)
139         v = tuple((v[i] + e[i]) % P for i in range(len(v)))
140     return ciclo
141
142 def colorir_arestas(arestas, vertices, gerador, P):
143     cores = {}
144     U = set(arestas)
```

```
145
146     while U:
147         (u, v) = U.pop()
148         u_idx = u - 1
149         v_idx = v - 1
150         U.add((u, v))
151
152         e = list((vertices[v_idx][i] - vertices[u_idx][i]) % P for
                i in range(len(vertices[u_idx])))
153         ei = list((vertices[u_idx][i] - vertices[v_idx][i]) % P
               for i in range(len(vertices[v_idx])))
154
155         for j in range(len(e)):
156             if e[j] == P-1:
157                 e[j] = -1
158         e = tuple(e)
159
160         for j in range(len(ei)):
161             if ei[j] == P-1:
162                 ei[j] = -1
163         ei = tuple(ei)
164
165         ciclo = gerar_ciclo(vertices[u_idx], e, vertices, P)
166
167         for g in range(len(gerador)):
168             if e == gerador[g]:
169                 break
170
171         for k in range(len(gerador)):
172             if ei == gerador[k]:
173                 break
174
175         cor_e = g+1
176         cor_e_inv = k+1
177
178         for i in range(P):
179             u = ciclo[(i) % P]
180             v = ciclo[(i + 1) % P]
181             u_idx = vertices.index(u) + 1
182             v_idx = vertices.index(v) + 1
183             if (i % 2 == 0):
184                 cores[(u_idx, v_idx)] = cor_e
185                 cores[(v_idx, u_idx)] = cor_e
186                 U.remove((u_idx, v_idx))
187                 U.remove((v_idx, u_idx))
188             else:
189                 cores[(u_idx, v_idx)] = cor_e_inv
190                 cores[(v_idx, u_idx)] = cor_e_inv
191                 U.remove((u_idx, v_idx))
192                 U.remove((v_idx, u_idx))
193     return cores
194
```

```
195 used_colors = set()
196
197 def criar_Hlp(vertices, arestas, gerador, P, L):
198     global used_colors
199     app = dash.Dash(__name__, external_stylesheets=[dbc.themes.
        BOOTSTRAP])
200
201     server = app.server
202     @server.after_request
203     def remove_csp_header(response):
204         response.headers.pop('Content-Security-Policy', None)
205         return response
206
207     most_recently_clicked_color = None
208     selected_node = None
209
210     N = len(vertices)
211     M = len(arestas)
212     grau = (L * (L - 1)) + 1
213     cores = []
214
215
216     ca = []
217
218     # Criar matriz de cores
219     C, G = clear(N)
220     X = [0] * (N + 1)
221     for i in range(grau):
222         cores.append('#%06X' % randint(0, 0xFFFFFF))
223     if P % 2 == 0:
224         G = colorir_arestas(arestas, vertices, gerador, P)
225         #G = vizing(C, G, X, arestas, N, M)
226
227         ca = [
228         {
229           'data': {
230             'source': str(source),
231             'target': str(target),
232             'color': str(cores[G[(source, target)] - 1])
233             }
234         }
235       for source, target in arestas
236     ]
237     else:
238         G = vizing(C, G, X, arestas, N, M)
239         #G = colorir_arestas(arestas, vertices, gerador, P)
240         #print(G)
241
242         ca = [
243             {
244             'data': {
245                     'source': str(source),
```

```
                    'target': str(target),
                    #'color': str(cores[G[source][target] - 1])
                    'color': str(cores[G[(source, target)] - 1])
                }
            }
            for source, target in arestas
        ]


    cv = []
    raio = 300
    cx, cy = 400, 400  # centro do grafo
    N = len(vertices)
    for i, v in enumerate(vertices):
        ang = 2 * pi * i / N
        x = cx + raio * cos(ang)
        y = cy + raio * sin(ang)
        cv.append({
            'data': {'id': str(i+1), 'label': str(v)},
            'position': {'x': x, 'y': y}
        })

    elements = cv + ca

    app.layout = dbc.Container([
    dbc.Navbar(
        dbc.Container([
            dbc.Row([
                dbc.Col([
                    html.Div("Grafo Hlp", className="navbar-brand
                       mx-auto")
                ])
            ], className="w-100")
        ], fluid=True),
        color="primary",
        dark=True,
        sticky="top",
        className="w-100"
    ),
    dcc.Store(id='clicked-color-index', data=None),
    dbc.Row([
        dbc.Col([
            cyto.Cytoscape(
                id='cytoscape-layout-1',
                elements=elements,
                style={'width': '100%', 'height': '900px'},
                layout={'name': 'preset'},
                stylesheet=[
                        {
                            'selector': 'edge',
                            'style': {
                                'line-color': 'data(color)'
```

```
297                                 }
298                             }
299                         ]
300                     )
301                 ],width=8),# Grafo ocupa 8 colunas
302                 dbc.Col([
303                     dbc.Row([
304                         dbc.Col([
305                             html.Label('L:', htmlFor='input-L'),
306                             dcc.Input(id='input-L', type='number', value=L
                                , min=3, max=6, style={'marginBottom': '10
                                px'}),
307                         ], width=6),# Input L ocupa metade da coluna
308                         dbc.Col([
309                             html.Label('P:', htmlFor='input-P'),
310                             dcc.Input(id='input-P', type='number', value=P
                                , min=3, max=6, style={'marginBottom': '10
                                px'}),
311                         ], width=6),# Input P ocupa metade da coluna
312                     ]),
313                     html.Hr(),
314                     html.Div(id='generation-time-display', style={'
                        marginTop': '10px', 'fontWeight': 'bold'}),
315                     html.Div(id='graph-stats', style={'marginTop': '10px'
                        }),
316                     html.Hr(),
317                     html.Div(id='legenda-cores', style={'marginTop': '20px
                        '})
318                 ], width=4)# Inputs e legenda ocupam 4 colunas
319             ]),
320             ], fluid=True)
321
322             @app.callback(
323                 [Output('cytoscape-layout-1', 'elements'),
324                 Output('legenda-cores', 'children'),
325                 Output('generation-time-display', 'children'),
326                 Output('graph-stats', 'children')
327                 ],
328                 [Input('input-L', 'value'), Input('input-P', 'value')]
329             )
330             def update_graph(L, P):
331                 global used_colors
332
333                 start = time.time()
334
335                 gerador = conjunto_gerador(L, P)
336                 vertices = gerar_vertices_Hlp(L, P)
337                 arestas = gerar_arestas_Hlp(vertices, gerador, P)
338                 N = len(vertices)
339                 M = len(arestas)
340
341                 if L <= 5 and P <= 5 or L < 3 and P < 3:
```

```
342             grau = (L * (L - 1)) + 1
343             cores = []
344             C, G = clear(N)
345             X = [0] * (N + 1)
346             for i in range(grau):
347                 cores.append('#%06X' % randint(0, 0xFFFFFF))
348
349             if P % 2 == 0:
350
351                 G = colorir_arestas(arestas, vertices, gerador, P)
352                 ca = [
353                     {
354                         'data': {
355                             'source': str(source),
356                             'target': str(target),
357                             'color': str(cores[G[(source, target)]
                                    - 1])
358                         }
359                     }
360                     for source, target in arestas
361                 ]
362             else:
363                 G = vizing(C, G, X, arestas, N, M)
364                 ca = [
365                     {
366                         'data': {
367                             'source': str(source),
368                             'target': str(target),
369                             #'color': str(cores[G[source][target]
                                - 1])
370                             'color': str(cores[G[(source, target)]
                                    - 1])
371                         }
372                     }
373                     for source, target in arestas
374                 ]
375             cv = []
376             N = len(vertices)
377             raio = max(200, 30 * N)
378             cx, cy = 400, 400  # centro do grafo
379             N = len(vertices)
380             for i, v in enumerate(vertices):
381                 ang = 2 * pi * i / N
382                 x = cx + raio * cos(ang)
383                 y = cy + raio * sin(ang)
384                 cv.append({
385                     'data': {'id': str(i+1), 'label': str(v)},
386                     'position': {'x': x, 'y': y}
387                 })
388             elements = cv + ca
389
390
```

```
391             # Coletar apenas as cores usadas nas arestas
392
393             used_colors = set()
394             for edge in ca:
395                 used_colors.add(edge['data']['color'])
396
397             # Atualizacao da legenda de cores com apenas as cores
                    usadas
398             legenda_cores = [
399                 html.Div([
400                     html.Span(style={'background-color': color, '
                            display': 'inline-block', 'width': '20px',
                            'height': '20px', 'margin-right': '10px'}),
401                     html.Span(f'Cor {i + 1}')
402                 ], style={'cursor': 'pointer'}, id={'type': 'color
                        -button', 'index': i})
403                 for i, color in enumerate(used_colors)
404             ]
405         else:
406             elements = []
407             legenda_cores = [html.Div(" Grafo muito grande para
                    renderizar no navegador.")]
408
409         end = time.time()
410         generation_time = f" Tempo de geracao (Python): {(end -
                start)*1000:.2f} ms (L={L}, P={P})"
411         graph_stats = f" Vertices: {N}, Arestas: {M}, Grau: {(L *
                (L-1)) + 1}"
412
413         return elements, legenda_cores, generation_time,
                graph_stats
414
415
416     @app.callback(
417     Output('cytoscape-layout-1', 'stylesheet'),
418     [Input('cytoscape-layout-1', 'tapNodeData'),
419      Input({'type': 'color-button', 'index': ALL}, 'n_clicks')],
420     [State('cytoscape-layout-1', 'elements')]
421 )
422     def highlight_edges(node_data, n_clicks_list, elements):
423         global used_colors
424         ctx = dash.callback_context
425         triggered_id = ctx.triggered[0]['prop_id'] if ctx.
                triggered else None
426
427         # Reseta o CSS para o padrao
428         stylesheet = [
429             {
430                 'selector': 'edge',
431                 'style': {
432                     'line-color': 'data(color)',
433                     'width': 2  # Default width for edges
```

```
434                 }
435             },
436             {
437                 'selector': 'node',
438                 'style': {
439                     'background-color': '#007bff',
440                     'label': 'data(label)'
441                 }
442             }
443         ]
444 
445         # Click no Vertice
446         if node_data and triggered_id == 'cytoscape-layout-1.
               tapNodeData':
447             connected_edges = [
448                 {'selector': f'edge[source = "{node_data["id"]}"],
                       edge[target = "{node_data["id"]}"]',
449                  'style': {
450                      'width': 5,
451                      'z-index': 999
452                  }}
453             ]
454             stylesheet.extend(connected_edges)
455             return stylesheet
456 
457         # Click na Legenda de Cores
458         if triggered_id and 'color-button' in triggered_id:
459             try:
460                 triggered_json = json.loads(triggered_id.split('.'
                       )[0])
461                 clicked_index = triggered_json['index']
462                 clicked_color = list(used_colors)[clicked_index]
463 
464                 color_edges = [
465                     {'selector': f'edge[color = "{clicked_color}"]
                           ',
466                      'style': {
467                          'width': 5,
468                          'z-index': 999
469                      }}
470                 ]
471                 stylesheet.extend(color_edges)
472             except (json.JSONDecodeError, IndexError, KeyError):
473                 pass
474 
475         return stylesheet
476 
477     app.run_server(debug=True)
478 
479 if __name__ == "__main__":
480     L = 3
481     P = 3
```

```
482     gerador = conjunto_gerador(L, P)
483     vertices = gerar_vertices_Hlp(L, P)
484     arestas = gerar_arestas_Hlp(vertices, gerador, P)
485     criar_Hlp(vertices, arestas, gerador, P, L)
```

Listing A.1 – Implementação em Python para geração e visualização do grafo $H_{l,p}$