CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA

Total Color: uma aplicação web para coloração total em grafos

Renan Carvalho Pinheiro da Silva

Professores Orientadores:
Mayara Midori Omai, M.Sc.
Mauro Nigro Alves Junior, Dr.

Rio de Janeiro,
Fevereiro de 2026

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA
CELSO SUCKOW DA FONSECA

Total Color: uma aplicação web para coloração total em grafos

Renan Carvalho Pinheiro da Silva

Projeto final apresentado em cumprimento às normas do Departamento de Educação Superior do Centro Federal de Educação Tecnológica Celso Suckow da Fonseca, CEFET/RJ, como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação.

Professores Orientadores:
Mayara Midori Omai, M.Sc.
Mauro Nigro Alves Junior, Dr.

Rio de Janeiro,
Fevereiro de 2026

DEDICATÓRIA

A Deus e à minha família, que estiveram comigo em todos os momentos, nos bons e nos difíceis.

"Eis que estou convosco todos os dias, até o fim do mundo."
São Mateus 28, 20b

AGRADECIMENTOS

Aos meus professores, pela dedicação e pelos ensinamentos que possibilitaram minha formação e contribuíram para que eu me tornasse o profissional que sou hoje.

"Se vi mais longe, foi por estar sobre os ombros de gigantes."
Isaac Newton

RESUMO

A visualização de grafos desempenha um papel importante no ensino e na compreensão de conceitos da Teoria dos Grafos, especialmente em problemas de coloração, que apresentam elevada complexidade teórica e computacional. Entre esses problemas, destaca-se a coloração total de grafos, que consiste na atribuição de cores a vértices e arestas de modo que elementos adjacentes ou incidentes recebam cores distintas. Apesar de sua relevância teórica e de sua relação com a Conjectura da Coloração Total, observa-se a escassez de ferramentas educacionais interativas voltadas especificamente para esse tema. Neste contexto, este trabalho apresenta o Total Color, uma aplicação web desenvolvida com finalidade acadêmica, cujo objetivo é permitir a visualização da coloração total em classes específicas de grafos, bem como promover a aprendizagem ativa por meio da interação e experimentação do usuário. A aplicação contempla as classes dos grafos caminho, ciclo e completos, incorporando resultados teóricos conhecidos sobre seus respectivos números cromáticos totais. Além disso, o sistema oferece funcionalidades para geração automática de grafos dessas classes e um modo livre para construção e manipulação manual de grafos. Como contribuição, o Total Color integra conceitos teóricos da coloração total a uma interface visual interativa, auxiliando no processo de ensino-aprendizagem e servindo como ferramenta de apoio para estudantes e pesquisadores da área de Teoria dos Grafos.

Palavras-chave: Teoria dos Grafos; Coloração Total; Visualização de Grafos; Aplicações Web; Aprendizagem Interativa.

ABSTRACT

Graph visualization plays a fundamental role in the teaching and understanding of Graph Theory concepts, especially in graph coloring problems, which involve significant theoretical and computational complexity. Among these problems, total graph coloring stands out, consisting of assigning colors to both vertices and edges in such a way that adjacent or incident elements receive distinct colors. Despite its theoretical relevance and its connection to the Total Coloring Conjecture, there is a noticeable lack of interactive educational tools specifically focused on this topic. In this context, this work presents Total Color, a web-based application developed for academic purposes, whose main goal is to enable the visualization of total coloring in specific classes of graphs, as well as to promote active learning through user interaction and experimentation. The application covers the classes of path graphs, cycle graphs, and complete graphs, incorporating well-established theoretical results regarding their total chromatic numbers. In addition, the system provides functionalities for the automatic generation of graphs from these classes and a free mode for manual graph construction and manipulation. As a contribution, Total Color bridges theoretical aspects of total graph coloring with an interactive visual interface, supporting the teaching and learning process and serving as a complementary tool for students and researchers in the field of Graph Theory.

Keywords: Graph Theory; Total Coloring; Graph Visualization; Web Applications; Interactive Learning.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

HOG	House Of Graphs	2, 22, 24
HTTP	Hypertext Transfer Protocol	21
ISGCI	Information System On Graph Classes And Their Inclusions	2, 23, 25, 28
STEM	Science, Technology, Engineering, And Mathematics	1
UI	User Interface	20, 31

Capítulo 1

Introdução

A visualização de grafos é importante para a compreensão do relacionamento de ideias ou objetos, identificando padrões e comportamentos. Enquanto as pessoas expressam suas ideias nas salas de reunião do trabalho ou em uma sala de aula de uma universidade, consciente ou inconscientemente, utilizam técnicas de grafos, rotulam os relacionamentos entre objetos e criam diagramas para explicar suas ideias aos outros. Expressar um problema definido no mundo real ou um modelo das relações que compõem um sistema está entre os usos mais valiosos e essenciais dos grafos [Das and Soylu, 2023].

Além da compreensão de relacionamento entre objetos através da aplicabilidade do dia a dia, a visualização de grafos possui aplicações técnicas importantes, como em redes de computadores, bioinformática, ciência da computação, ciências naturais, aprendizado de máquina, engenharia de software, bancos de dados, design de sites e em interfaces visuais para outros domínios técnicos [Chen et al., 2019].

Criar explicações visuais sobre fenômenos nas áreas de Ciência, Tecnologia, Engenharia e Matemática (do inglês: Science, Technology, Engineering, and Mathematics (STEM)) melhora a aprendizagem, pois podem mostrar diretamente as partes e processos de sistemas complexos. Além disso, fornecem verificações quanto à completude e coerência, ou seja, verificação de que todos os elementos necessários do sistema estão representados e funcionam juntos adequadamente para produzir os resultados dos processos. Explicações visuais também oferecem uma referência concreta para fazer e checar inferências sobre comportamento, a causalidade e a função do sistema [Bobek and Tversky, 2016].

Assim como em pesquisas científicas em que o processo visual é utilizado como um objeto epistêmico, ou seja, um instrumento cognitivo para o entendimento do conteúdo e resultados do trabalho, o processo visual é importante para o ensino, pois cria um ambiente de engajamento para o método científico, como a observação e o procedimento experimental [Evagorou et al., 2015].

No contexto do ensino de matemática, Santos [2014] destaca a importância do acesso ao conhecimento matemático por intermédio dos processos de visualização. Descreve, como exem-

plo, a apresentação do conceito de quadrado perfeito. Além de apenas falar que o número quadrado perfeito é um número natural cuja raiz quadrada resulta em outro número natural, pode-se construir uma representação geométrica para a compreensão do conceito. Como na Figura 1, em que o número 16 é quadrado perfeito pois, com 16 quadrados menores dispostos em quatro linhas e quatro colunas, obtém-se um quadrado maior; no entanto, o número 5 não é quadrado perfeito, pois, com 5 quadrados menores, não é possível obter um quadrado maior.

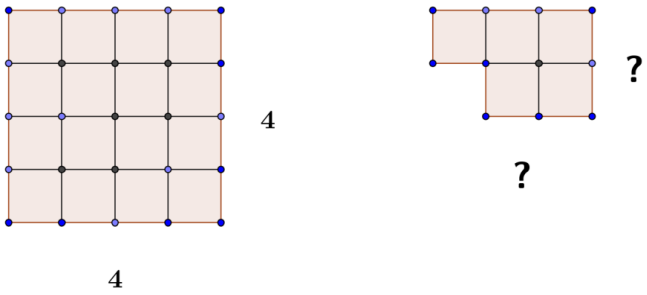


Figura 1: Representação geométrica do número quadrado perfeito por Santos [2014].

Na área de Teoria dos Grafos, existem aplicações que reúnem dados estruturados sobre grafos e suas propriedades, como o House of Graphs (HoG) e o Information System on Graph Classes and their Inclusions (ISGCI), bem como ferramentas que promovem a aprendizagem interativa, como o VisuAlgo e o Graph Online. No entanto, no que se refere à Coloração de Grafos, especialmente à coloração total, há poucas aplicações que disponibilizam informações relevantes. Mesmo o HoG, uma plataforma amplamente reconhecida pela comunidade de Teoria dos Grafos, não contempla conteúdo específico relacionado à coloração total. Além disso, no contexto de ambientes com aprendizado interativo, observa-se a ausência de trabalhos voltados especificamente à coloração total.

Logo, a solução proposta neste trabalho visa o desenvolvimento de uma aplicação web para a visualização da coloração total em classes de grafos específicas, com finalidade acadêmica. Essa aplicação, denominada *Total Color*, possui dois objetivos principais: (i) permitir a visualização da coloração total em determinadas classes de grafos[1]; e (ii) promover a aprendizagem ativa por meio da interação e experimentação do usuário.

O *Total Color* é implementado com tecnologias *client-side*, possibilitando sua execução diretamente em navegadores web e sua disponibilização por meio de serviços de hospedagem. A aplicação conta ainda com uma *interface* responsiva, o que permite sua utilização em diferentes

[1]Nesse trabalho, são abordadas as classes dos caminhos, ciclos e completos, definidas no Capítulo 2.

dispositivos, incluindo dispositivos móveis.

No que se refere às funcionalidades, o sistema contempla as classes de grafos caminhos, ciclos e completos. Para cada uma dessas classes, o usuário pode visualizar o grafo correspondente, bem como o respectivo número cromático total, além de acompanhar, de forma visual e progressiva, o processo de coloração realizado pelo algoritmo. Adicionalmente, a aplicação permite que o usuário atribua manualmente cores aos vértices e arestas do grafo, possibilitando a verificação da corretude da coloração proposta.

Além dessas funcionalidades, o *Total Color* disponibiliza um modo livre, no qual o usuário pode realizar o *upload* de um grafo no formato *.g6* ou fornecer diretamente sua matriz de adjacência. Como contribuição adicional deste trabalho, são apresentadas as funcionalidades de um pacote publicado no *npm*, denominado *graph6*, desenvolvido no contexto desta pesquisa, que realiza a conversão entre matrizes de adjacência e o formato *.g6*.

Com base na proposta apresentada, este trabalho está organizado da seguinte forma: no Capítulo 2, são introduzidos os conceitos teóricos fundamentais, incluindo Teoria dos Grafos, Coloração de Grafos, demonstrações relacionadas à coloração total nas classes abordadas neste trabalho e Aplicações Web. O Capítulo 3 apresenta aplicações com escopo semelhante ao do *Total Color*, sendo essa relação sintetizada ao final por meio de uma tabela comparativa. Em seguida, o Capítulo 4 detalha as funcionalidades do *Total Color*, as tecnologias utilizadas, a estrutura da aplicação, bem como os algoritmos empregados na coloração total. Por fim, o Capítulo 5 apresenta as conclusões obtidas com o desenvolvimento da aplicação, destacando as funcionalidades implementadas em atendimento ao objetivo principal do trabalho, bem como as funcionalidades adicionais e as sugestões de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta conceitos da Teoria dos Grafos relevantes ao escopo deste trabalho, com ênfase em problemas de coloração, bem como introduz fundamentos sobre Aplicações Web. A Seção 2.1 apresenta conceitos básicos da Teoria dos Grafos, enquanto a Seção 2.2 trata do problema de coloração, com foco em colorações clássicas e na coloração total de grafos caminhos, ciclos e completos. A Seção 2.3 introduz conceitos fundamentais sobre aplicações web. Os conceitos de Teoria dos Grafos baseiam-se em Diestel [2005]; Bondy and Murty [1982]; West et al. [2001], enquanto os fundamentos de aplicações web seguem Connolly et al. [2015].

2.1 Teoria dos Grafos

Na região de Königsberg, na Prússia, atual Kaliningrado na Rússia, havia uma ilha chamada *Kneiphof*. Essa ilha, representada na Figura 2 com a letra A, era rodeada por um rio, formando assim quatro regiões: A, B, C e D, conectadas por sete pontes: a, b, c, d, e, f e g. Um problema conhecido acerca dessas pontes era o questionamento sobre a possibilidade de realizar um passeio que cruzasse cada ponte exatamente uma vez.

O matemático Leonhard Euler, então, em 1736, decidiu solucionar esse problema no artigo *Solutio problematis ad geometriam situs pertinentis* (Solução de um problema na geometria de posição) que ficou conhecido como **Problema das Pontes de Königsberg**. Como resultado do estudo, Euler chegou à conclusão de que não era possível realizar esse passeio pelas sete pontes de Königsberg.

No entanto, demonstrou que esse tipo de trajeto seria possível em situações semelhantes, desde que houvesse exatamente zero ou duas regiões com número ímpar de pontes. No caso de duas regiões com número ímpar de pontes, o passeio deve, necessariamente, começar em uma delas [Euler, 1736][1].

[1] Artigo em inglês disponível em https://www.cantab.net/users/michael.behrend/repubs/maze_maths/pages/euler_en.html. Acessado em 08/06/2025.

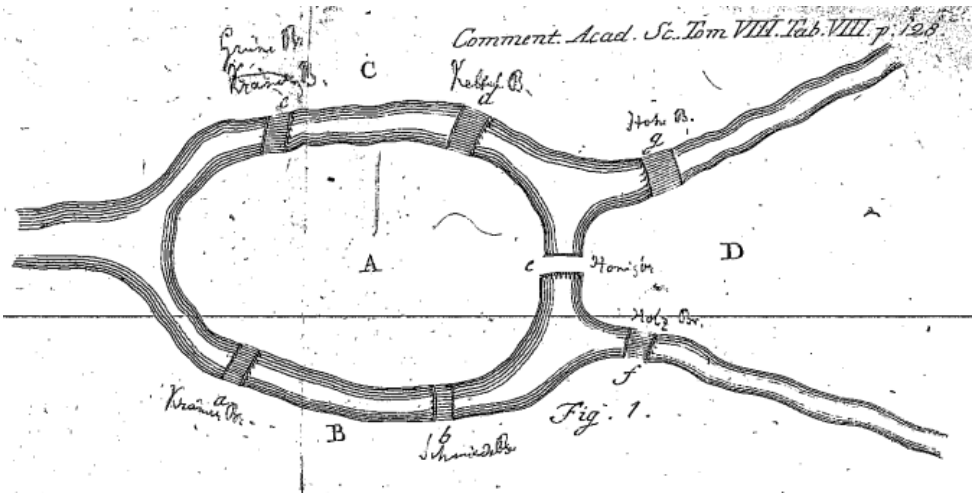


Figura 2: Ilustração das 7 pontes de Königsberg [Euler, 1736].

O Problema das Pontes de Königsberg marca a origem do desenvolvimento da Teoria dos Grafos. Os elementos do problema foram modelados com base nos conceitos da geometria de posição, cujo foco está no relacionamento e no posicionamento dos objetos no espaço, em vez da realização de cálculos métricos. Na modelagem desse problema, as regiões são representadas por vértices e as pontes que conectam essas regiões são representadas por arestas entre os respectivos vértices. Assim como neste caso, diversos outros problemas podem ser modelados como grafos.

Formalmente, um **grafo** é uma tripla ordenada $G = (V, E, \psi)$ em que V é um conjunto não vazio de vértices, E um conjunto de arestas, e ψ uma função que associa cada aresta a um par de vértices não necessariamente distintos. Denotaremos os conjuntos V e E de um grafo G por $V(G)$ e $E(G)$, respectivamente.

A Figura 3 apresenta um grafo G com $V(G) = \{A,B,C,D\}$, $E(G) = \{a,b,c,d,e,f,g\}$ e as seguintes associações: $\psi(a) = \{A,B\}, \psi(b) = \{A,B\}, \psi(c) = \{A,C\}, \psi(d) = \{A,C\}, \psi(e) = \{A,D\}, \psi(f) = \{B,D\}$, e $\psi(g) = \{C,D\}$. Observe que este grafo modela o Problema das Pontes de Königsberg, em que os vértices são as regiões e as arestas são as pontes que os conectam.

Um **grafo não vazio** é um grafo $G = (V,E)$ tal que $V(G) \neq \emptyset$. Seja G um grafo não vazio, uma aresta associada aos vértices u e v é denotada por uv. Se $uv \in E(G)$, então u e v são chamados **vértices adjacentes**. Além disso, u e v são denominados **vértices terminais** de uv. Se $uv, vz \in E(G)$, então uv e vz são chamadas **arestas adjacentes**. Como também, dizemos que uv **incide** em u e v.

Um **laço** é uma aresta cujos vértices terminais coincidem. Arestas distintas associadas ao mesmo par de vértices terminais são denominadas **arestas paralelas**. Um **multigrafo** é um

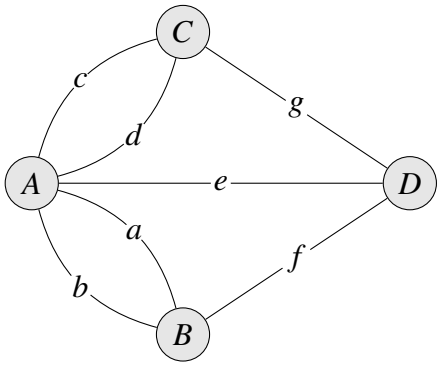
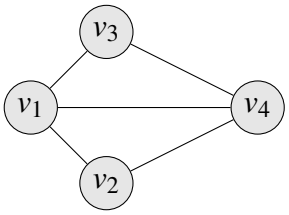


Figura 3: Problema das Pontes de Königsberg modelado como um grafo.

grafo que pode conter tanto laços quanto arestas paralelas. A Figura 3 apresenta um exemplo de multigrafo o qual modela o Problema das Pontes de Königsberg.

Neste trabalho, nos restringimos à análise de **grafos simples**, que são grafos que não possuem arestas paralelas nem laços. Podemos então simplificar a notação de um **grafo** como $G = (V, E)$, em que os elementos de E são subconjuntos com dois elementos de V. Para evitar ambiguidades, $V \cap E = \emptyset$.

A quantidade de vértices de um grafo G, denotada por $|V(G)|$, é chamada de **ordem do grafo**, assim como a quantidade de arestas de um grafo G, denotada por $|E(G)|$, é chamada de **tamanho do grafo**. A Figura 4(a) apresenta um exemplo de um grafo simples G, com $V(G) = \{v_1, v_2, v_3, v_4\}$ e $E(G) = \{v_1v_2, v_1v_3, v_1v_4, v_2v_4, v_3v_4\}$, tendo $|V(G)| = 4$ e $|E(G)| = 5$.



(a)

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

(b)

Figura 4: Em (a) um grafo simples, e em (b) sua matriz de adjacência.

Um grafo G com $V(G) = \{v_1, v_2, \ldots, v_n\}$ pode ser representado computacionalmente por meio de uma **matriz de adjacência**, que é uma matriz $A(G) = [a_{ij}]$ de ordem $n \times n$ em que $a_{i,j} = 1$ se v_i e v_j são adjacentes em G, e $a_{i,j} = 0$, caso contrário. A Figura 4(b) apresenta a matriz de adjacência do grafo representado na Figura 4(a).

Além da sua estrutura, um grafo possui parâmetros que o caracterizam, como por exemplo, as propriedades relacionadas às suas arestas. O **grau** de um vértice v, denotado por $d(v)$, corresponde ao número de arestas incidentes em v. O **grau máximo** de um grafo G, denotado por

$\Delta(G)$, é o maior grau entre os graus dos vértices de G, ou seja, $\Delta(G) = max\{d(v) \mid v \in V\}$. Um grafo G em que todos os vértices têm o mesmo grau k, ou seja, $\forall v \in V(G),\ d(v) = k$, é chamado de **k-regular**.

A Figura 5 apresenta um grafo G, em que $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$ e $E(G) = \{v_1v_2, v_1v_3, v_1v_4, v_1v_5\}$, no qual $d(v_1) = 4$, $d(v_2) = d(v_3) = d(v_4) = d(v_5) = 1$ e $\Delta(G) = 4$. A Figura 6 apresenta um grafo G, com $V(G) = \{v_1, v_2, v_3\}$ e $E(G) = \{v_1v_2, v_1v_3, v_2v_3\}$, que é um grafo 2-regular, isto é, um grafo em que todo vértice tem grau 2.

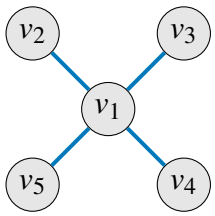


Figura 5: Um grafo com $\Delta(G) = 4$.

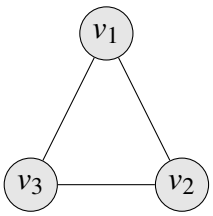


Figura 6: Um grafo 2-regular.

Um **caminho** em G é uma sequência de vértices distintos $v_0, v_1, \ldots, v_k$ de tal forma que para todo $i = 0, 1, \ldots, k-1$, o par $v_iv_{i+1} \in E(G)$. Um grafo é dito **conexo** se existe caminho entre qualquer par de vértices distintos. Os grafos das Figuras 5 e 6 são exemplos de grafo conexo; além disso como exemplo de caminho, temos o caminho $v_1v_2v_3$ na Figura 6.

Um **grafo completo**, denotado por K_n, é um grafo com n vértices no qual existe uma aresta entre quaisquer vértices distintos. Observe que K_n é $(n-1)$-regular. A Figura 7 ilustra o grafo completo K_5 que é um grafo 4-regular.

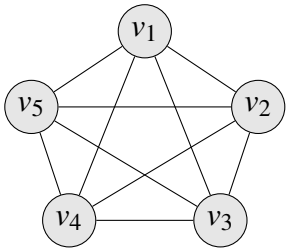


Figura 7: Grafo K_5.

Um **grafo caminho**, denotado por P_n, é um grafo com n vértices em que $V(P_n) = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$ e $E(P_n) = \{v_0v_1, v_1v_2, \ldots, v_{n-2}v_{n-1}\}$. Uma notação para um grafo caminho pode ser dada pela sequência de seus elementos $P_n = v_0, v_0v_1, \ldots, v_{n-2}v_{n-1}, v_{n-1}$, onde $v_0, v_1, \ldots, v_{n-1}$ são os vértices do caminho.

Um **grafo ciclo**, denotado por C_n, é um grafo com n vértices em que $V(C_n) = \{v_0, v_1, v_2, \ldots, v_{n-1}\}$ e $E(C_n) = \{v_0v_1, v_1v_2, \ldots, v_{n-2}v_{n-1}, v_{n-1}v_0\}$. Uma notação para um grafo ciclo pode ser

dada pela sequência de seus elementos $C_n = v_0, v_0v_1, \ldots, v_{n-1}, v_{n-1}v_0$, onde $v_0, v_1, \ldots, v_{n-1}$ são os vértices do ciclo. O comprimento de C_n corresponde ao seu tamanho e um grafo ciclo é classificado como par ou ímpar de acordo com a paridade do seu comprimento.

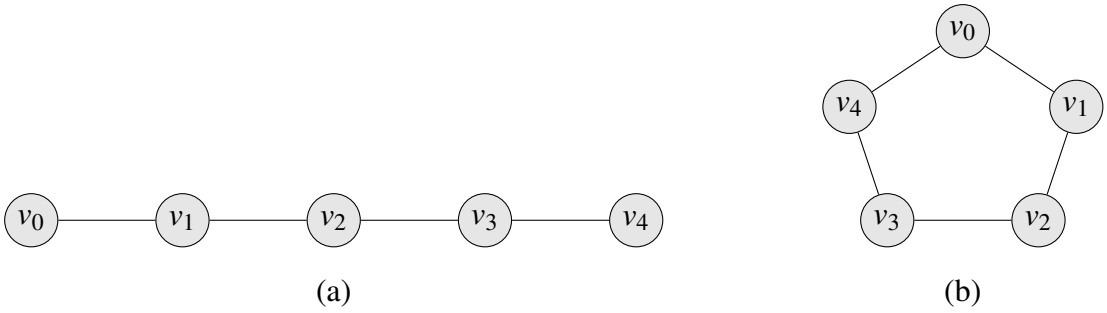


Figura 8: Em (a) um P_5, e em (b) um C_5.

A Figura 8(a) ilustra um grafo caminho P_5, com $V(P_5) = \{v_0, v_1, v_2, v_3, v_4\}$ e $E(P_5) = \{v_0v_1, v_1v_2, v_2v_3, v_3v_4\}$ e a Figura 8(b) ilustra um grafo ciclo ímpar C_5, com $V(C_5) = \{v_0, v_1, v_2, v_3, v_4\}$ e $E(C_5) = \{v_0v_1, v_1v_2, v_2v_3, v_3v_4, v_4v_0\}$.

Existem conjuntos que representam a independência entre os elementos de um grafo, seja entre vértices, entre arestas ou entre ambos. Esses conjuntos são fundamentais para a demonstração de resultados de coloração apresentados nesse trabalho.

Um conjunto $S \subseteq V(G)$ é chamado de **conjunto independente** em G se nenhum par de vértices em S é adjacente em G. Um conjunto independente S é dito **maximal** se não é possível adicionar a S nenhum outro vértice de G sem que o conjunto deixe de ser independente. Além disso, um conjunto independente S é dito **máximo** se não há um conjunto independente $S' \subseteq V(G)$ tal que $|S'| > |S|$. O **número de independência** de G, denotado por $\alpha(G)$, é o tamanho de um conjunto independente máximo de G.

Dado um grafo G, com $V(G) = \{v_1, v_2, v_3, v_4\}$ e $E(G) = \{v_1v_2, v_1v_3, v_1v_4, v_3v_4\}$, o conjunto $S_1 = \{v_1\}$ é um conjunto independente maximal, conforme ilustrado na Figura 9(a), e $S_2 = \{v_2, v_4\}$ é um conjunto independente máximo com $\alpha(G) = 2$, conforme ilustrado na Figura 9(b).

Um conjunto $M \subseteq E(G)$ é chamado de **emparelhamento** em G se não existem arestas adjacentes em M. Os vértices terminais de uma aresta pertencente a um emparelhamento são ditos **saturados**. Um emparelhamento M é dito **maximal** caso não seja possível incluir arestas de G em M sem que M deixe de ser um emparelhamento. Um emparelhamento M é dito **máximo** se não há um emparelhamento $M' \subseteq E(G)$ tal que $|M'| > |M|$. O **número de independência de**

(a) (b)

Figura 9: Em (a) um conjunto independente maximal e em (b) um conjunto independente máximo.

arestas de G, denotado por $\alpha'(G)$, é o tamanho de um emparelhamento máximo de G.

Dado um grafo G, com $V(G) = \{v_1, v_2, v_3, v_4\}$ e $E(G) = \{v_1v_2, v_1v_3, v_3v_4\}$, o conjunto $M_1 = \{v_1v_3\}$, por inspeção, é um emparelhamento maximal, conforme ilustrado na Figura 10(a), e $M_2 = \{v_1v_2, v_3v_4\}$ um emparelhamento máximo, conforme ilustrado na Figura 10(b). Como não existe um conjunto $M' \subseteq E(G)$ tal que $|M'| \geq |M_2|$, então $\alpha'(G) = |M_2| = 2$.

(a) (b)

Figura 10: Em (a) um emparelhamento maximal e em (b) um emparelhamento máximo.

Um conjunto $T \subseteq V(G) \cup E(G)$ é chamado de **conjunto independente total** se existem um emparelhamento $M \subseteq E(G)$ e/ou um conjunto independente $S \subseteq V(G)$ tais que $T = M \cup S$ e nenhum vértice de S é saturado por M. Um conjunto independente total T é dito **maximal** se não é possível adicionar nenhum elemento de G em T tal que T continue sendo um conjunto independente total.

Um conjunto independente total T é dito **máximo** se não há um conjunto independente total $T' \subseteq V(G) \cup E(G)$ tal que $|T'| > |T|$. O **número de independência total** de G, denotado por $\alpha''(G)$, é o tamanho do maior conjunto independente total em G. Na Figura 11 um exemplo de conjunto independente total $T = \{v_1v_2, v_3v_4, v_5v_6, v_7\}$, dado que $M = \{v_1v_2, v_3v_4, v_5v_6\}$ e $S = \{v_7\}$.

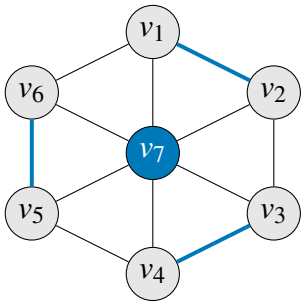


Figura 11: Exemplo de conjunto independente total.

2.2 Coloração de grafos

A coloração de grafos possui origem no **Problema das Quatro Cores**, introduzido no ano de 1852, quando Francis Guthrie ao tentar colorir regiões da Inglaterra, propôs ao seu irmão Frederick, estudante universitário de Matemática em Cambridge, que regiões de um mapa poderiam ser coloridas com no máximo 4 cores de modo que regiões fronteiriças possuíssem cores diferentes. Posteriormente, o tema foi amplamente investigado pela comunidade científica [Cayley, 1879].

O Problema das Quatro Cores pode ser modelado como um grafo, em que dado um mapa, cada região é representada por um vértice, e existe uma aresta entre dois vértices caso as regiões representadas por eles compartilhem uma fronteira. Cada cor atribuída é representada por um número inteiro no vértice. A Figura 12 apresenta um exemplo do Problema das Quatro Cores. Na Figura 12(a) uma coloração do mapa da região Sudeste do Brasil, e na Figura 12(b) um grafo correspondente ao mapa da Figura 12(a), tendo as seguintes atribuições de cores: verde (1) para RJ, lilás (2) para SP, azul (3) para MG, e marrom (4) para ES.

(a) (b)

Figura 12: Em (a) um exemplo do Problema das Quatro Cores com uma coloração do mapa de região Sudeste do Brasil, e em (b) um modelo em grafo dessa coloração.

Uma **coloração de vértices** de um grafo G é uma função $c : V \to S$ que mapeia os vértices do grafo a um conjunto de cores S em que as cores são representadas por números inteiros positivos, de forma que $c(u) \neq c(v)$, para todo par de vértices u e v adjacentes. Uma **k-coloração de vértices** é uma coloração de vértices na qual são utilizadas $|S| = k$ cores. O **número cromático** de G, denotado por $\chi(G)$, é o menor k tal que G possui uma k-coloração de vértices.

A Figura 12(b) apresenta uma 4-coloração de vértices do grafo correspondente ao mapa da região Sudeste do Brasil, enquanto que a Figura 13(b) exibe uma 3-coloração de vértices do mesmo grafo. Os vértices que representam os estados RJ, SP e MG são adjacentes entre si, o que exige o uso de cores distintas para cada um, totalizando três cores. O vértice que representa o estado ES, por sua vez, não requer uma nova cor, pois pode reutilizar a cor atribuída ao vértice correspondente a SP, que não é seu adjacente e, ao mesmo tempo, possui cor diferente dos vértices adjacentes a ES (RJ e MG). Assim, conclui-se que $\chi(G) = 3$.

(a) (b)

Figura 13: Em (a) um mapa de região Sudeste do Brasil com 3 cores, e em (b) o grafo correspondente com uma 3-coloração total de vértices.

O **Teorema de Brooks** é um dos principais teoremas da coloração de vértices. Nele estabelece-se um limitante superior para o número cromático de um grafo conexo, relacionando-o ao grau máximo do grafo.

Teorema 1 (Teorema de Brooks [Brooks, 1941])**.** *Seja G um grafo conexo simples com grau máximo $\Delta(G)$, que não é um grafo completo nem um ciclo ímpar, então $\chi(G) \leq \Delta(G)$.*

Uma **coloração de arestas** de um grafo G é uma função $c\colon E \to S$ que mapeia arestas do grafo a um conjunto de cores S em que as cores são representadas por números inteiros positivos, tal que $c(e_i) \neq c(e_j)$, sendo e_i e e_j arestas adjacentes. Uma **k-coloração de arestas** é uma coloração de arestas na qual são utilizadas $|S| = k$ cores. O **índice cromático** de G, denotado por $\chi'(G)$, é o menor k tal que G possui uma k-coloração de arestas.

Um aspecto importante da coloração de arestas é seu limitante inferior natural. Isso ocorre pois, para colorir as arestas de um grafo G é necessário atribuir pelo menos $\Delta(G)$ cores distintas às arestas incidentes em um vértice de grau máximo $\Delta(G)$. Vizing [1964] demonstrou que $\chi'(G) \leq \Delta(G)+1$ dando origem ao **Teorema de Vizing** apresentado a seguir.

Teorema 2 (Teorema de Vizing [Vizing, 1964])**.** *Seja G um grafo simples com grau máximo $\Delta(G)$, então $\Delta(G) \leq \chi'(G) \leq \Delta(G)+1$.*

Como consequência do Teorema 2, grafos simples admitem a seguinte classificação quanto ao índice cromático: Classe 1, se $\chi'(G) = \Delta(G)$; e Classe 2, se $\chi'(G) = \Delta(G)+1$.

Uma **coloração total** de um grafo G é uma função $c : (V \cup E) \rightarrow S$ que mapeia vértices e arestas do grafo a um conjunto de cores S, em que as cores são representadas por números inteiros positivos, de modo que vértices adjacentes, arestas adjacentes e cada aresta e seus vértices incidentes recebam cores distintas. Uma **k-coloração total** é uma coloração total na qual são utilizadas $|S| = k$ cores. O **número cromático total** de G, denotado por $\chi''(G)$, é o menor k tal que G possui uma k-coloração total.

A Figura 14 apresenta diferentes colorações para um mesmo grafo G. Na Figura 14(a) uma 2-coloração de vértices de G, na Figura 14(b) uma 2-coloração de arestas de G, e na Figura 14(c) uma 4-coloração total de G.

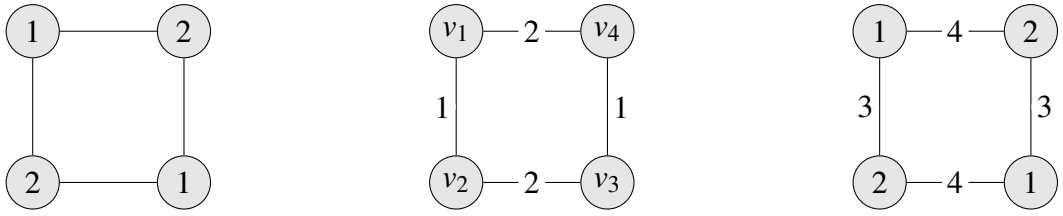


Figura 14: Exemplo dos tipos de colorações no grafo G. Em (a) uma 2-coloração de vértices, em (b) uma 2-coloração de arestas, e em (c) uma 4-coloração total.

Um limitante inferior natural para o número cromático total é $\chi''(G) \geq \Delta(G)+1$. Isso se deve ao fato de que para se obter uma coloração total de um grafo G, devem ser atribuídas ao menos $\Delta(G)$ cores distintas às arestas incidentes em um vértice de grau $\Delta(G)$, além de uma outra cor ao próprio vértice. De forma independente, Vizing [1964] e Behzad [1965] conjecturaram que $\chi''(G) \leq \Delta(G)+2$, dando origem à conjectura a seguir.

Conjectura 1 (Conjectura da Coloração Total (TCC) [Vizing, 1964; Behzad, 1965])**.** *Para todo grafo G simples, $\Delta(G)+1 \leq \chi''(G) \leq \Delta(G)+2$.*

De acordo com a Conjectura da Coloração Total, um grafo G é dito do Tipo 1 se $\chi''(G) = \Delta(G)+1$, e do Tipo 2 se $\chi''(G) = \Delta(G)+2$. Assim, provar que um grafo G é do Tipo 1 equivale a mostrar que G tem uma $(\Delta(G)+1)$-coloração total, enquanto provar que G é do Tipo 2 corresponde a mostrar que G tem uma $(\Delta(G)+2)$-coloração total e não tem uma $(\Delta(G)+1)$-coloração total.

A conjectura da coloração total foi verificada para casos restritos, como em grafos com $\Delta(G) \leq 5$, mas o problema geral permanece em aberto há mais de cinco décadas, ilustrando a dificuldade da coloração total; a conjectura não foi resolvida, por exemplo, para grafos regulares e grafos planares. Há algumas classes de grafos cujo número cromático total foi determinado, como os grafos ciclos, grafos completos, bipartidos completos e árvores [de Figueiredo, 2021].

Sob a ótica da complexidade computacional, problemas de decisão associados à coloração total podem ser analisados no contexto das classes P e NP. Enquanto problemas pertencentes à classe P admitem algoritmos determinísticos em tempo polinomial[2], problemas em NP caracterizam-se pela possibilidade de verificação eficiente de uma solução candidata. No caso geral, a coloração total de grafos está associada a problemas para os quais não se conhecem algoritmos determinísticos em tempo polinomial para a sua solução; entretanto, qualquer solução candidata pode ser verificada de forma eficiente.

Diante desse cenário, torna-se fundamental restringir o estudo da coloração total a classes específicas de grafos. Essa abordagem permite a obtenção de resultados mais precisos e aprofundados, tanto no que diz respeito ao número cromático total quanto ao comportamento estrutural das colorações. Nesse sentido, ao restringir o estudo a classes específicas de grafos, torna-se possível explorar propriedades estruturais que permitem resolver o problema de coloração total. A seguir, são apresentados os resultados contemplados na aplicação *Total Color*, os quais englobam as classes dos caminhos, ciclos e grafos completos.

Para $n \geq 3$, o grafo caminho P_n possui $\Delta(P_n) = 2$ e, pelo limitante inferior natural para o número cromático total, tem-se $\chi''(P_n) \geq \Delta(P_n)+1 = 3$. O Lema 1 mostra que existe uma 3-coloração total de P_n. Assim, conclui-se que $\chi''(P_n) = 3$ para $n \geq 3$.

No caso particular $n = 2$, o caminho P_2 consiste em uma única aresta cujos vértices são adjacentes e incidentes à mesma aresta, o que exige três cores distintas em uma coloração total. Portanto, também se tem $\chi''(P_2) = 3$.

[2]Algoritmos determinísticos que podem ser executados em um limite de tempo polinomial em relação ao tamanho da entrada são também chamados de algoritmos eficientes.

Lema 1. *(Campos [2006]). Seja P um caminho e sejam e_1 e e_2 dois elementos distintos de P, adjacentes ou incidentes, coloridos com duas cores distintas. Então, existe uma única 3-coloração total que preserva as cores de e_1 e e_2.* □

O Lema 1, além de auxiliar na determinação do número cromático total dos grafos caminhos, também é usado na demonstração do número cromático total dos grafos ciclos, conforme formalizado no Teorema 3.

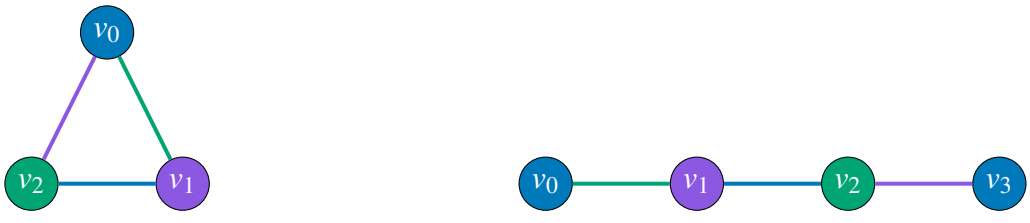
Teorema 3. *(Yap [1996]) Para qualquer inteiro $n \geq 3$,*

$$\chi''(C_n) = \begin{cases} 3, & \text{se } n \equiv 0 \ (mod\ 3) \\ 4, & \text{caso contrário} \end{cases}$$

Demonstração. Podemos compreender o grafo $C_n = v_0, v_0v_1, \ldots, v_{n-1}v_0, v_0$ como um $P_{n+1} = v_0, v_0v_1, \ldots, v_{n-1}v_n, v_n$, de tal forma que v_0 e v_n representam o mesmo vértice. Pelo Lema 1, o caminho P_{n+1} admite uma única 3-coloração total. Note que $|V(C_n) \cup E(C_n)| = 2n$. Os valores 0, 1, 2 e 3 correspondem, respectivamente, às cores azul, verde, lilás e cinza utilizadas.

Além disso, utiliza-se uma 3-coloração total em P_{n+1} partindo de v_0, seguido de v_0v_1, seguido de v_1, e assim sucessivamente. Os vértices e arestas de P_{n+1} recebem, nessa ordem, as cores 0, 1 e 2, repetindo-se esse padrão de forma periódica ao longo de todo o caminho.

Para o caso $n \equiv 0 \pmod 3$, tem-se $|V(C_n) \cup E(C_n)| \equiv 0 \pmod 3$. O padrão de coloração implica que $c(v_0) = c(v_n)$. Além disso, $c(v_0v_1) \neq c(v_{n-1}v_0)$, pois no padrão alternado da coloração total de P_{n+1} arestas consecutivas incidentes a um mesmo vértice recebem cores distintas entre si e também distintas da cor atribuída ao vértice. Portanto, como $\chi''(P_{n+1}) = 3$ e $\chi''(C_n) \geq \Delta(C_n) + 1 = 3$, conclui-se que $\chi''(C_n) = 3$. Como exemplo, apresentamos o grafo P_4 na Figura 15(b), a partir do qual se obtém a 3-coloração total de C_4 mostrada na Figura 15(a).



(a) (b)

Figura 15: Em (a) uma 3-coloração total de C_3, e em (b) sua representação como P_4 de tal forma que v_0 é o mesmo vértice que v_3.

Para o caso $n \equiv 1 \pmod 3$, tem-se $|V(C_n) \cup E(C_n)| \equiv 2 \pmod 3$. Como $\chi''(P_{n+1}) = 3$, considere uma 3-coloração total do C_4 a partir da 3-coloração total do P_{n+1} conforme o caso anterior. Pelo padrão periódico da 3-coloração total do caminho, quando $n \equiv 1 \pmod 3$ os vértices extremos v_0 e v_n recebem cores distintas, isto é, $c(v_0) \neq c(v_n)$.

Entretanto, na construção do ciclo C_n ocorre a identificação dos vértices v_0 e v_n. Assim, essa identificação exigiria que ambos recebessem a mesma cor, o que contradiz o padrão induzido pela coloração de P_{n+1}. Logo, a coloração total do caminho não pode induzir uma 3-coloração total válida em C_n.

Como exemplo, a Figura 16(a) apresenta uma tentativa de 3-coloração total de C_4 com conflitos, pois $c(v_0) = c(v_3)$ e $c(v_0v_1) = c(v_0v_3)$. Na Figura 16(b), a coloração do caminho P_5 evidencia que v_0 e v_4 recebem cores distintas, impossibilitando a identificação desses vértices no ciclo.

Dado que não é possível obter uma 3-coloração total, pode-se construir uma 4-coloração total realizando as seguintes modificações: $c(v_{n-2}) = 3$, $c(v_{n-2}v_{n-1}) = 1$, $c(v_{n-1}) = 2$ e $c(v_{n-1}v_0) = 3$. A Figura 16(c) ilustra uma 4-coloração total de C_4 após essas modificações.

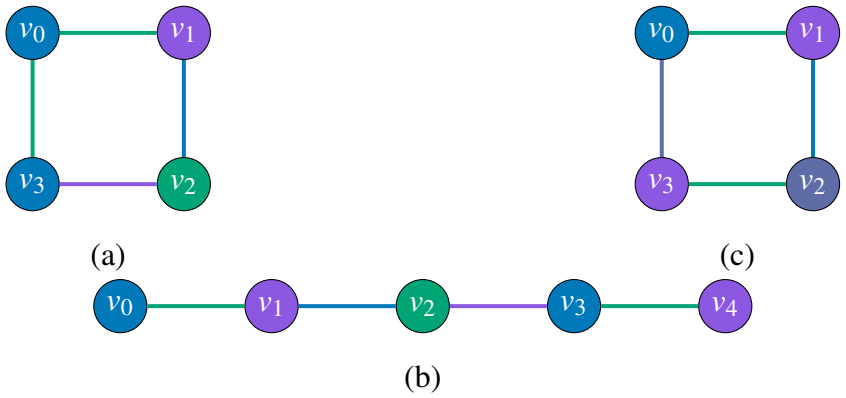


Figura 16: Em (a) a apresentação de conflito na 3-coloração total de C_4, em (b) a apresentação de $c(v_0) \neq c(v_4)$ em P_5, e em (c) uma 4-coloração total em C_4 após modificações realizadas em (a).

Para o caso $n \equiv 2 \pmod 3$, tem-se $|V(C_n) \cup E(C_n)| \equiv 1 \pmod 3$. Como $\chi''(P_{n+1}) = 3$, considere uma 3-coloração total do C_4 a partir da 3-coloração total do P_{n+1} conforme os casos anteriores. Pelo padrão periódico da coloração do caminho, quando $n \equiv 2 \pmod 3$ os vértices v_0 e v_n recebem cores distintas, isto é, $c(v_0) \neq c(v_n)$.

Novamente, a identificação de v_0 e v_n no ciclo impõe que esses vértices tenham a mesma cor, o que contradiz a coloração induzida por P_{n+1}. Assim, não existe uma 3-coloração total para C_n nesse caso.

Como exemplo, a Figura 17(a) apresenta uma tentativa de 3-coloração total de C_5 com conflito, pois $c(v_0) = c(v_0v_4)$. Na Figura 17(b), a coloração de P_6 evidencia a atribuição de cores distintas a v_0 e v_n.

Assim, obtém-se uma 4-coloração total ao atribuir-se cor 3 a aresta $v_{n-1}v_0$. Figura 17(c) apresenta uma 4-coloração total de C_5 após essa modificação.

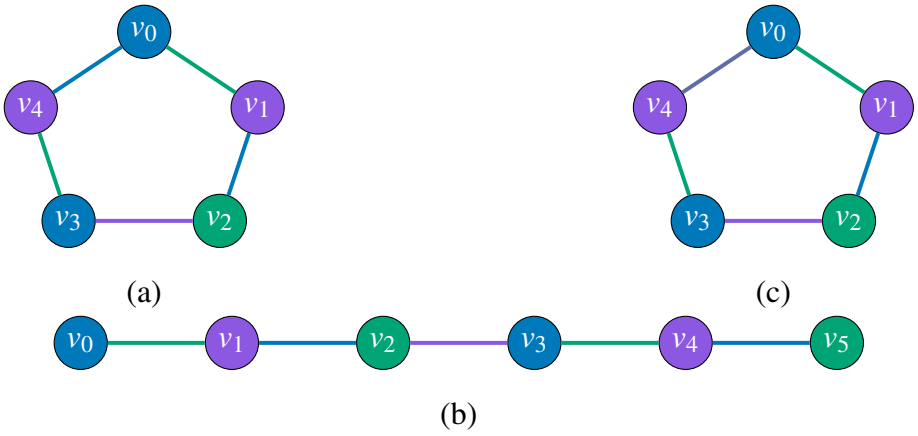


Figura 17: Em (a) a apresentação de conflito na coloração total de C_5, em (b) a apresentação de $c(v_0) \neq c(v_5)$ em P_6, e em (c) uma 4-coloração total em C_5 após modificação realizada em (a).

□

Para determinar o número cromático total dos grafos completos, como formalizado no Teorema 4, Yap [1996] faz uso do Lema 2 como ferramenta auxiliar na demonstração.

Lema 2 (Yap [1996])**.** *Se um grafo G contém um conjunto independente maximal S, e se $G-S$ contém um emparelhamento E' tal que:*

$$|E'| = \left\lfloor \frac{|V(G)| - |S|}{2} \right\rfloor$$

então:

$$\alpha''(G) = |S| + |E'|$$

Em geral, teremos:

$$\alpha''(G) \leq \alpha(G) + \left\lfloor \frac{|V(G)| - \alpha(G)}{2} \right\rfloor$$

□

Como exemplo de aplicação do Lema 2, temos o grafo G na Figura 18, que contém um conjunto independente maximal $S = \{v_4, v_5\}$ ilustrado na Figura 18(a). E também $G-S$ contém

um emparelhamento $E' = \{v_1v_2\}$, ilustrado na Figura 18(b), tal que:

$$|\{v_1v_2\}| = \left\lfloor \frac{|\{v_1,v_2,v_3,v_4,v_5\}| - |\{v_4,v_5\}|}{2} \right\rfloor$$

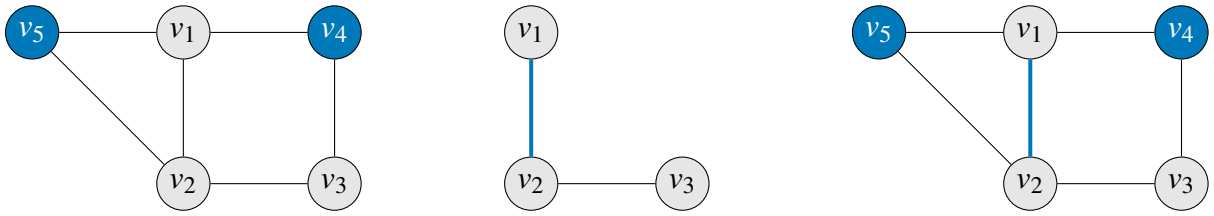
$$1 = \left\lfloor \frac{5-2}{2} \right\rfloor$$

$$1 = \left\lfloor \frac{3}{2} \right\rfloor$$

$$1 = 1$$

Então, como ilustrado na Figura 18(c) pelo conjunto independente total máximo $T = \{v_1v_2, v_4, v_5\}$:

$$\alpha''(G) = |\{v_4,v_5\}| + |\{v_1v_2\}| = 2+1 = 3$$



(a) (b) (c)

Figura 18: Em (a) um grafo G com um conjunto maximal $S = \{v_4, v_5\}$, em (b) $G - S$ com um emparelhamento $E' = \{v_1v_2\}$, e em (c) G com um conjunto independente total máximo $T = \{v_1v_2, v_4, v_5\}$.

Teorema 4. *(Yap [1996]) Um grafo completo K_n é Tipo 1 quando n é ímpar, e Tipo 2 quando n é par.*

Demonstração. Seja $V(K_n) = \{v_1, \ldots, v_n\}$. Como K_n é $(n-1)$-regular, $\alpha(K_n) = 1$, e todo conjunto independente máximo de K_n é da forma $\{v_i\}$ para algum $i \in \{1, \ldots, n\}$.

Primeiro, suponha que n é ímpar. Pelo Lema 2, existe um emparelhamento da seguinte forma $M_i = \{v_{i+j}v_{i-j} \mid 1 \leq j \leq \frac{n-1}{2}\}$, em que os índices $i+j$ e $i-j$ são calculados em módulo n. Denotemos o conjunto independente total de elementos com cores i, o conjunto $T_i = M_i \cup \{v_i\}$ e que $\bigcup_{i=1}^{n} T_i = V(K_n) \cup E(K_n)$, conclui-se, portanto, que $\chi''(K_n) = n = \Delta(K_n) + 1$.

Agora, suponha n par. Pelo Lema 2, temos:

$$\begin{aligned}\alpha''(K_n) &\leq \alpha(K_n) + \left\lfloor \frac{|V(K_n)| - \alpha(K_n)}{2} \right\rfloor \\ &\leq 1 + \left\lfloor \frac{n-1}{2} \right\rfloor \\ &= 1 + \frac{n-2}{2} \\ &= \frac{n}{2}\end{aligned}$$

E como

$$|V(K_n) \cup E(K_n)| = n + \frac{n(n-1)}{2} = \frac{2n+n^2-n}{2} = \frac{n(2+n-1)}{2} = \frac{n(n+1)}{2}, e$$

$$\begin{aligned}\chi''(K_n) &\geq \frac{|V(K_n) \cup E(K_n)|}{\alpha''(K_n)} \\ &\geq \frac{\frac{n(n+1)}{2}}{\frac{n}{2}} \\ &\geq n+1\end{aligned}$$

Então, como K_{n+1} é ímpar, admite uma $(n+1)$-coloração total, e a restrição dessa coloração ao subgrafo induzido por K_n define uma coloração total válida de K_n, conclui-se que:

$$n+1 \leq \chi''(K_n) \leq \chi''(K_{n+1})$$

$$n+1 \leq \chi''(K_n) \leq n+1$$

$$\chi''(K_n) = n+1 = \Delta(K_n) + 1 + 1 = \Delta(K_n) + 2$$

□

A Figura 19 apresenta um exemplo do Teorema 4 para n ímpar, em que são apresentados alguns passos da demonstração de uma 5-coloração total de K_5. Na Figura 19(a), $T_1 = M_1 \cup \{v_1\} = \{v_{1+j}v_{1-j} \mid 1 \leq j \leq 2\} \cup \{v_1\} = \{v_2v_5, v_3v_4, v_1\}$. Na Figura 19(b), $T_2 = M_2 \cup \{v_2\} = \{v_{2+j}v_{2-j} \mid 1 \leq j \leq 2\} \cup \{v_2\} = \{v_3v_1, v_4v_5, v_2\}$. Para v_3, v_4, e v_5 os passos são análogos. E na Figura 19(c), $\bigcup_{i=1}^{5} T_i = \{T_1, T_2, T_3, T_4, T_5\}$. Tendo, portanto, $\chi''(K_5) = \Delta(K_5) + 1 = 5$.

A Tabela 1 apresenta um resumo dos resultados de coloração total para as três classes de

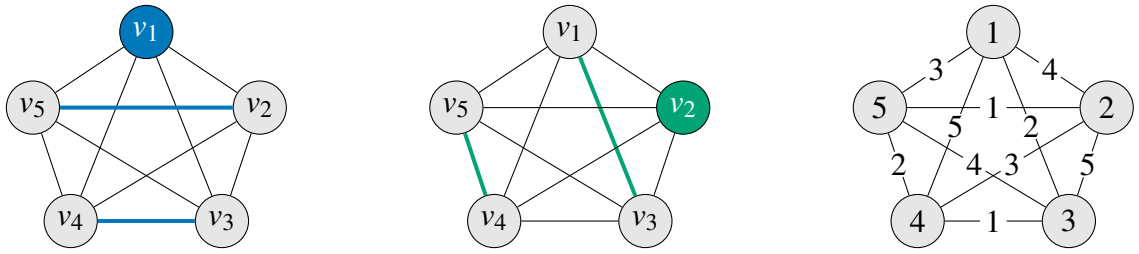


Figura 19: Em (a) o conjunto T_1, em (b) o conjunto T_2, e em (c) uma 5-coloração total de K_5, união dos conjuntos $T_1, T_2, ..., T_5$.

grafos abordadas neste trabalho. Nela são indicados o nome da classe e sua denotação em função da quantidade de vértices, a condição sob a qual o número cromático total χ'' é determinado, o respectivo valor de χ'' e o seu *Tipo*. Essa classificação está associada à Conjectura 1, distinguindo os casos em que χ'' assume os valores $\Delta(G)+1$ ou $\Delta(G)+2$.

Na coluna referente a χ'', observa-se que, para as classes dos caminhos e dos ciclos, o valor é constante, uma vez que $\Delta(P_n) = 2$ para $n \geq 2$, e $\Delta(C_n) = 2$ para $n \geq 3$. No entanto, para a classe dos grafos completos $\Delta(K_n) = n-1$, o que implica uma dependência direta do número cromático total em relação à ordem do grafo.

	Condição	χ''	Tipo
Caminhos (P_n)	$n \geq 2$	3	1
Ciclos (C_n)	$n \geq 3$ e $n \equiv 0\ (mod\ 3)$	3	1
	$n \geq 3$ e $n \equiv 1\ (mod\ 3)$	4	2
	$n \geq 3$ e $n \equiv 2\ (mod\ 3)$	4	2
Completos (K_n)	n ímpar	n	1
	n par	$n+1$	2

Tabela 1: Resumo da coloração total nas classes dos grafos caminho, ciclo e completos.

2.3 Aplicações Web

Uma **aplicação web** é um sistema acessado por meio de navegadores, projetado para funcionar via internet ou em redes locais. Diferentemente das aplicações instaladas localmente, uma aplicação web é, por natureza, **multiplataforma**, pois independe do sistema operacional utilizado pelo usuário. Além disso, não requer instalação, o que facilita seu uso imediato e acessível em diversos dispositivos.

Sua comunicação é realizada por meio do modelo **cliente-servidor**, denominação que caracteriza os dois tipos de agentes envolvidos. O **servidor** é um agente computacional, que pode ser constituído de um ou mais computadores, geralmente mantido em operação contínua (24 horas por dia, 7 dias por semana), aguardando solicitações de qualquer cliente. Por sua vez, o **cliente** é o agente responsável por enviar requisições ao servidor e receber respostas, que podem incluir códigos de status, imagens, arquivos de texto ou outros dados. Exemplos de clientes incluem navegadores, aplicativos móveis e sistemas computacionais diversos.

As informações provenientes dessa comunicação são exibidas no navegador por meio de uma interface desenvolvida especificamente para a aplicação web. A **interface**, também conhecida como **User Interface (UI)**, refere-se à forma como as informações são apresentadas ao usuário, por meio da disposição visual de menus, listagens, tabelas e botões. Essa interface é responsável por mediar a interação entre o usuário e o sistema, sendo um elemento essencial para proporcionar uma boa experiência de uso.

Para a construção da UI e a implementação dos algoritmos executados no servidor, utiliza-se uma **linguagem de programação**, definida como um conjunto de instruções preestabelecidas que possibilitam a comunicação entre o desenvolvedor e a máquina. Uma **linguagem superset** é um tipo de linguagem que incorpora todas as funcionalidades de outra linguagem-base, adicionando novos recursos e mecanismos que facilitam o desenvolvimento.

Já uma **biblioteca** consiste em um conjunto de funções ou rotinas reutilizáveis, escritas em uma determinada linguagem de programação, com o objetivo de agilizar o desenvolvimento de software. Essas bibliotecas são normalmente desenvolvidas com propósitos específicos; por exemplo, podem ser utilizadas para facilitar a criação de interfaces gráficas ou para lidar com operações do lado servidor, como o tratamento de requisições.

Um servidor pode incluir uma **base de dados**, ou seja, um conjunto de informações organizadas e armazenadas digitalmente. O **banco de dados** é o sistema responsável por gerenciar

essa base, permitindo operações como inserção, atualização, exclusão e consulta de dados. Por meio de uma linguagem de programação no lado servidor, é possível acessar essas informações e enviá-las ao cliente, como um navegador web, para serem exibidas ao usuário de forma estruturada e dinâmica.

Um **protocolo** é um conjunto de regras e procedimentos padronizados para a comunicação entre dispositivos ou sistemas. Um dos protocolos fundamentais para o funcionamento de uma aplicação web é o **Hypertext Transfer Protocol (HTTP)**. Seu funcionamento ocorre da seguinte forma: o HTTP estabelece a conexão entre cliente e servidor; em seguida, o servidor aguarda uma requisição. Ao recebê-la, responde com um código de status e uma mensagem opcional, que pode incluir arquivos.

Uma **requisição** é principalmente composta por um método HTTP, um caminho e, opcionalmente, um corpo com informações adicionais. O **método HTTP** indica o propósito da requisição. Os principais métodos são: `GET` (solicita um recurso), `POST` (solicita a adição de um novo recurso), `PUT` (solicita a atualização de um recurso existente) e `DELETE` (solicita a remoção de um recurso). O **caminho** especifica o endereço do recurso solicitado; por exemplo, `/computacao/alunos` pode representar um caminho em uma aplicação para o recurso que contém informações acadêmicas de alunos de computação. O **corpo da requisição**, comumente chamado de *body*, quando presente, contém dados adicionais que auxiliam o servidor no processamento, como, por exemplo, os dados de um novo usuário a ser inserido na base de dados da aplicação. Um exemplo de requisição é: `GET /disciplinas/GCC1208/alunos`, que pode representar, em uma aplicação, a solicitação das informações dos alunos da disciplina de código `GCC1208`.

Já uma **resposta** é composta, principalmente, por um código de status HTTP e, opcionalmente, por uma mensagem. Um **código de status HTTP** é um número que indica se uma requisição foi concluída com sucesso ou se ocorreu algum erro. Esses códigos são agrupados em cinco categorias, de acordo com sua faixa numérica: 100–199 (informacional), 200–299 (sucesso), 300–399 (redirecionamento), 400–499 (erro do cliente) e 500–599 (erro do servidor). A **mensagem de resposta** pode conter a informação solicitada pelo cliente, uma confirmação de sucesso ou até mesmo um arquivo.

Capítulo 3

Aplicações Relacionadas

A comunidade científica de Teoria dos Grafos, além de produzir artigos e contribuições teóricas, desenvolveu ao longo dos anos artefatos computacionais voltados à aplicação prática do conhecimento gerado, bem como sistemas para centralização e busca de informações sobre o tema, e aplicações que auxiliam a compreensão de resultados teóricos.

Neste capítulo, são descritas quatro aplicações web relacionadas a este trabalho, seja por apresentarem informações sobre problemas de coloração, seja por possuírem um escopo acadêmico com foco em estruturas de dados, como grafos. Em seguida, são mencionados um *software* matemático e uma biblioteca Python relevantes para a comunidade de Teoria dos Grafos. Por fim, é apresentado um comparativo entre as aplicações apresentadas e o trabalho proposto, sintetizado ao final do capítulo por meio de uma tabela.

A primeira aplicação relacionada, o *House of Graphs* (HoG)[1], foi desenvolvida pela Universidade de Gante e pela Universidade Católica de Lovaina, ambas na Bélgica. Seu principal objetivo é fornecer um serviço útil à comunidade de Teoria dos Grafos, permitindo aos usuários realizar buscas por grafos específicos, geralmente no contexto de algum problema ou conjectura particular. A aplicação consiste, basicamente, em duas partes: um meta-diretório de grafos e um banco de dados pesquisável [Coolsaet et al., 2023].

Em seu meta-diretório, são hospedadas listas completas de todos os grafos pareados pertencentes a uma determinada classe, até uma ordem específica. Juntamente com cada lista, são disponibilizados *links* para artigos que descrevem os algoritmos utilizados em sua geração. Algumas listas são geradas pela equipe do HoG, outras são disponibilizadas por colaboradores externos.

Em seu banco de dados pesquisável, principal componente do HoG, são disponibilizados três tipos de busca: por grafos, por um grafo específico e pelos grafos adicionados recentemente. A busca de grafos realiza a pesquisa de grafos que correspondam aos critérios estabelecidos pelo usuário, como valor exato de uma invariante (um parâmetro estrutural do grafo), valor de uma invariante que pertence a um intervalo de valores, pertencimento a uma classe de grafos,

[1] Disponível em: https://houseofgraphs.org/. Acessado em 17/06/2025.

valor de uma invariante do tipo inteiro que seja ímpar ou par, texto que corresponda a alguma informação do grafo (nome conhecido do grafo, classe, etc), dentre outros. Na Figura 20(a) um exemplo de busca de grafo pela invariante número de independência com o valor 1 e alguns dos resultados. Na Figura 20(b) um grafo K_7, um dos grafos encontrados pela busca, com suas informações como matriz/lista de adjacências, valores de invariantes do grafo, disponibilização do *download* do grafo em alguns formatos. No contexto da coloração, são apresentados o número cromático e o índice cromático.

A busca de um grafo específico pode ser feita de três formas: por um id do grafo gerado pela própria plataforma, por um *graph6 string*[2], ou por meio de um desenho do grafo feito pelo usuário na plataforma. A busca pelos grafos adicionados recentemente é realizada a partir de uma entrada do usuário, que informa um valor entre 5 e 50 para determinar a quantidade de grafos exibidos no resultado.

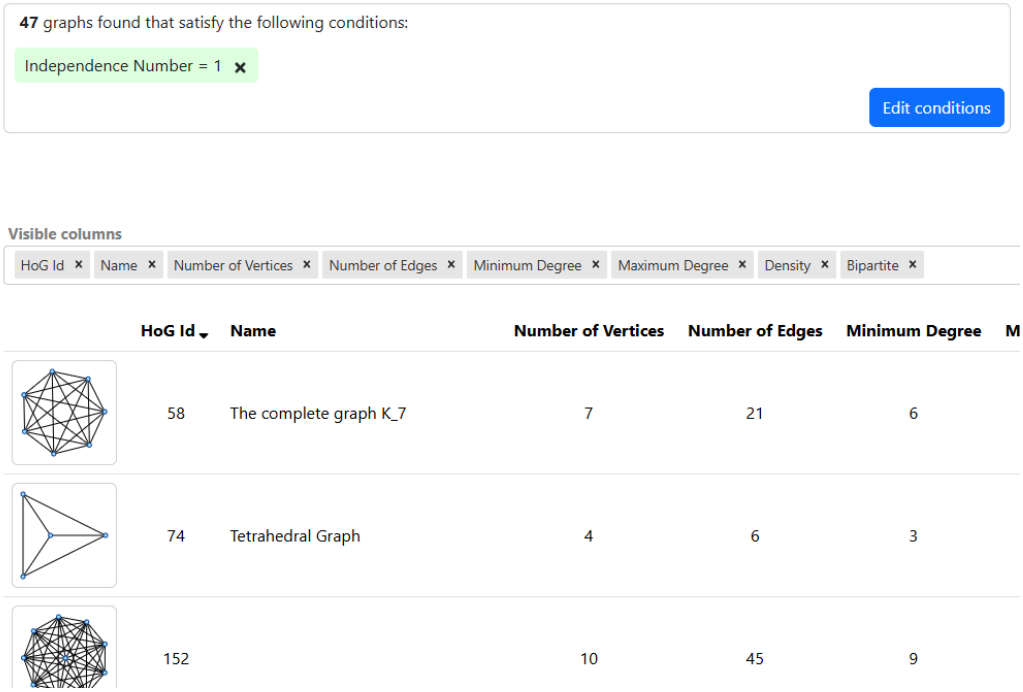
A ISGCI[3] , desenvolvida pelo Departamento de Ciência da Computação da Universidade de Rostock (Alemanha), é uma enciclopédia de classes de grafos. Para de Ridder et al. [2025] ao longo dos anos, as comunidades de matemática e ciência da computação descreveram muitas classes de grafos na tentativa de ampliar tanto a compreensão sobre propriedades fundamentais dos grafos quanto a capacidade de resolver problemas práticos de forma eficiente. Esse trabalho tem sido tão produtivo que um grande número de classes foi definido, cujas relações são difíceis de visualizar mesmo para os mais experientes. Logo a ISGCI disponibiliza informações de classes de grafos e suas relações.

Além da pesquisa direta pelo nome de uma classe, a ISGCI possui quatro seções em sua aplicação: classes clássicas, classes por definição, problemas e parâmetros. Classes clássicas apresenta classes de grafos que possuem identidade estabelecida na literatura, por exemplo, a classe dos bipartidos, planares e árvores. Classes por definição apresenta classes de grafos por um termo textual de sua definição, por exemplo, a busca pelo termo *edge* tem como resultado algumas classes, dentre elas os completos, pois sua definição na aplicação é: *"A graph is complete if there is an edge between any two vertices."*.

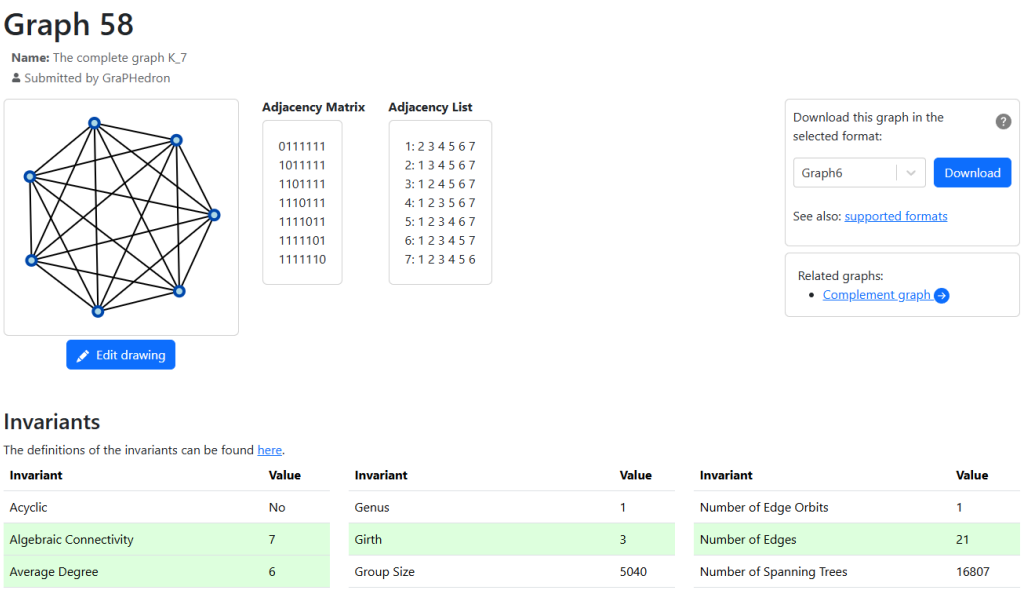
A seção problemas apresenta classes de grafos agrupadas por complexidade computacional de seu algoritmo para resolução de um problema, por exemplo, para o problema de coloração

[2]Graph6 é uma codificação de grafos simples que utiliza o triângulo superior da matriz de adjacência (excetuando a diagonal principal), e caracteres ASCII de código decimal 63 a 126. A descrição completa da codificação graph6 feita por seu autor Brendan McKay está disponível em `https://users.cecs.anu.edu.au/~bdm/data/formats.txt`. Acessado em 19/06/2025.

[3]Disponível em `https://www.graphclasses.org/`. Acessado em 19/06/2025.



(a)



(b)

Figura 20: Exemplo de busca de grafo no HoG. Em (a) a pesquisa por grafos pela invariante número de independência igual a 1, e em (b) um do grafos encontrados K_7 e suas informações

(de vértices), as classes são agrupadas por: linear, polinomial, NP-completo e desconhecido para o ISGCI. E a seção parâmetros apresenta classes de grafos agrupadas por limitada e não-limitada de acordo com a presença ou não de um limitante superior ao valor desse parâmetro na classe.

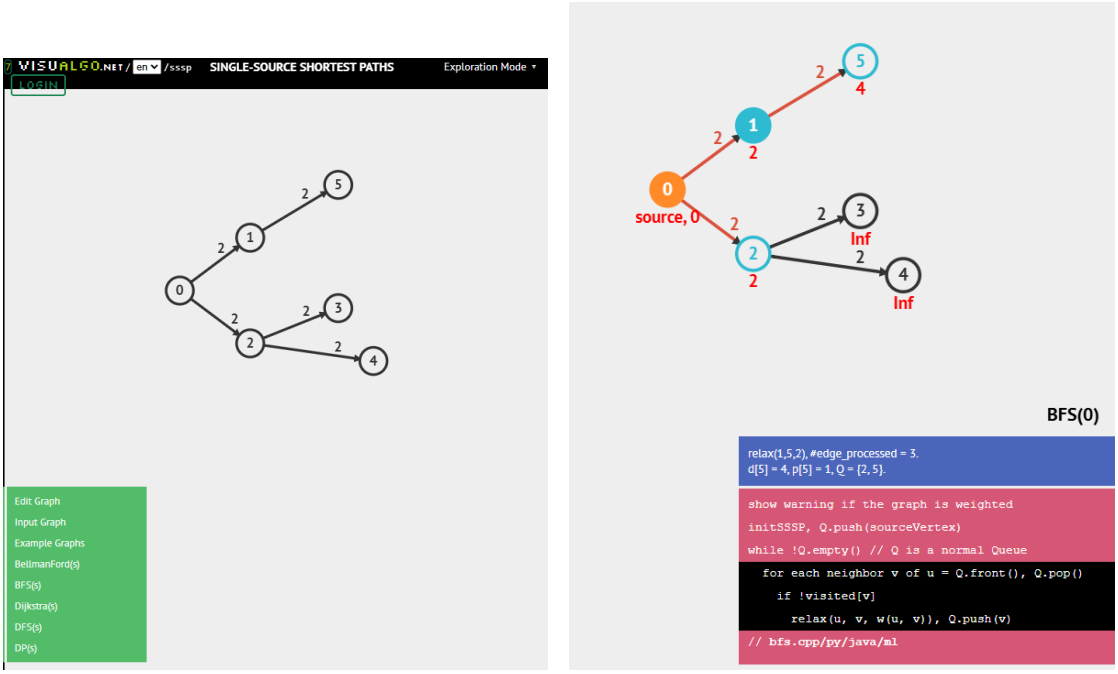
Quando uma classe de grafos é selecionada por meio da busca textual ou pela identificação através de uma das seções disponíveis, uma página com informações acerca da classe é exibida. Informações como sua definição, classes relacionadas, inclusões (diagrama hierárquico de algumas classes que contêm a classe selecionada), parâmetros (e classificação como limitada e não limitada) e problemas (e a complexidade computacional do algoritmo para resolução desse problema na classe selecionada). No contexto de coloração de grafos, são apresentados a classificação do parâmetro número cromático e a complexidade computacional para o problema de coloração de vértices. Por exemplo, para a classe dos grafos completos[4] são apresentados o número cromático como não limitado e o problema de coloração de vértices com a complexidade linear.

O VisuAlgo[5] , desenvolvido pela Escola de Computação da Universidade Nacional de Singapura, teve como objetivo inicial facilitar a compreensão de estruturas de dados e algoritmos para seus alunos, oferecendo uma plataforma de aprendizagem interativa e no próprio ritmo. Com visualizações das estruturas de dados e do passo a passo dos algoritmos, a aplicação está disponível na internet para utilização.

A aplicação possui 26 módulos, dentre eles: algoritmos de ordenação, *linked list*, árvore binária, *hash table* e busca binária. Para o contexto deste trabalho, outros módulos relevantes: grafos, caminho mínimo (busca em largura, dijkstra entre outros), emparelhamento máximo e fluxo máximo. Na Figura 21(a) um grafo de exemplo no módulo de caminho mínimo e as opções como editar o grafo e escolher entre algumas opções de algoritmos como Busca em Largura, Busca em Profundidade, BellmanFord e Dijkstra. Na Figura 21(b) uma etapa da execução de uma busca em largura em que o vértice em laranja é o vértice inicial, os vértices com as bordas em azul e fundo transparente estão na pilha e o vértice com borda e fundo azul está sendo visitado; a parte do código executado na etapa está evidenciada com fundo preto.

O Graph Online[6][7] , um projeto de código aberto da Unick Soft (Rússia), possui uma interface interativa para geração e visualização de grafos, assim como o resultado da aplicação de

[4]Disponível em https://www.graphclasses.org/classes/gc_1241.html Acessado em 20/06/2025.
[5]Disponível em https://visualgo.net/. Acessado em 20/06/2025.
[6]Disponível em https://graphonline.top/pt/. Acessado em 20/06/2025.
[7]Repositório do projeto em https://github.com/UnickSoft/graphonline. Acessado em 20/06/2025.



(a) (b)

Figura 21: Exemplo de funcionalidade no VisuAlgo. Em (a) um exemplo no módulo de caminho mínimo com opções de algoritmos, em (b) uma etapa da execução de uma busca em largura.

algoritmos em grafos. Entre suas funcionalidades, destaca-se a possibilidade de gerar e editar grafos manualmente por meio de uma interface visual, permitindo a adição de vértices e arestas diretamente na tela com cliques do usuário, além da manipulação da matriz de adjacência de forma direta. É possível selecionar vértices e arestas em conjunto, editá-las ou removê-las, e alterar as cores dos elementos do grafo e do plano de fundo.

Em relação às funcionalidades de algoritmos do Graph Online, algumas delas são: busca em largura, dijkstra, encontrar caminho euleriano, encontrar ciclo euleriano e encontrar componentes conexos. No contexto de coloração de grafos, possui uma funcionalidade que apresenta uma coloração de vértices do grafo e seu número cromático, como ilustrado na Figura 22.

Por fim, embora não sejam aplicações web para fins informacionais ou acadêmicas, é importante citar um *software* matemático e uma biblioteca python relevantes à comunidade de Teoria dos Grafos. O SageMath[8] é um *software* baseado em python que possui estruturas matemáticas e algoritmos nativos para uso, como exemplo grafos e aplicações em coloração. Em termos de coloração, a aplicação permite obter uma coloração de vértices e seu número cromático, bem como uma coloração de arestas e seu índice cromático; entretanto, não possui funcionalidade

[8]Site: https://www.sagemath.org/. Acessado em 22/06/2025.

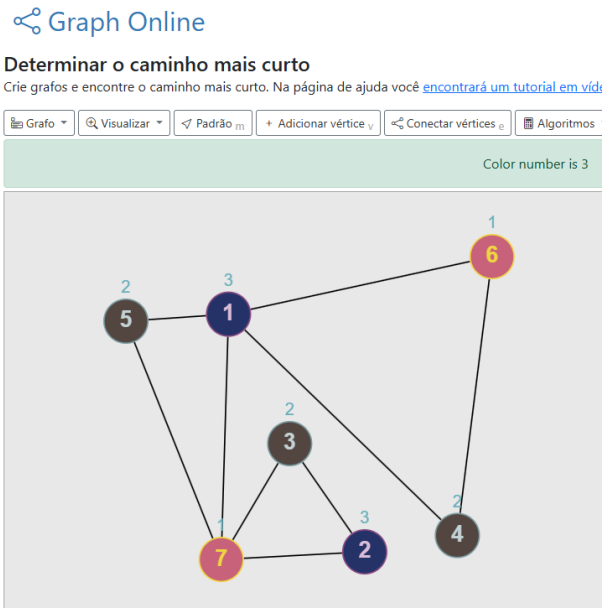


Figura 22: Exemplo da funcionalidade de coloração de vértices no GraphOnline.

nativa para coloração total. A Networkx[9] é uma biblioteca Python para criação, manipulação e estudo da estrutura, dinâmica e funções de grafos (especialmente redes complexas). Em coloração, é possível obter uma coloração subótima de vértices.

Dadas as aplicações web descritas, a Tabela 2 apresenta um comparativo entre elas e o Total Color, apresentado no Capítulo 4, com base em quatro características: escopo, coloração, forma de apresentação e interatividade. O escopo refere-se ao objetivo ou finalidade da aplicação, sendo informacional (uma base de dados, enciclopédia) ou acadêmico (ilustrativo com fins didáticos ou de pesquisa). A coloração refere-se ao tipo de coloração apresentada na aplicação, sendo V (coloração de vértices), A (coloração de arestas) e T (coloração total).

A apresentação refere-se ao modo que a coloração contida na aplicação é apresentada, sendo este textual, ilustrativo (apresentação da coloração graficamente) ou ilustrativo passo a passo (construção em etapas). E a interatividade refere-se à disponibilidade de interação do usuário com a coloração apresentada pela aplicação, de forma a permitir modificação e testes da coloração. O Total Color difere das demais aplicações relacionadas por ser o único com foco em coloração total com objetivo acadêmico, permitindo interatividade e recurso para apoio à pesquisa.

[9]Site: https://networkx.org/. Acessado em 22/06/2025.

	Escopo	Coloração	Apresentação	Interatividade
House of Graphs	Informacional	V e A	Textual	Não
ISGCI	Informacional	V	Textual	Não
VisuAlgo	Acadêmico	-	-	-
Graph Online	Acadêmico	V	Ilustrativo	Não
Total Color	Acadêmico	T	Ilustrativo (passo a passo)	Sim

Tabela 2: Aplicações relacionadas e a proposta do Total Color.

Capítulo 4

Total Color

A aplicação web proposta neste trabalho, denominada Total Color, está disponível para utilização em https://total-color.vercel.app/[1], e tem como objetivos apresentar a coloração total em diferentes classes de grafos, promover o aprendizado interativo do usuário e fornecer resultados que possam apoiar atividades de pesquisa. Neste capítulo, são descritas as funcionalidades do *Total Color*, as tecnologias utilizadas, a estrutura da aplicação, bem como os algoritmos empregados na coloração total.

O fluxo principal das funcionalidades do sistema estão modeladas no diagrama de atividades apresentado na Figura 23. Inicialmente, o usuário escolhe um dos modos de operação disponíveis: classes ou livre. No modo classes, o usuário seleciona uma das classes de grafos disponíveis (caminhos, ciclos ou grafos completos), define o valor do parâmetro associado à classe, que para as classes disponíveis, corresponde ao número de vértices, escolhe o layout e, em seguida, aciona a opção para gerar o grafo.

Após a geração do grafo da classe selecionada, o usuário pode visualizar uma coloração total desse grafo. Além disso, é possível interagir com a estrutura gerada selecionando um ou mais elementos, informando um número para associá-los a uma cor e, posteriormente, recebendo um retorno do sistema quanto à validade da coloração. Caso algum elemento receba a mesma cor de um elemento adjacente ou incidente, esses elementos são destacados em vermelho, indicando a ocorrência de conflito.

No modo livre, o usuário carrega um arquivo .txt contendo a matriz de adjacência ou um arquivo no formato .g6, seleciona o layout e solicita a geração do grafo. Em seguida, o usuário pode interagir com o grafo da mesma forma que no modo classes, com exceção da visualização da coloração total.

[1] Acessado em 16/12/2025.

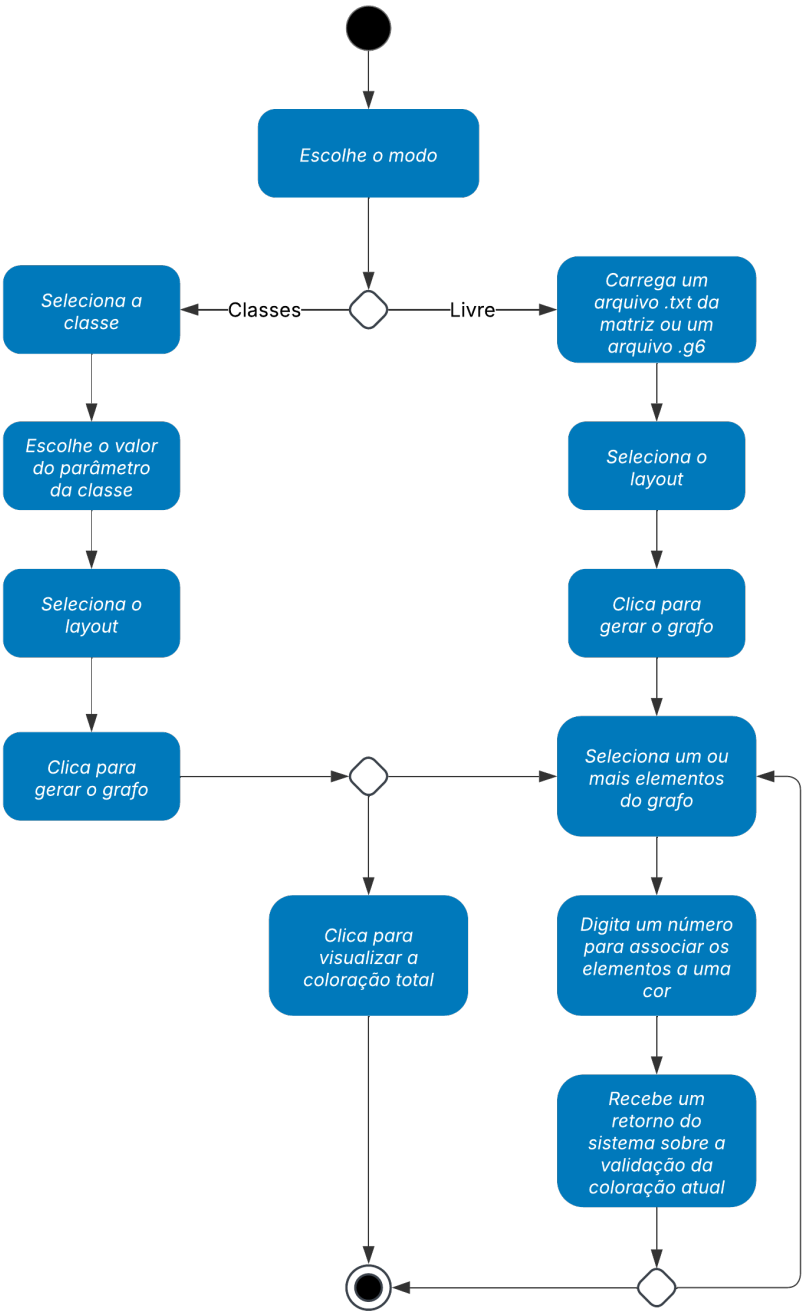


Figura 23: Diagrama de atividades do fluxo principal do Total Color.

O diagrama de atividades apresentado na Figura 23 descreve funcionalidades que são implementadas no Total Color exclusivamente no lado cliente, com execução em navegador web, assegurando à aplicação um ambiente multiplataforma. A implementação faz uso das tecnologias Next.js, React, TypeScript e Cytoscape.js.

O framework Next.js[2] é utilizado como base para o desenvolvimento da aplicação. Criado pela empresa Vercel, o Next.js expande as funcionalidades do React, oferecendo recursos como renderização híbrida (estática e dinâmica), geração de páginas no lado do servidor (*server-side rendering*) e otimização automática de desempenho. Essas características proporcionam uma experiência de uso mais fluida ao usuário, além de facilitar a escalabilidade e a manutenção da aplicação. Atualmente, o Next.js é amplamente adotado por grandes empresas e projetos de destaque na área de desenvolvimento web, consolidando-se como um dos principais frameworks do ecossistema JavaScript.

A geração da UI é realizada no Next.js por meio do React[3], biblioteca JavaScript desenvolvida pelo Facebook que permite a criação de interfaces utilizando elementos de marcação semelhantes ao HTML, além da reutilização de código por meio de componentes. O React possui ampla relevância na comunidade de desenvolvimento web, sendo, segundo o Stack Overflow Developer Survey 2024[4], a tecnologia web preferida entre desenvolvedores profissionais.

Como o React é uma biblioteca baseada em JavaScript, linguagem de tipagem dinâmica, utiliza-se a linguagem TypeScript[5], desenvolvida pela Microsoft. O TypeScript é uma linguagem *superset* de JavaScript, com tipagem estática, que contribui para a legibilidade do código, organização estrutural e detecção de erros durante o desenvolvimento.

Para a visualização e manipulação de grafos, empregou-se a biblioteca Cytoscape.js[6], desenvolvida pela Universidade de Toronto, no Canadá. Essa biblioteca é amplamente utilizada por empresas como Google e QuantStack (Jupyter), bem como por órgãos governamentais, como a Agência de Segurança Nacional dos Estados Unidos (NSA) e o Serviço Nacional de Saúde do Reino Unido (NHS) [Franz et al., 2023].

[2]Site oficial: https://nextjs.org/ . Acessado em 24/06/2025.
[3]Site oficial: https://react.dev/. Acessado em 23/06/2025.
[4]Disponível em: https://survey.stackoverflow.co/2024/technology. Acessado em 23/06/2025.
[5]Site oficial: https://www.typescriptlang.org/. Acessado em 24/06/2025.
[6]Site oficial: https://js.cytoscape.org/. Acessado em 24/06/2025.

4.1 Funcionalidades

O Total Color é dividido em dois modos principais de utilização: classes e livre. Na tela inicial, ilustrada na Figura 24, esses modos são apresentados ao usuário acompanhados de um breve texto que descreve suas funcionalidades.

No modo classes, é possível selecionar grafos pertencentes a uma das classes contempladas neste trabalho (caminhos, ciclos e grafos completos), visualizar suas representações, realizar a coloração total e verificar automaticamente a validade da solução dada.

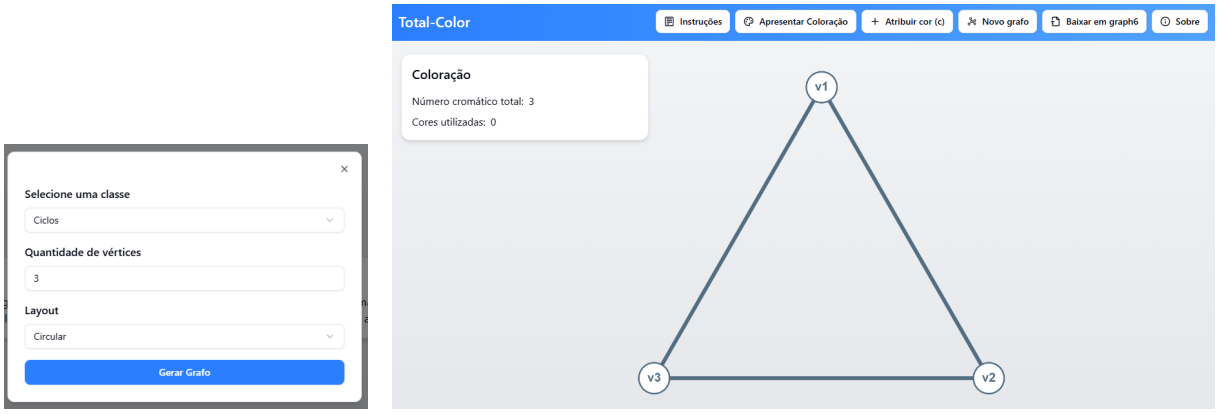
No modo livre, o usuário pode importar um arquivo .txt contendo uma matriz de adjacência, um arquivo .txt com a coloração do grafo inclusa, ou um arquivo no formato .g6, permitindo a construção e manipulação de grafos arbitrários. Após o carregamento do grafo, o sistema disponibiliza as ferramentas necessárias para realizar a coloração total e acompanhar sua validação.

Figura 24: Tela inicial do Total Color.

Ao selecionar o modo classes, o usuário insere as informações necessárias para a geração do grafo, como a classe desejada, o parâmetro associado à criação do grafo e o *layout*. Tanto o parâmetro quanto o *layout* possuem valores padrão previamente definidos. A Figura 25(a) apresenta um exemplo de preenchimento das opções para a geração do grafo C_3.

A Figura 25(b) exibe o grafo C_3 resultante após a configuração mostrada na Figura 25(a). Ainda na Figura 25(a), observa-se à esquerda um painel contendo informações relacionadas à coloração total da classe selecionada, sendo elas o número cromático total do grafo e a quantidade de cores utilizadas no estado atual. À medida que novas cores são atribuídas aos elementos do grafo, seja automaticamente pelo sistema ou manualmente pelo usuário, a quantidade de cores utilizadas e os rótulos correspondentes são atualizados dinamicamente.

Por meio das opções do menu apresentadas na Figura 25(b), o usuário pode visualizar ins-



(a) (b)

Figura 25: Exemplo de geração de um C_3 no Total Color. Em (a) as opções para gerar um C_3, e em (b) a visualização de um C_3 com a interface para interação com o sistema.

truções para interagir com o grafo, acompanhar a coloração total de forma progressiva, atribuir colorações manualmente, gerar um novo grafo, acessar informações sobre o sistema, bem como realizar o download do grafo no formato `.g6`.

A leitura e a geração de grafos no formato *graph6* (`.g6`) constituem uma das funcionalidades complementares da aplicação[7], formato esse que mapeia matrizes de grafos não direcionados em caracteres ASCII. No *Total Color*, há a possibilidade de importar grafos nesse formato, que são posteriormente convertidos em matrizes para uso interno nas funcionalidades do sistema. Além disso, após a geração de um grafo, o sistema permite sua exportação no formato *graph6*.

A funcionalidade de leitura e conversão do formato *graph6* não estava disponível como *node package* para uso público pela comunidade JavaScript/TypeScript. Assim, uma das contribuições deste trabalho consiste em um *node package* denominado *graph6*, atualmente hospedado no repositório oficial do *Node Package Manager (NPM)* para uso público no endereço: `https://www.npmjs.com/package/graph6`.

O *node package graph6* consiste em uma classe, denominada *Graph6*, com dois métodos estáticos: *parse* e *toMatrix*. O método *parse* realiza a transformação de uma matriz de adjacência para o formato textual .g6, enquanto o método *toMatrix* realiza o processo inverso. Para o entendimento do método *parse* (Algoritmo 3), é necessário compreender previamente os métodos *R* (Algoritmo 1) e *N* (Algoritmo 2), ambos documentados na especificação do formato .g6.

O método *R* (Algoritmo 1) consiste na transformação de um número binário em sua representação textual, na qual cada componente corresponde a um conjunto de seis dígitos consecu-

[7]Especificação disponível em: `https://users.cecs.anu.edu.au/~bdm/data/formats.txt`. Acessado em 23/11/2025.

tivos do binário convertido em um número inteiro de 63 a 126, representando um símbolo da tabela ASCII.

Algoritmo 1: Método auxiliar privado *R* da classe Graph6

Entrada: Forma textual de um número binário (x)
Saída: Texto com números inteiros de 63 a 126 separados por espaço
codigosCaracteres $\leftarrow \varepsilon$
vetor $\leftarrow$ completar com 0's à direita de x até que seu tamanho seja múltiplo de 6
Para $i \leftarrow 0$ **até** $vetor.tamanho/6-1$
  codigosCaracteres $\leftarrow$ *codigosCaracteres* $+$ ' ' $+$ [(parte *i* do vetor de binários na base decimal) + 63]
Fim-Para
Retorne *codigosCaracteres*

O método *N* (Algoritmo 2) consiste na transformação da ordem do grafo (n), a quantidade de vértices, em uma representação textual separada por espaços, na qual cada componente corresponde a um número inteiro entre 63 e 126, representando um símbolo da tabela ASCII. Caso n esteja entre 0 e 62, o método retorna o valor $n+63$; caso contrário, o retorno é composto por um ou dois números iguais a 126, seguidos por um espaço e, então, pelo resultado da aplicação do método *R* sobre a forma binária de n.

Algoritmo 2: Método auxiliar privado *N* da classe Graph6

Entrada: Ordem do grafo (n)
Saída: Texto com números inteiros de 63 a 126 separados por espaço
codigosCaracteres $\leftarrow \varepsilon$
Se $0 \leq n \wedge n \leq 62$ **então**
  codigosCaracteres $\leftarrow [(n+63)\ em\ forma\ textual]$
Fim-Se
Senão se $63 \leq n \wedge n \leq 258047$ **então**
  codigosCaracteres $\leftarrow$ '126 ' $+$ R(Forma binária de n)
Fim-Se
Senão se $258048 \leq n \wedge n \leq 68719476735$ **então**
  codigosCaracteres $\leftarrow$ '126 126 ' $+$ R(Forma binária de n)
Fim-Se
Retorne *codigosCaracteres*

Por fim, o método *parse* (Algoritmo 3) consiste no mapeamento da matriz de adjacência de um grafo não direcionado para um texto no formato `.g6`. Para esse mapeamento, duas informações principais são extraídas da matriz: (i) a ordem do grafo (n); e (ii) um vetor textual (x), cujos elementos representam os valores de adjacência do triângulo superior da matriz, com exceção da diagonal principal. A ordem de inserção dos valores do triângulo superior em x segue a varredura por colunas da matriz.

O retorno do método *parse* é formado pela concatenação dos símbolos ASCII correspondentes aos valores inteiros obtidos por meio da aplicação de $N(n)$ e $R(x)$, sendo N responsável pela codificação da ordem do grafo e R responsável pela transformação da sequência binária referente às adjacências.

Algoritmo 3: Método estático *parse* da classe Graph6

Entrada: Matriz de adjacência composta por números 0's e 1's
Saída: Texto no formato .g6
n $\leftarrow$ *matriz.tamanho*
x $\leftarrow \varepsilon$
Para *coluna* $\leftarrow 1$ **até** $n-1$
  Para *linha* $\leftarrow 0$ **até** *coluna* -1
    x $\leftarrow$ x + *matriz*[*linha*][*coluna*]
  Fim-Para
Fim-Para
graph6 $\leftarrow$ (símbolos ASCII da representação decimal de $N(n) + ' ' + R(x)$)
Retorne *graph6*

Como exemplo, a obtenção da representação de C_4 em *.g6* a partir da conversão de sua matriz de adjacência. Inserindo como entrada a matriz de adjacência de C_4 (apresentada na Figura 26) no método `Graph.parse`, temos pelo Algoritmo 3 que $n = 4$ e $x = '101101'$ e como retorno temos símbolos ASCII da representação decimal de $N(4) + ' ' + R('101101')$. Para $N(4)$ aplicamos o Algoritmo 2: $N(4) = '(4+63)' = '67'$; e para $R('101101')$ aplicamos o Algoritmo 1: $R('101101') = '(45+63)' = '108'$. Logo o método `Graph.parse` retorna símbolos ASCII da representação decimal de $'67\ 108'$, ou seja, *Cl*.

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Figura 26: Matriz de adjacência de C_4.

Ao selecionar o modo *livre*, o usuário dispõe de duas opções: gerar um grafo do zero ou gerar um grafo a partir de um arquivo nos formatos `.txt` ou `.g6`. Para a geração de um grafo do zero, o usuário conta com opções para a criação de vértices e arestas. Já na geração a partir de um arquivo `.txt`, o conteúdo pode assumir duas formas: (i) uma matriz de adjacências; ou (ii) um padrão contendo uma coloração do grafo. Ambos os tipos de conteúdo no formato `.txt`, assim como o formato `.g6`, são descritos e exemplificados na própria aplicação.

A Figura 27(a) apresenta a interface para a geração de um grafo no modo livre, enquanto a Figura 27(b) ilustra a visualização de um grafo após o usuário optar pela geração de um grafo do zero, adicionar quatro vértices, inserir quatro arestas e atribuir coloração a dois elementos do grafo. Observa-se que, além das funcionalidades disponíveis no modo classes, nesse modo o usuário pode adicionar vértices e arestas, bem como atribuir cores diretamente ao grafo.

Essas interações podem ser realizadas tanto por meio das opções da interface quanto por atalhos de teclado, sendo `v` para adicionar vértices, `e` para adicionar arestas e `c` para atribuir cor.

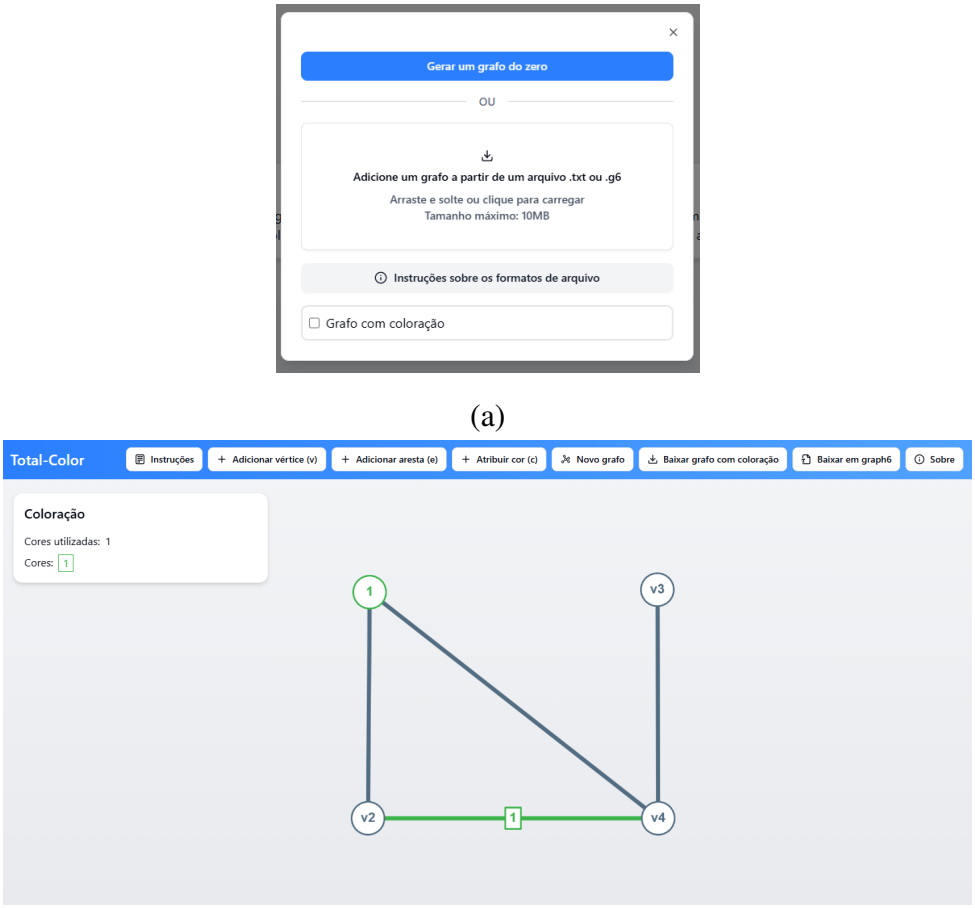


Figura 27: Em (a) as opções para geração de um grafo no modo livre: grafo do zero ou grafo por meio de um arquivo *.txt* ou *.g6*, e em (b) a visualização de um grafo gerado do zero pelo usuário ao interagir com o sistema.

4.2 Estrutura da aplicação e algoritmos de coloração total

O *Total Color* é estruturado em quatro partes principais: *app*, *components*, *contexts* e *lib*. O diretório *app* é responsável pela inicialização da aplicação. Em *components* encontra-se a estrutura dos principais módulos da interface, organizada em três subcomponentes: *appbar*, *graph-generator* e *graph-viewer*.

O componente *appbar* concentra a lógica do *menu*, bem como os algoritmos responsáveis pela exportação do grafo, tanto no formato `.txt` com uma coloração quanto no formato *.g6*. O componente *graph-generator* implementa a lógica da interface para a obtenção das informações necessárias à geração de grafos pertencentes a uma classe específica ou de grafos genéricos. Por fim, o componente *graph-viewer* reúne toda a lógica de visualização e interação do usuário com o grafo, incluindo a atribuição de cores, a validação da coloração total, a criação de vértices e

arestas e a apresentação passo a passo da coloração total para classes de grafos.

No diretório *contexts* concentram-se as informações relacionadas ao grafo e à sua visualização, as quais são disponibilizadas para todos os componentes da interface. Dessa forma, os componentes podem acessar e utilizar dados como a matriz de adjacência, a coloração total quando disponível, informações de *layout*, a opção de *menu* selecionada pelo usuário, entre outros estados compartilhados da aplicação.

No diretório *lib* encontra-se o núcleo da aplicação, onde estão implementadas as classes que representam os grafos utilizados em todo o sistema, incluindo sua matriz de adjacência e, quando disponível, a coloração total. Além disso, esse diretório reúne os algoritmos responsáveis pela geração das matrizes de adjacência das classes de grafos consideradas (caminhos, ciclos e grafos completos), bem como os algoritmos associados à coloração dessas estruturas.

A Figura 28 apresenta o diagrama de classes do núcleo da aplicação. A classe abstrata *Graph* serve como base tanto para a classe abstrata *GraphClass*, responsável pelo comportamento de uma classe de grafos, quanto para a classe *FreeGraph*, responsável pelo comportamento de grafos genéricos. Essa interface define os atributos *matrix*, que representa a matriz de adjacência do grafo, e *totalColoring*, que corresponde a um mapeamento no qual as chaves são os elementos do grafo e os valores indicam as cores atribuídas a cada elemento.

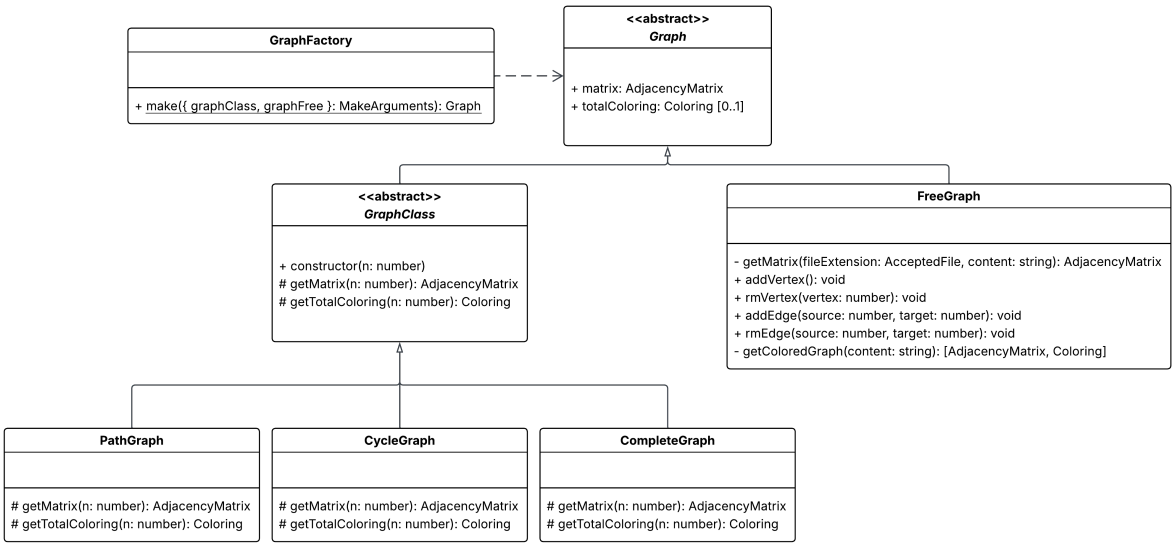


Figura 28: Diagrama de classe do núcleo da aplicação do Total Color.

A classe abstrata *GraphClass* é definida a partir de um padrão de projeto denominado *Template Method*, o qual estabelece, na classe base, a estrutura geral de um algoritmo e permite que as classes derivadas implementem etapas específicas dessa estrutura de acordo com seu con-

texto. Na *GraphClass*, essa estrutura é definida no método *constructor*, que invoca os métodos *getMatrix* e *getTotalColoring*, responsáveis, respectivamente, por obter a matriz de adjacência do grafo pertencente à classe e sua coloração total. As classes *PathGraph*, *CycleGraph* e *CompleteGraph* estendem a classe *GraphClass* e implementam, cada uma delas, os métodos *getMatrix* e *getTotalColoring*.

A classe *FreeGraph*, por meio de seu método construtor, determina se o grafo foi instanciado a partir de um arquivo `.txt` contendo uma coloração previamente definida ou não. No primeiro caso, é invocado o método *getColoredGraph*, o qual retorna tanto a matriz de adjacências quanto o mapeamento correspondente à coloração. Caso contrário, é chamado o método *getMatrix*, que retorna apenas a matriz de adjacências do grafo, independentemente de sua origem ser um arquivo `.txt` ou `.g6`. Além disso, os métodos *addVertex*, *rmVertex*, *addEdge* e *rmEdge* viabilizam a interação do usuário com o grafo, permitindo a adição e a remoção de vértices e arestas durante o processo de edição.

A classe *GraphFactory* é definida a partir de um padrão de projeto denominado *Factory Method*, o qual estabelece, na classe base, uma interface para a criação de objetos. Na *GraphFactory*, essa interface é responsável por instanciar um objeto do tipo *Graph*, seja ele uma instância de *GraphClass*, por meio de suas subclasses, ou um objeto da classe *FreeGraph*. Essa interface de criação é realizada por meio do método *make*.

Dada a estrutura do núcleo da aplicação, outra parte fundamental corresponde aos algoritmos de coloração total implementados pelas classes *PathGraph*, *CycleGraph* e *CompleteGraph*. Esses algoritmos são diretamente baseados nas demonstrações apresentadas na Seção 2.2 para cada uma dessas classes de grafos.

O Algoritmo 4 apresenta o procedimento de coloração total de um grafo caminho, implementado pelo método *getTotalColoring* da classe *PathGraph*. Como entrada, o algoritmo recebe a ordem do grafo, denotada por n, e percorre todos os elementos do grafo caminho. A atribuição de cores segue uma sequência cíclica entre 0 e 2, variando de acordo com o índice do elemento considerado. Quando o índice é par, o elemento corresponde a um vértice; quando é ímpar, corresponde a uma aresta. Após essa identificação, é realizada a atribuição da cor ao respectivo elemento. Por fim, o algoritmo retorna o mapeamento correspondente à coloração total do grafo.

O Algoritmo 5 apresenta o procedimento de coloração total de um grafo ciclo, implementado pelo método *getTotalColoring* da classe *CycleGraph*. Como entrada, o algoritmo recebe a

Algoritmo 4: Método *getTotalColoring* da classe PathGraph

```
Entrada: Ordem do grafo (n)
Saída: Mapeamento entre os elementos do grafo e as cores a eles atribuídas
coloracao ← new Map()
Para indice ← 0 até 2n − 2
    cor ← mod(indice, 3)
    Se mod(indice, 2) = 0 então
        vertice ← 'indice/2'
        coloracao.set(vertice, cor)
    Fim-Se
    Senão
        aresta ← '(indice − 1)/2  (indice + 1)/2'
        coloracao.set(aresta, cor)
    Fim-Se
Fim-Para
Retorne coloracao
```

ordem do grafo, denotada por n, e percorre todos os elementos do grafo ciclo, com exceção da aresta $v_{n-1}v_0$, atribuindo cores de forma análoga ao Algoritmo 4. A coloração da aresta $v_{n-1}v_0$ é realizada posteriormente, por conveniência, de modo que o índice 0 apareça no início da chave correspondente no mapeamento.

Após a atribuição inicial de cores aos elementos do grafo ciclo, tanto nos casos em que $\text{mod}(\text{indice}, 3) = 1$ quanto em que $\text{mod}(\text{indice}, 3) = 2$, ocorrem conflitos de coloração. Dessa forma, é realizada uma etapa de correção como apresentado no Teorema 3. Por fim, o algoritmo retorna o mapeamento correspondente à coloração total do grafo.

O Algoritmo 6 apresenta o procedimento de coloração total de um grafo completo, implementado pelo método *getTotalColoring* da classe *CompleteGraph*. Como entrada, o algoritmo recebe a ordem do grafo, denotada por n, e verifica se n é par. Nesse caso, utiliza-se como base o grafo completo de ordem ímpar imediatamente superior. Em seguida, todos os elementos do grafo são percorridos e coloridos de acordo com a demonstração para o caso ímpar apresentada no Teorema 4.

Posteriormente, quando n é par, o vértice adicional introduzido no início do algoritmo, bem como as arestas a ele incidentes, são removidos. Por fim, o algoritmo retorna o mapeamento correspondente à coloração total do grafo.

Algoritmo 5: Método *getTotalColoring* da classe CycleGraph

Entrada: Ordem do grafo (n)
Saída: Mapeamento entre os elementos do grafo e as cores a eles atribuídas
coloracao $\leftarrow$ *new* $Map()$
indice $\leftarrow 0$
Para *indice* **até** $2n-2$
  cor $\leftarrow mod(indice, 3)$
  Se $mod(indice, 2) = 0$ **então**
    vertice $\leftarrow$ $'indice/2'$
    coloracao.set(vertice, cor)
  Fim-Se
  Senão
    aresta $\leftarrow$ $'(indice-1)/2 \quad (indice+1)/2'$
    coloracao.set(aresta, cor)
  Fim-Se
Fim-Para
cor $\leftarrow mod(indice, 3)$
aresta $\leftarrow$ $'0 \quad (indice+1)/2'$
coloracao.set(aresta, cor)
Se $mod(indice, 3) = 1$ **então**
  coloracao.set($'n-2'$, $'3'$)
  coloracao.set($'n-2 \quad n-1'$, $'1'$)
  coloracao.set($'n-1'$, $'2'$)
  coloracao.set($'0 \quad n-1'$, $'3'$)
Fim-Se
Senão se $mod(indice, 3) = 2$ **então**
  coloracao.set($'0 \quad n-1'$, $'3'$)
Fim-Se
Retorne *coloracao*

Algoritmo 6: Método *getTotalColoring* da classe CompleteGraph

Entrada: Ordem do grafo (n)
Saída: Mapeamento entre os elementos do grafo e as cores a eles atribuídas
par $\leftarrow mod(n,2) = 0$
numeroCromaticoTotal $\leftarrow n + (par\ ?\ 1 : 0)$
coloracao $\leftarrow new\ Map()$
Para $i \leftarrow 0$ **até** $numeroCromaticoTotal - 1$
    vertice $\leftarrow\ 'i'$
    cor $\leftarrow\ 'i'$
    coloracao.set(vertice, cor)
    Para $j \leftarrow 1$ **até** $(numeroCromaticoTotal - 1)/2$
        indiceEsquerdo $\leftarrow mod(i-j, numeroCromaticoTotal)$
        indiceDireito $\leftarrow mod(i+j, numeroCromaticoTotal)$
        aresta
        $\leftarrow\ 'min(indiceEsquerdo, indiceDireito)\ \ max(indiceEsquerdo, indiceDireito)'$
        coloracao.set(aresta, cor)
    Fim-Para
Fim-Para
Se *par* **então**
    indiceAuxiliar $\leftarrow numeroCromaticoTotal - 1$
    verticeAuxiliar $\leftarrow\ 'indiceAuxiliar'$
    coloracao.delete(verticeAuxiliar)
    Para $i \leftarrow 0$ **até** $indiceAuxiliar - 1$
        arestaAuxiliar $\leftarrow\ 'i\ \ indiceAuxiliar'$
        coloracao.delete(arestaAuxiliar)
    Fim-Para
Fim-Se
Retorne *coloracao*

Capítulo 5

Conclusão

O *Total Color* tem como objetivo permitir a visualização da coloração total em determinadas classes de grafos, bem como promover a aprendizagem ativa por meio da interação e da experimentação do usuário. Para a sua construção, foi necessário, inicialmente, estabelecer conceitos fundamentais da Teoria dos Grafos e dos fundamentos da coloração de grafos, com ênfase na coloração total.

Dado que a complexidade computacional do problema da coloração total, no caso geral, pertence à classe dos problemas NP-difíceis [Sánchez-Arroyo, 1989], torna-se conveniente restringir o estudo a classes específicas de grafos. Nesse contexto, ao longo deste trabalho, foram demonstradas a coloração total e o número cromático total das classes dos grafos caminhos, ciclos e completos.

A partir dessa abordagem, o *Total Color* permite ao usuário interagir com essas classes de grafos, conhecer seus respectivos números cromáticos totais e visualizar, de forma progressiva, a coloração dos grafos. Além disso, o próprio usuário pode atribuir cores ao grafo, tendo como referência o número cromático total apresentado pela aplicação. Complementarmente, o sistema possibilita a exportação de um arquivo `.txt` contendo a coloração atribuída, seja ela definida pelo usuário ou gerada automaticamente pela aplicação.

De forma adicional ao propósito principal da aplicação, foi desenvolvido um pacote no *npm* que possibilita a conversão entre matriz de adjacência e o formato `.g6`. Tal funcionalidade foi incorporada ao *Total Color*, permitindo a importação e a exportação de grafos no formato *graph6*. Outra contribuição relevante é o modo livre da aplicação, que possibilita ao usuário criar um grafo do zero, adicionando vértices e arestas e atribuindo cores, bem como importar um grafo previamente colorido, com o objetivo de analisá-lo e interagir com sua coloração de forma visual.

Visando ampliar o alcance e a acessibilidade da ferramenta, a aplicação também foi projetada para funcionar em dispositivos móveis, que geralmente possuem telas reduzidas, permitindo ao usuário um engajamento rápido e facilitado com o *Total Color*. Posteriormente, o usuário pode optar pelo uso de dispositivos com telas maiores, o que possibilita uma interação

mais detalhada ou a visualização de grafos com uma maior quantidade de vértices e arestas.

No que se refere ao projeto e à manutenção do *Total Color*, sua estruturação permite ao desenvolvedor a inclusão de novas classes de grafos de forma facilitada, em função dos padrões de projeto adotados e das classes implementadas. Além disso, a organização dos componentes responsáveis pela interação com o usuário favorece o desenvolvimento de novas funcionalidades, possibilitando a ampliação da aplicação em trabalhos futuros.

Quanto ao uso de Inteligência Artificial no desenvolvimento deste trabalho, esta foi empregada como ferramenta de apoio, principalmente para a realização de ajustes gramaticais e aprimoramento da clareza textual, bem como para auxiliar na compreensão da documentação e do funcionamento das bibliotecas utilizadas na construção da solução proposta.

Por fim, como trabalhos futuros, podem ser consideradas a adição de novas classes de grafos ao *Total Color*, como, por exemplo, grafos circulares, grafos de *Kneser* e grafos *split*. Além disso, pode-se estender ao modo classes as funcionalidades de modificação do grafo já disponíveis no modo livre, bem como permitir a imposição de restrições adicionais à coloração, além daquelas previstas na definição clássica de coloração total.

Referências Bibliográficas

Mehdi Behzad. *Graphs and their chromatic numbers*. Michigan State University, 1965.

Eliza Bobek and Barbara Tversky. Creating visual explanations improves learning. *Cognitive research: principles and implications*, 1:1–14, 2016.

John Adrian Bondy and Uppaluri Siva Ramachandra Murty. *Graph theory*. Springer Publishing Company, Incorporated, 1982.

Rowland Leonard Brooks. On colouring the nodes of a network. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 37, pages 194–197. Cambridge University Press, 1941.

Christiane Neme Campos. *O problema da coloração total em classes de grafos.* PhD thesis, University of Campinas, Brazil, 2006.

Professor Cayley. On the colouring of maps. In *Proceedings of the Royal Geographical Society and Monthly Record of Geography*, volume 1, pages 259–261. JSTOR, 1879.

Yi Chen, Zeli Guan, Rong Zhang, Xiaomin Du, and Yunhai Wang. A survey on visualization approaches for exploring association relationships in graph data. *Journal of Visualization*, 22:625–639, 2019.

Randy Connolly, Ricardo Hoar, Soumen Mukherjee, and Arup Kumar Bhattacharjee. *Fundamentals of web development*. Pearson, 2015.

Kris Coolsaet, Sven D'hondt, and Jan Goedgebeur. House of graphs 2.0: A database of interesting graphs and more. *Discrete Applied Mathematics*, 325:97–107, 2023.

Resul Das and Mucahit Soylu. A key review on graph data science: The power of graphs in scientific studies. *Chemometrics and Intelligent Laboratory Systems*, 240:104896, 2023.

Celina M. H. de Figueiredo. Total colouring. In Lowell W. Beineke, Martin Charles Golumbic, and Robin J. Wilson, editors, *Topics in Algorithmic Graph Theory*, chapter 3, pages 52–67. Cambridge University Press, 2021.

H. N. de Ridder et al. Information System on Graph Classes and their Inclusions (ISGCI). https://www.graphclasses.org, 2025.

R. Diestel. *Graph Theory: 3th edition*. Springer Publishing Company, Incorporated, 2005.

Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1736.

Maria Evagorou, Sibel Erduran, and Terhi Mäntylä. The role of visual representations in scientific practices: from conceptual understanding and knowledge generation to 'seeing'how science works. *International journal of Stem education*, 2:1–13, 2015.

Max Franz, Christian T Lopes, Dylan Fong, Mike Kucera, Manfred Cheung, Metin Can Siper, Gerardo Huck, Yue Dong, Onur Sumer, and Gary D Bader. Cytoscape. js 2023 update: a graph theory library for visualization and analysis. *Bioinformatics*, 39(1):btad031, 2023.

Abdón Sánchez-Arroyo. Determining the total colouring number is np-hard. *Discret. Math.*, 78 (3):315–319, 1989.

Alessandra Hendi dos Santos. Um estudo epistemológico da visualização matemática: o acesso ao conhecimento matemático no ensino por intermédio dos processos de visualização. Master's thesis, Universidade Federal do Paraná, 2014.

Vadim G Vizing. On an estimate of the chromatic class of a p-graph. *Diskret analiz*, 3:25–30, 1964.

Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

Hian Poh Yap. *Total Colourings of Graphs*. Springer, 1996.