UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS FÍSICAS E MATEMÁTICAS
BACHARELADO EM MATEMÁTICA

Renan Rabelo Goularti

Teoria dos Grafos Introdutória: com implementações computacionais

Florianópolis
2023

Renan Rabelo Goularti

Teoria dos Grafos Introdutória: com implementações computacionais

Trabalho de Conclusão de Curso de Graduação em Bacharelado em Matemática do Centro de Ciências Físicas e Matemáticas da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Matemática.
Orientador: Prof. Gilles Gonçalves de Castro

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Goularti, Renan Rabelo
Teoria dos Grafos Introdutória : com implementações computacionais / Renan Rabelo Goularti ; orientador, Gilles Gonçalves de Castro, 2023.
39 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro de Ciências Físicas e Matemáticas, Graduação em Matemática - Bacharelado, Florianópolis, 2023.

Inclui referências.

1. Matemática - Bacharelado. 2. Grafos. 3. Computação. 4. Coloração. I. de Castro, Gilles Gonçalves. II. Universidade Federal de Santa Catarina. Graduação em Matemática - Bacharelado. III. Título.

Renan Rabelo Goularti

Teoria dos Grafos Introdutória: com implementações computacionais

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Matemática e aprovado em sua forma final pelo Curso de Bacharelado em Matemática.

Florianópolis, 28 de novembro de 2023.

Documento assinado digitalmente
Felipe Lopes Castro
Data: 15/12/2023 16:46:05-0300
CPF: ***.060.710-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Felipe Lopes Castro, Dr.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente
Gilles Goncalves de Castro
Data: 12/12/2023 10:03:59-0300
CPF: ***.977.489-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Gilles Gonçalves de Castro
Orientador
Instituição UFSC

Prof. Daniel Gonçalves, Dr.
Avaliador
Instituição UFSC

Prof. Natã Machado, Dr.
Avaliador
Instituição UFSC

Para Alcides, meu amado pai, e Giani, minha amada mãe.

AGRADECIMENTOS

Agradeço ao meu pai Alcides e minha mãe Giani, pois sem a educação, apoio e ensinamentos ao longo da vida eu jamais estaria onde estou. Agradeço também a todas as brasileiras e brasileiros que lutaram por um país mais justo e uma educação pública, a existência desta universidade é fruto de suas lutas. À minha namorada Sofia, que está ao meu lado desde o ensino médio e me acompanha até hoje me dando forças e carinho em todos os momentos da graduação. Aos meus amigos, que moldaram meu caráter e trouxeram alegria nos momentos de fragilidade. Aos meus bons professores, que me inspiram a seguir na docência.

‘‘Diagramas são para os geômetras o que um tabuleiro é para um enxadrista: recursos visuais, úteis mas não indispensáveis’’

(TRUDEAU, 1994)

RESUMO

Grafos são as estruturas matemáticas que codificam abstratamente um conjunto munido de uma relação simétrica. Neste trabalho iremos trazer a definição de grafo com a linguagem de conjuntos, definir caminhos, distâncias, conectividade, árvores e colorações. Diversos resultados iniciais são demonstrados, apresentando conexões entre as grandezas que um grafo possui. Ao final do trabalho consta a implementação computacional em linguagem Julia de algoritmos abordados no texto.

Palavras-chave: Grafos. Algoritmos. Coloração.

ABSTRACT

Graphs are mathematical structures that abstractly encode a set with a symmetric relationship. In this work we will bring the definition of a graph with the set language, define paths, distances, connectivity, trees and colorings. Several initial results are demonstrated, presenting connections between the quantities that a graph has. At the end of the work there is the computational implementation in Julia language of algorithms covered in the text.

Keywords: Graphs. Algorithms. Coloring.

SUMÁRIO

1 INTRODUÇÃO 10
2 TEORIA 12
2.1 GRAFOS E VIZINHANÇAS 12
2.2 CAMINHOS E CICLOS 15
2.3 DISTÂNCIAS 16
2.4 CONECTIVIDADE 17
2.5 ÁRVORES E FLORESTAS 21
2.6 HOMOMORFISMOS E ISOMORFISMOS 24
2.7 COLORAÇÕES 25
2.8 OUTROS TIPOS DE GRAFOS 30
3 IMPLEMENTAÇÃO COMPUTACIONAL 32
3.1 REPRESENTAÇÃO DE UM GRAFO 32
3.2 CÓDIGO 32
4 CONSIDERAÇÕES FINAIS 37
REFERÊNCIAS 38

1 INTRODUÇÃO

Em um estado temos centenas de cidades, onde cada uma pode ou não compartilhar fronteira. Será que é possível passar por todas as cidades uma única vez em uma única viagem de carro? Como que sequer podemos abordar este problema?

Em uma análise mais abstrata, percebemos que o que importa é se podemos sair de uma cidade e chegar em outra. Não nos importa o tamanho da cidade, o seu relevo ou a qualidade da estrada, queremos uma lista que descreva ‘‘Cidade A é vizinha da cidade B’’ para planejarmos nossa viagem. Neste caso a relação de vizinhança entre cidades é simétrica, ou seja, se A é vizinha de B então B é vizinha de A. Esta estrutura que descrevemos nada mais é que um grafo.

Grafos são a junção de um conjunto abstrato, o conjunto dos vértices, com um conjunto que indica quais vértices estão relacionadas entre si, o conjunto das arestas. A relação entre dois vértices é a relação de vizinhança, que é simétrica. A partir desta definição podemos definir caminhos entre vértices, distância entre vértices, conectividade de vértices entre outras coisas.

Um sistema que envolve agentes e uma relação simétrica entre eles pode ser descrito por um grafo. O exemplo das cidades em um estado foi um dentre as diversas possibilidades: relações de amizade em uma comunidade de pessoas, conexões de componentes eletrônicos ou até estações de linha de metrô.

A partir da leitura dos capítulos iniciais do livro *Graph Theory* de *Reinhard Diestel* [2] com o acompanhamento do livro *Graph Theory* de *William Thomas Tutte* [4] e encontros semanais com o orientador, este trabalho dedica-se a enunciação de algumas definições importantes e demonstração de resultados gerais sobre grafos. Acoplado a isto, um capítulo inteiro é dedicado à implementação computacional em linguagem Julia (JuliaLang) de algumas proposições, inspirado em códigos descritos no livro *Algoritmos: Teoria e Prática* de *Thomas Cormen* [1].

Este trabalho está dividido em duas partes. A primeira é teórica e composta por oito seções, enquanto a segunda aborda as aplicações computacionais dos resultados da primeira. Na primeira seção, discutimos as definições iniciais de grafos e a relação de vizinhança, onde definimos o que é um grafo e as grandezas relacionadas à vizinhança de um vértice. O segundo capítulo dedica-se aos resultados sobre caminhos (ou sequência de vértices vizinhos) e ciclos em grafos, mostrando a existência de caminhos com certas propriedades. Em seguida, abordamos distâncias e conectividade, tópicos necessários para compreender o quão "conectados"os vértices estão entre si, ou seja, quantas formas podemos sair de um vértice e chegar a outro por meio de um caminho.

Posteriormente, falamos sobre árvores, que são casos especiais de grafos com diversas aplicações e servem para demonstrar alguns resultados. Homomorfismos são importantes, pois mostram que os resultados provados não se preocupam com a natureza dos vértices, mas sim

com sua estrutura de vizinhança. Após isso, discutimos o significado de colorir um grafo e apresentamos alguns métodos para obter a coloração de um grafo, incluindo alguns resultados originais do autor na parte de reduções. Por fim, tratamos de diferentes tipos de grafos e suas aplicações.

2 TEORIA

A definição de grafo é bastante simples e depende apenas de conceitos da teoria de conjuntos. Diversas demonstrações e definições neste capítulo fazem uso desta linguagem, mas as vezes é necessário utilizar argumentos visuais a partir do diagrama de um grafo.

2.1 GRAFOS E VIZINHANÇAS

Definição 2.1. (Conjunto das partes)

Seja X um conjunto e n um natural. O conjunto formado por todos os subconjuntos de X de cardinalidade n é denotado por $\mathcal{P}_n(X)$.

Definição 2.2. (Grafo e subgrafo)

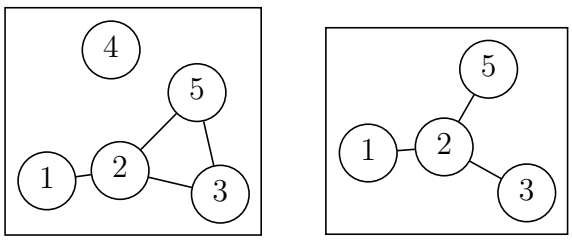
Seja V um conjunto finito e $E \subset \mathcal{P}_2(V)$. O par $G = (V, E)$ é chamado de **grafo** G com o conjunto de **vértices** V e o conjunto de **arestas** E. A **ordem** de G, denotada por $|G|$, é a quantidade de vértices de G. Um grafo deve conter ao menos um vértice.

Um grafo $G_1 = (V_1, E_1)$ é **subgrafo** de G, denotado por $G_1 \leq G$, se $V_1 \subset V$ e $E_1 \subset E$, ou seja, os vértices e arestas de G_1 são vértices e arestas de G. Dizemos que G_1 é um subgrafo **próprio** de G se $G_1 \leq G$ e $G_1 \neq G$. Dizemos que G_1 **gera** (ou é **gerador**) G se $V_1 = V$, ou seja, subgrafos geradores são aqueles que contém todos os vértices do grafo maior.

O grafo G_1 é **maximal** em G em relação a uma propriedade se não é subgrafo próprio de outro subgrafo de G que possui a mesma propriedade em G. O grafo G_1 é **minimal** em G em relação a uma propriedade se nenhum subgrafo próprio de G_1 dele possui a mesma propriedade em G.

As imagens abaixo são **diagramas** de um grafo e neles representamos os vértices por círculos etiquetados e as arestas por curvas conectando os vértices. Não há restrição quando a trajetória que uma aresta forma, podendo interseccionar outras arestas. É comum não etiquetar os vértices quando a natureza do vértice não importa para a representação.

Figura 1 – Exemplo de grafo e subgrafo



Fonte: Autor

À esquerda o grafo com vértices $\{1, 2, 3, 4, 5\}$ e arestas $\{\{1, 2\}, \{2, 3\}, \{2, 5\}, \{3, 5\}\}$ e à direita um subgrafo com vértices $\{1, 2, 3, 5\}$ e arestas $\{\{1, 2\}, \{2, 3\}, \{2, 5\}\}$. Para evitar

repetição, daqui em diante os símbolos G, V e E *sempre* representarão um grafo na forma $G = (V, E)$. O mesmo ocorre para G_n, V_n e E_n com $G_n = (V_n, E_n)$.

Definição 2.3. (Vizinhança de um vértice)

Dois vértices u e v são **vizinhos** em G se $\{u, v\} \in E$. Segue da definição de grafos que um vértice não pode ser vizinho de si mesmo. A aresta $\{u, v\}$, denotada por uv, **incide** sobre suas **extremidades** u e v.

A **vizinhança** de v em G, denotado por $\mathcal{V}_G(v)$, é o conjunto de todos os vizinhos de v em G. Similarmente, a vizinhança de $V_1 \subset V$ em G, denotado por $\mathcal{V}_G(V_1)$, é a união das vizinhanças de seus elementos em G.

$$\mathcal{V}_G(v) = \{u \in V \ : \ uv \in E\} \qquad \mathcal{V}_G(V_1) = \bigcup_{v \in V_1} \mathcal{V}_G(v) \tag{1}$$

O conjunto das arestas de v em G, denotado por $\mathcal{E}_G(v)$, é o conjunto de todas as arestas em G que incidem sobre v. Similarmente, o conjunto das arestas de $V_1 \subset V$ em G, denotado por $\mathcal{E}_G(V_1)$, é a união dos conjuntos das arestas de seus elementos em G.

$$\mathcal{E}_G(v) = \{e \in E \ : \ v \in e\} \qquad \mathcal{E}_G(V_1) = \bigcup_{v \in V_1} \mathcal{E}_G(v) \tag{2}$$

O **grau** de um vértice v em G, denotado por $|v|_G$, é a cardinalidade de sua vizinhança em G. O **maior grau** de G, denotado por $\Delta(G)$, é o maior grau entre todos os vértices de G. O **menor grau** de G, denotado por $\delta(G)$, é o menor grau entre todos os vértices de G. A **densidade** de G, denotado por $\varepsilon(G)$, é a razão entre a quantidade de arestas e vértices de G. Um grafo k-**regular** é aquele que todos os seus vértices tem grau k.

$$|v|_G = |\mathcal{V}_G(v)| \qquad \Delta(G) = \max_{v \in V}\{|v|_G\} \qquad \delta(G) = \min_{v \in V}\{|v|_G\} \qquad \varepsilon(G) = \frac{|E|}{|V|} \tag{3}$$

Na figura Figura 1 temos que a vizinhança do vértice 2 são os vértices $\{1, 3, 5\}$, enquanto as arestas incidentes são $\{\{1, 2\}, \{2, 3\}, \{2, 5\}\}$.

A representação gráfica de um grafo não precisa ser necessariamente plana, basta imaginar o grafo formado pelos vértices e arestas de um poliedro.

A relação de vizinhança em um grafo é induzida a partir do conjunto das arestas, mas o contrário poderia ser feito. Se temos uma relação simétrica no conjunto dos vértices podemos induzir o conjunto das arestas a partir dela. Na Seção 2.8 discutiremos outros possíveis tipos de grafos e suas definições.

Definição 2.4. (Operações com grafos)

Seja $D \subset \mathcal{P}_2(V)$. O grafo $G - D$ é aquele com arestas $E \setminus D$ e vértices V. O grafo $G + D$ é aquele com arestas $E \cup D$ e vértices V. Para $e \in D$ denotaremos $G - \{e\}$ por $G - e$ e $G + \{e\}$ por $G + e$.

Seja $U \subset V$. O grafo $G - U$ é aquele com vértices $V - U$ e arestas $E - \mathcal{E}_G(U)$. Para $v \in U$ denotaremos $G - \{v\}$ por $G - v$.

O grafo **induzido** por $U \subset V$ em G, denotado por $\mathcal{I}_G(U)$, é o grafo com vértices U e arestas $\mathcal{E}_G(U) \cap \mathcal{P}_2(U)$, ou seja, as arestas são aquelas de G que ambas as extremidades estão em U. Claramente $\mathcal{I}_G(U) \leq G$.

Proposição 2.5. (Paridade dos vértices de grau ímpar)

A quantidade de vértices de grau ímpar em G é par.

Demonstração:

O somatório $\sum_{v \in V} |v|_G = 2|E|$ contabiliza duas vezes a mesma aresta, uma vez em cada ponta, portanto o somatório tem valor par. A quantidade de vértices de grau ímpar ser ímpar leva à um absurdo, pois o somatório seria ímpar. Logo a quantidade de vértices de grau ímpar em G é par.

■

Esta primeira demonstração evidencia o caráter combinatório desta teoria. Poderíamos discutir formalmente o porquê cada aresta é contada duas vezes, mas utilizamos um argumento mais intuitivo para evitar o uso excessivo de notações e conceitos abstratos que podem causar confusão.

Proposição 2.6. (Relação entre menor grau e densidade)

Se $|E| \geq 1$ então existe um grafo H tal que $H \leq G$ e $\delta(H) > \varepsilon(H) \geq \varepsilon(G)$.

Demonstração:

Se $\delta(G) > \varepsilon(G)$, defina $H \coloneqq G$ e está demonstrada a proposição. Suponha que exista $v_1 \in V$ tal que $|v_1|_G \leq \varepsilon(G)$. Defina $G_1 \coloneqq G - v_1$ e note que $\varepsilon(G_1) \geq \varepsilon(G)$ pois

$$\varepsilon(G_1) - \varepsilon(G) = \frac{|E| - |v_1|_G}{|V| - 1} - \frac{|E|}{|V|} = \frac{|V| \cdot |E| - |V| \cdot |v_1|_G - |V| \cdot |E| + |E|}{|V|^2 - |V|} \tag{4}$$

$$= \frac{|E| - |V| \cdot |v_1|_G}{|V|^2 - |V|} \geq \frac{|E| - |V| \cdot \varepsilon(G)}{|V|^2 - |V|} = 0 \tag{5}$$

Construa uma sequência de grafos da seguinte maneira: tome um vértice v_{i+1} de G_i tal que $|v_{i+1}|_{G_i} \leq \varepsilon(G_i)$, caso não exista então defina $H \coloneqq G_i$ e termine a sequência de grafos. Eventualmente a sequência deve terminar, visto que a quantidade de vértices é finita.

Note que $G \geq G_1 \geq \cdots \geq H$, $\delta(H) > \varepsilon(H)$ e $\varepsilon(H) \geq \cdots \geq \varepsilon(G_1) \geq \varepsilon(G)$. Portanto $\delta(H) > \varepsilon(H) \geq \varepsilon(G)$. Além disso, nenhum dos grafos na sequência é vazio, caso o contrário seria necessário que o grafo anterior na sequência fosse constituído de apenas um vértice com densidade nula, mas a densidade deste grafo deve ser maior que a densidade de G, que não é nula.

■

2.2 CAMINHOS E CICLOS

A definição de caminho é basilar para estudarmos grafos. Um caminho nos diz como ‘‘sair de um vértice e chegar em outro passeando pelas arestas’’. No problema proposto inicialmente na introdução deste trabalho o que queríamos encontrar é um caminho que passa por todos os vértices sem repetições.

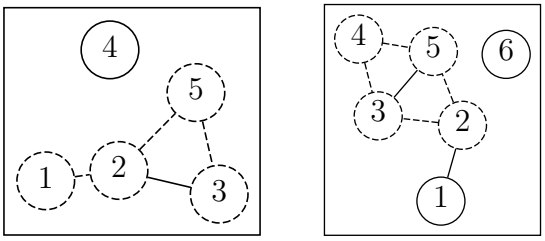
Definição 2.7. (Caminho e ciclo)

Uma sequência de vértices $(v_n)_a^b = (v_a, v_{a+1}, \ldots, v_b)$ é um **caminho** em G se $v_i \neq v_j$ e $v_iv_{i+1} \in E$ para $a \leq i \neq j \leq b$, ou seja, em um caminho vértices não se repetem e vértices consecutivos são vizinhos em G. A sequência é um **ciclo** se $v_i \neq v_j$ para $a < i \neq j < b$, $v_kv_{k+1} \in E$ para $a \leq k \leq b$ e $v_a = v_b$, ou seja, no ciclo o primeiro e último vértice são iguais. As **extremidades** de $(v_n)_a^b$ são os vértices iniciais e finais enquanto seu **interior** são os outros vértices. Dois caminhos são **independentes** se o interior de um caminho não contém nenhum vértice do outro e vice-versa. O grafo gerado por $(v_n)_a^b$, denotado por $\mathcal{G}((v_n)_a^b)$, é o grafo com vértices v_i e arestas v_iv_{i+1}. O **comprimento** de $(v_n)_a^b$, denotado por $|(v_n)_a^b| = b - a$, é quantidade de arestas do grafo gerado por ele. É necessário permitir a existência de caminhos unitários que contém apenas um vértice, e seu comprimento é nulo.

Para $u, v \in V$ o conjunto de todos os caminhos em G com vértice inicial u e vértice final v é denotado por $u - v$. Se $A = (v_n)_a^b$ e $B = (u_n)_c^d$ são caminhos independentes, os vértices v_a, v_b, u_c e u_d são todos distintos entre si e $v_bu_c \in E$ então a **concatenação** de A com B, denotada por $(v_n)_a^b + (u_n)_c^d$, é o caminho $(v_a, \ldots, v_b, u_c, \ldots, u_d)$.

O **giro** de G, denotado por $g(G)$, é o comprimento do menor ciclo contido em G. A **circunferência** de G, denotado por $circ(G)$, é o comprimento do maior ciclo em G.

Figura 2 – Em pontilhado exemplos de caminho e ciclo



Fonte: Autor

Na esquerda temos o caminho $(1, 2, 5, 3)$ e na direita o ciclo $(2, 3, 4, 5, 2)$.

Proposição 2.8. (Menor grau e comprimento de caminhos e ciclos)

Se $\delta(G) \geq 2$ então G contém um caminho de comprimento ao menos $\delta(G)$ e um ciclo de comprimento ao menos $\delta(G) + 1$.

Demonstração:

Seja $A = (v_n)_0^k$ o caminho de maior comprimento de G. Todos os vizinhos de v_k em G estão em A, pois caso o contrário poderíamos concatenar o vizinho que está fora ao final A e obter um caminho de comprimento maior. Portanto $k \geq |v_k|_G \geq \delta(G)$. Note que k é a quantidade de vértices de A diferentes de v_k.

Seja i o menor natural tal que v_i é vizinho de v_k em G, então $B = (v_n)_i^k + v_i = (v_n)_i^{k+1}$ é um ciclo que contém todos os vizinhos de v_k em G. Sabendo que $k - i$ é a quantidade de vértices distintos de v_k em B temos que $k - i \geq |v_k| \geq \delta(G)$, ou seja, $|B| = k - i + 1 \geq \delta(G) + 1$.

■

2.3 DISTÂNCIAS

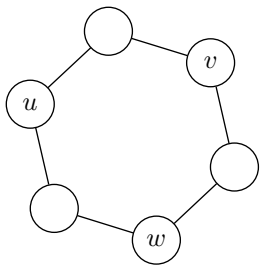
Na Geometria Euclidiana, a distância entre dois pontos é a quantidade mínima de unidades de comprimento que os separam. Na Teoria dos Grafos, a noção de distância é similar, mas ao invés de unidades queremos saber quantas arestas separam dois vértices.

Definição 2.9. (Distância entre vértices)

Sejam u e v vértices de G. A **distância** entre u e v em G, denotada por $d_G(u, v)$, é o comprimento do menor caminho $u - v$ em G. Caso tal caminho não exista então $d_G(u, v) = +\infty$. O **diâmetro** de G, denotado por $\text{diam}(G)$, é a maior distância finita entre dois vértices em G.

Se sempre existir um caminho[1] $u - v$ em G para qualquer $u, v \in V$ então $d_G(u, v)$ é uma métrica em V. Claramente d_G é não negativa e simétrica. A distância entre vértices é nula se, e somente se, os vértices são iguais (caminho unitário). A desigualdade triangular decorre do seguinte argumento: sejam $u, v, w \in V$, A o menor caminho $u - v$, B o menor caminho $u - w$ e C o menor caminho $w - v$. O comprimento de A não pode ser menor que o comprimento de $B + C$, pois A tem o menor comprimento. Portanto $d_G(u, v) \leq d_G(u, w) + d_G(w, v)$. A Figura 3 mostra um exemplo da desigualdade triangular.

Figura 3



Fonte: Autor

[1] Na Seção 2.4 vamos definir esta propriedade como ‘‘conectividade’’

Proposição 2.10. (Relação entre giro e diâmetro)

Se G contém um ciclo então $g(G) \leq 2 \cdot \text{diam}(G) + 1$.

Demonstração:

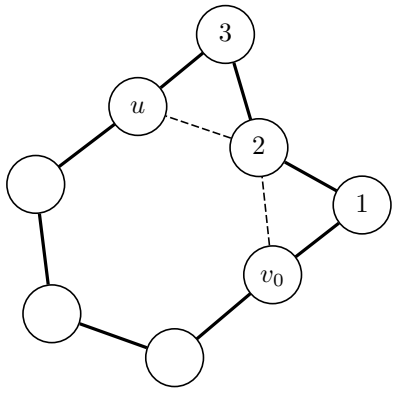
Suponha, por absurdo, que $g(G) > 2 \cdot \text{diam}(G) + 1$ e portanto $g(G) \geq 2 \cdot \text{diam}(G) + 2$. Seja $C = (v_n)_0^a$ o ciclo de menor comprimento de G e G_1 o grafo gerado por ele. Tome $u \coloneqq v_{\text{diam}(G)+1}$ e note que $d_{G_1}(v_0, u) = \text{diam}(G) + 1$. Em G, deve existir um caminho $A = (u_n)_0^k$ que conecta v_0 e u com comprimento menor ou igual a $\text{diam}(G)$, portanto A não pode estar inteiramente contido em C. Seja $i < k$ o maior índice tal que $u_i = v_j$, para algum j. Note que o ciclo abaixo tem comprimento menor que $2 \cdot \text{diam}(G) + 1$, o que é impossível pois $g(G) > 2 \cdot \text{diam}(G) + 1$.

$$(v_j, v_{j+1}, \ldots, v_{\text{diam}(G)+1}, u_{k-1}, u_{k-2}, \ldots, u_i) \tag{6}$$

■

A Figura 4 apresenta um exemplo da proposição 2.10. Note que o vértice 2 faz o papel de u_i e portanto o ciclo formado é $(2, 3, u, 2)$. Intuitivamente podemos pensar que se o menor ciclo C de um grafo G tem dois vértices v_0 e u com uma distância $d_C(v_0, u)$ maior que o diâmetro de G, então deve existir um caminho A com comprimento menor que $d_C(v_0, u)$ em G. Podemos utilizar este caminho A para formar um novo ciclo em G com comprimento menor que C.

Figura 4 – Em negrito o ciclo C e pontilhado o caminho A



Fonte: Autor

2.4 CONECTIVIDADE

A conectividade de um grafo nos diz o quão conectados estão os vértices entre si. De quantas maneiras diferentes podemos sair de um vértice e chegar em outro?

Definição 2.11. (Conectividade)

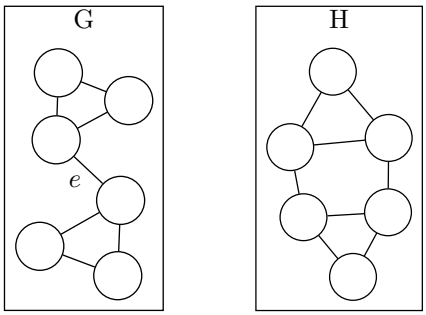
Os vértices $u, v \in V$ estão **conectados** em G se existir um caminho $u - v$ em G. Um grafo é dito **conectado** se todos os seus pares de vértices estão conectados. Por definição, todo grafo com apenas um vértice é conectado. Um **componente conectado** de G é um subgrafo conectado maximal.

Seja k um número natural. G é **k-conectado** se para todo conjunto de vértices $U \subset V$ com menos de k vértices temos que $G - U$ é conectado. G é **k-aresta-conectado** se para todo conjunto de arestas $D \subset E$ com menos de k arestas temos que $G - D$ é conectado.

A **conectividade** de G, denotado por $\kappa(G)$, é o maior k possível tal que G é k-conectado. A **conectividade por arestas** de G, denotado por $\lambda(G)$, é o maior k possível tal que G é k-aresta-conectado.

Observe os seguinte grafos. Note que para tornar G desconectado basta remover e, portanto $\lambda(G) = 1$. Por outro lado, no grafo H é necessário remover qualquer par de arestas para desconectar H, logo $\lambda(H) = 2$.

Figura 5 – Na esquerda G e na direita H



Fonte: Autor

Definição 2.12. (Grafo completo e clique)

Um grafo G é **completo** se $E = \mathcal{P}_2(V)$, ou seja, o grafo contém todas as possíveis arestas. Um **n-clique** de G é um subgrafo completo de G com n vértices. O **clique** de G, denotado por $\Gamma(G)$, é a ordem do maior n-clique de G.

Descobrir o clique de um grafo não é uma tarefa simples. Para grafos pequenos é fácil, mas para grafos com muitos vértices e arestas o problema se torna exponencialmente mais complicado. Até que se prove ao contrário, não é conhecido um algoritmo ‘‘simples’’[2] que resolva este problema.

Proposição 2.13. (Conectividade e subgrafos)

Sejam G e G_1 conectados, $e \in E$ e $G_1 \leq G$ tal que $e \in E_1$. Se $G - e$ é desconectado então $G_1 - e$ também é.

[2] Entre aspas pois o conceito de ‘‘simplicidade’’ de um algoritmo envolve uma teoria que não será abordada aqui, para mais informação confira o livro de Michael Sipser *Introduction to the Theory of Computation*[3]

Demonstração:

Suponha, por absurdo, que $G_1 - e$ é conectado. Sejam $u, v \in V$ e tome o caminho A que conecta u e v em G. Se e não está no grafo gerado por A então o caminho permanece intacto em $G - e$ garantindo que u e v estão conectados em $G - e$. Suponha portanto que e está no grafo gerado por A. Se G_1 contém e então suas extremidades estão em G_1, sejam v_1 e v_2 estas extremidades. Como $G_1 - e$ é conectado então existe um caminho em $G_1 - e$ que conecta v_1 e v_2, seja B este caminho. Claramente B conecta v_1 e v_2 em $G - e$ portanto mesmo que removido e de A ainda é possível inserir B dentro de A de tal maneira que o caminho ‘‘contorne’’ e, garantindo que u e v estão conectados. Sendo assim $G - e$ deve ser conectado, o que é um absurdo.

■

Proposição 2.14. (Conectividade e caminhos)

Se para quaisquer $u, v \in V$ distintos existem k caminhos $u - v$ independentes em G então G é k-conectado e k-aresta-conectado.

Demonstração:

Sejam $U \subset V$ um subconjunto de vértices tal que $|U| < k$, $D \subset E$ um conjunto de arestas tal que $|D| < k$. Considere $V_1 = V - U$ e note que se $|V_1| \leq 1$ então G_1 é conectado, portanto G é k-conectado. Suponha que $|V_1| > 1$ e tome $u, v \in V_1$ distintos. Um vértice $w \in U$ não pode pertencer ao mesmo tempo a dois caminhos independentes $u - v$ em G pois o interior dos caminhos são distintos. Pelo princípio da casa dos pontos isto garante que pelo menos um dos k caminhos será preservado caso U seja removido de G, e portanto G_1 é conectado e isto implica na k-conectividade de G. O mesmo argumento em D pode ser feito para demonstrar a k-aresta-conectividade de G.

■

Proposição 2.15. (Enumeração de grafos conectados)

Para todo grafo conectado G existe uma enumeração $\{v_1, v_2, \ldots, v_n\}$ de V tal que $\mathcal{I}_G(V_i)$ é conectado para todo i tal que $V_i = \{v_1, \ldots, v_i\}$.

Demonstração:

Escolha $v_1 \in V$ e note que $\mathcal{I}_G(V_1)$ é conectado. Suponha indutivamente que exista uma enumeração $V_i = \{v_1, \ldots, v_i\}$ tal que $\mathcal{I}_G(V_i)$ é conectado e escolha um vértice $u \in V - V_i$. Pela conectividade de G, podemos garantir que existe um caminho $v_1 - u$ em G. Escolha o maior índice j deste caminho tal que v_j esteja em V_i, portanto v_{j+1} está em $V - V_i$. Defina $v_{i+1} \coloneqq v_{j+1}$ e note que $\mathcal{I}_G(V_{i+1})$ é conectado pois v_{i+1} faz vizinhança com v_j que está em V_i. Por indução em i a demonstração está completa.

■

Proposição 2.16. (Relação entre conectividades e menor grau)

Para todo grafo G temos que $\kappa(G) \leq \lambda(G) \leq \delta(G)$.

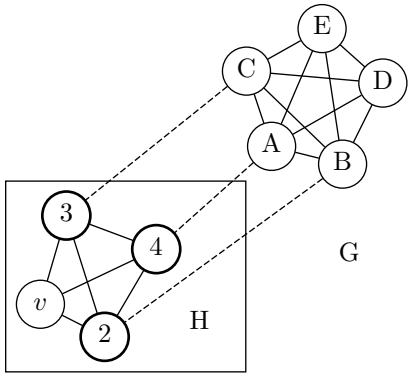
Demonstração:

Seja $v \in V$ tal que $|v|_G = \delta(G)$. Claramente os grafos $H \coloneqq G - \mathcal{V}_G(v)$ e $K \coloneqq G - \mathcal{E}_G(v)$ são desconectados, pois $v \in H, K$ e v não tem vizinhos nestes grafos. Como $|\mathcal{V}_G(v)| = |\mathcal{E}_G(v)| = \delta(G)$ então $\kappa(G) \leq \delta(G)$ e $\lambda(G) \leq \delta(G)$ pois encontramos conjuntos de vértices e arestas com $\delta(G)$ elementos que ao serem removidos deixam G desconectado.

Resta-nos mostrar $\kappa(G) \leq \lambda(G)$. Seja $D \subset E$ um conjunto com $\lambda(G)$ arestas tal que $G - D$ é desconectado (a existência de D é garantida pela definição de aresta-conectividade). Primeiro, considere o caso onde existe um vértice em G que não é incidido por nenhuma aresta de D e seja $v \in V$ este vértice, ou seja, $\mathcal{E}_G(v) \cap D = \emptyset$. Seja $H = (U, F)$ o componente conectado de $G - D$ tal que $v \in U$. Seja $U_1 \subset U$ um subconjunto de vértices de H tal que os elementos de U são todos aqueles que são incididos por alguma aresta de D em G. Como não pode haver aresta de D com ambas as extremidades em U (pois se assim fosse existiria um conjunto com menos que $\lambda(G)$ arestas que desconecta G) então podemos garantir que $|U_1| \leq |D| = \lambda(G)$. Além disto, o grafo $G - U_1$ é desconectado, portanto $\kappa(G) \leq |U_1|$ garantindo que $\kappa(G) \leq \lambda(G)$.

O diagrama abaixo elucida as ideias do argumento, considerando D como as arestas pontilhadas e U_1 sendo os vértices em negrito.

Figura 6



Fonte: Autor

Suponha agora o caso que todo vértice de G seja incidido por alguma aresta de D e seja $v \in V$ um destes vértices. Seja $H = (U, F)$ o componente conectado de $G - D$ tal que $v \in U$. Pelo mesmo argumento do parágrafo anterior, uma aresta de D não incide ao mesmo tempo em dois vértices distintos na vizinhança de v em H, portanto para cada vértice em

$\mathcal{V}_H(v)$ temos pelo menos uma aresta em D. Estas arestas não podem incidir sobre v logo $[\mathcal{E}_G(v) \cap D] \cap [\mathcal{E}_G(\mathcal{V}_H(v)) \cap D]$ é vazio, sendo assim:

$$|[\mathcal{E}_G(v) \cap D] \cup [\mathcal{E}_G(\mathcal{V}_H(v)) \cap D]| = |\mathcal{E}_G(v) \cap D| + |\mathcal{E}_G(\mathcal{V}_H(v)) \cap D| \tag{7}$$

$$\implies |v|_G \leq |\mathcal{E}_G(v) \cap D| + |\mathcal{E}_G(\mathcal{V}_H(v)) \cap D| \leq |D| \tag{8}$$

Portanto $\delta(G) \leq |D| = \lambda(G)$. Neste caso os vizinhos de v em G separam v, portanto $\kappa(G) \leq |v|_G \leq |D| = \lambda(G)$.

■

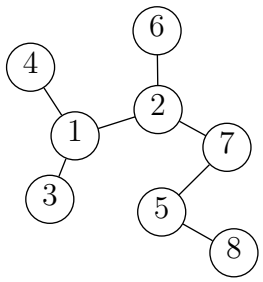
2.5 ÁRVORES E FLORESTAS

Árvores são um caso particular de grafos, pois aparecem em diversas aplicações e problemas, principalmente em estruturas de dados na Computação.

Definição 2.17. (Árvore e floresta)

Uma **árvore** é um grafo conectado que não contém ciclos (ou **acíclico**). Uma **floresta** é um grafo que acíclico onde os seus componentes conectados são árvores.

Figura 7 – Um exemplo de árvore



Fonte: Autor

Proposição 2.18. (Equivalências para árvores)

São equivalentes:

(1) G é uma árvore;

(2) Dois vértices de G são conectados por um único caminho;

(3) G é minimamente aresta-conectado, ou seja, remover qualquer aresta torna o grafo desconectado.

(4) G é maximalmente acíclico, ou seja, G não contém ciclos e adicionar qualquer aresta cria um ciclo em G.

Demonstração:

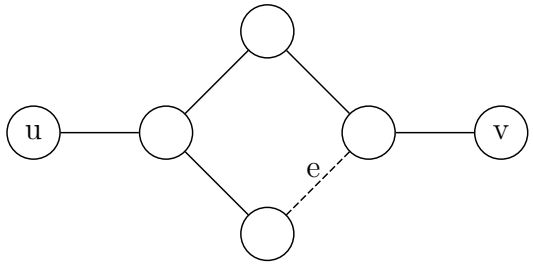
(1) $\implies$ (2): Por contrapositiva, basta mostrar que se existem nenhum ou mais de um caminho distintos (não necessariamente independentes) que conecta dois vértices então G não é uma árvore. Se não houver nenhum caminho então G não é conectado, e portanto não é uma árvore. Sejam u e v vértices de G e considere dois caminhos $u - v$ distintos denotados por $(v_n)_0^a$ e $(u_n)_0^b$. Sejam i o primeiro índice tal que $v_i \neq u_i$, $j > i$ o primeiro índice tal que $v_j = v_k$ para algum k. Note que o caminho abaixo é um ciclo em G pois $u_{i-1} = v_{i-1}$, portanto G não pode ser uma árvore.

$$(v_{i-1}, v_i, \ldots, v_j, u_{k-1}, u_{k-2}, \ldots, u_i, u_{i-1}) \tag{9}$$

(2) $\implies$ (3): Seja $e = \{u, v\} \in E$ e note que se removermos e do grafo o único caminho que conecta u e v deixará de existir, e portanto G se torna desconectado.

(3) $\implies$ (1): Se é minimamente aresta-conectado então é conectado, portanto precisamos provar que G não contém ciclos. Suponha, por absurdo, que G contém um ciclo. Escolha uma aresta neste ciclo e a remova do grafo. Se dois vértices eram conectados por um caminho que continha esta aresta note que ainda sim é possível construir outro caminho ‘‘dando a volta pelo outro lado’’ do ciclo, como mostrado no diagrama abaixo. Isto contradiz o fato de G ser minimamente conectado e portanto não pode conter ciclos.

Figura 8 – Mesmo removendo e ainda é possível conectar u e v



Fonte: Autor

(2) $\implies$ (4): Se G contiver um ciclo então contradiz o fato de que entre dois vértices existe um único caminho que os conecta. Portanto resta-nos provar que é maximalmente acíclico. Sejam u e v vértices de G que não são vizinhas em G. Note que só pode haver um único caminho A na forma $u - v$, e claramente este caminho não passa pela aresta $e = \{u, v\}$. Se adicionarmos e à G o caminho $A + v$ é um ciclo.

(4) $\implies$ (1): Se G é maximalmente acíclico então não contém ciclos. Portanto resta-nos provar que G é conectado. Suponha, por absurdo, que G seja desconectado e considere G_1 e G_2 dois componentes conectados distintos de G. Seja u um vértice de G_1 e v um vértice de G_2, temos que não existe aresta entre u e v em G. Se G é maximalmente acíclico então $G + e$ deve conter um ciclo, e é claro que este ciclo deve conter u e v. Pelo argumento utilizado na demonstração (3) $\implies$ (1), temos que deve existir dois caminhos em $G + e$ que conectam u e

v, onde um deles passa por e e outro não. Neste caso, a existência do caminho que não passa por e nos diz que G deve ter um caminho que conecta u com v, o que é um absurdo visto que ambos estão em componentes conectados distintos.

■

Proposição 2.19. (Enumeração de árvores)

Se G é uma árvore então existe uma enumeração $\{v_1, \ldots, v_n\}$ de V tal que para $i \geq 2$ v_i tem um *único* vizinho em $\{v_1, v_2, \ldots, v_{i-1}\}$.

Demonstração:

Pelas proposições 2.15 e 2.18, temos que existe enumeração $\{v_1, \ldots, v_n\}$ de uma árvore tal que $V_i = \{v_1, \ldots, v_i\}$ é conectado. Se v_i tivesse dois vizinhos em V_{i-1} então seria possível formar um ciclo com um caminho entre os vizinhos (pois V_{i-1} é conectado) e o próprio v_i, o que é um absurdo pois G é árvore. Portanto, v_i não pode ter dois ou mais vizinhos em V_{i-1}.

■

Proposição 2.20. (Existência de uma árvore geradora)

Se G é conectado então existe uma árvore que gera G.

Demonstração:

Seja $\{e_1, \ldots, e_n\}$ uma enumeração de E. Se $G - e_1$ é conectado defina $E_1 \coloneqq E - e_1$, caso o contrário $E_1 \coloneqq E$. Seja $V_i \coloneqq V$ para todo $i \leq n$ e considere a seguinte sequência para $i > 1$:

$$E_i \coloneqq \begin{cases} E_{i-1} - e_i, & G_{i-1} - e_i \text{ é conectado} \\ E_{i-1}, & \text{c.c} \end{cases} \tag{10}$$

Note que G_n será um subgrafo que gera G e é minimamente aresta-conectado, pois pela construção da sequência e pela proposição 2.13 não podemos remover nenhum vértice sem quebrar a conectividade. Como demonstrado em 2.18 temos que G_n é uma árvore que gera G.

■

Proposição 2.21. (Equivalência para árvores)

Seja G conectado com ordem n. G é uma árvore se, e somente se, $|E| = n - 1$.

Demonstração:

A ida decorre da proposição 2.19. Se v_i tem um único vizinho em V_{i-1} para todo $i \leq n$ então $E_i \setminus E_{i-1} = e$ onde e é a aresta que incide sobre v_i e seu vizinho em V_{i-1}. Conforme

aumentamos i, estamos adicionando apenas uma aresta a G_i, sendo assim um argumento indutivo em i garante que V_i possui $i-1$ arestas. Tomando $i = n$ provamos o que queríamos.

Agora a volta. Seja H uma árvore geradora de G. A primeira implicação demonstrada garante que H tem $n-1$ arestas e portanto devem ser as mesmas $n-1$ arestas de G, logo $G = H$.

■

Definição 2.22. (Raiz e ordem de uma árvore)

Seja G uma árvore e r um vértice de G. Chamaremos r de **raiz** de G e sabendo que só existe um único caminho que conecta dois vértices em uma árvore, definimos a relação $u \preceq_r v$ para os vértices u e v se u está contido no interior no caminho $r - v$. Por definição dizemos que $w \preceq_r w$ para todo $w \in V$.

Note que $\preceq_r$ é uma relação de ordem parcial em V. Por definição ela é reflexiva. Se $u \neq v$ e $u \preceq_r v$ então claramente $v \preceq_r u$ não pode ser verdade, tornando a relação antissimétrica. Se $u \preceq_r v$ e $v \preceq_r w$ então o caminho $r - w$ deve conter $r - v$, sendo assim o caminho $r - w$ contém u, portanto $u \preceq_r w$ garantindo a transitividade.

2.6 HOMOMORFISMOS E ISOMORFISMOS

Homomorfismos são funções entre grafos que preservam vizinhanças. Homomorfismos são importantes para podermos estudar um grafo a partir de outro visto que a relação de vizinhança edifica todas as definições e proposições.

Isomorfismo é o mesmo que dizer que dois grafos são ‘‘iguais’’ quando olhamos para as relações de vizinhança, ignorando a natureza dos vértices. O grafo com vértices que são frutas pode ter a mesma estrutura de um grafo onde os vértices são carros de luxo. Portanto o que nos importa no fundo é a estrutura do grafo e não o que são seus vértices.

Definição 2.23. (Homomorfismo e isomorfismo) Uma função $\phi : V \to V_1$ é um **homomorfismo** de G para G_1 se para todo $u, v \in V$ vizinhos em G tal que $\phi(u)$ e $\phi(v)$ também são vizinhos em G_1. Um **isomorfismo** é um homomorfismo bijetor que a sua inversa também é um homomorfismo. G é **homorfo** a G_1 se existe um homomorfismo de V para V_1 e, similarmente, G é **isomorfo** a G_1 se existe um isomorfismo de V para V_1. Note que que todo grafo é homorfo/isomorfo a si mesmo e que composição de homomorfismo/isomorfismo é homomorfismo/isomorfismo.

2.7 COLORAÇÕES

É possível colorir o mapa-múndi com apenas quatro cores sem que dois países vizinhos tenham a mesma cor?[3] Esta pergunta aparentemente inocente se demonstrou um desafio para diversos matemáticos e motiva a definição de coloração de grafos.

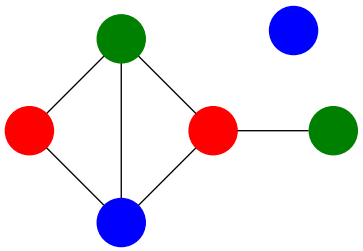
Colorir um grafo é associar para cada vértice uma cor de tal maneira que vértices vizinhos tenham cores diferentes. À primeira vista, isto pode parecer simples, pois podemos utilizar uma quantidade de cores igual a quantidade de vértices. O desafio reside em encontrar a menor quantidade de cores necessária para colorir um dado grafo. Tal qual o processo de encontrar um clique, encontrar a quantidade mínima de cores é bastante difícil e não existe método fácil senão testar todas as possíveis maneiras distintas de se colorir o grafo.

Neste capítulo, discutiremos as n-reduções de um grafo, tópico este que foi desenvolvido independentemente pelo autor e não consta na bibliografia. A n-redução se prova uma ferramenta importante para colorir grafos devido a sua propriedade de reduzir a complexidade do problema.

Definição 2.24. (Coloração)

Seja n um número natural não nulo. A função $\phi : V \to \{1, \ldots, n\}$ é uma **n-coloração** de G se para todo v e u vértices vizinhos em G tem-se que $\phi(u) \neq \phi(v)$. Se existir n-coloração em G então G é **n-colorável**. O **número cromático** de G, denotado por $\chi(G)$, é o menor número n tal que G é n-colorável. G é **perfeito** se $\chi(G) = \Gamma(G)$. É claro que todo subgrafo de um grafo n-colorável também é n-colorável e que um grafo é n-colorável se todos os seus componentes conectados são n-coloráveis.

Figura 9 – Exemplo de 3-coloração



Fonte: Autor

Podemos utilizar o diagrama de um grafo para representar uma possível coloração. No caso da Figura 9 estamos literalmente colorindo os vértices, mas podemos dizer que, por exemplo, colorir um vértice de vermelho é o mesmo que associar o valor 1, colorir o vértice de verde é associar o valor 2 e de azul o valor 3.

[3] Este é chamado "O Problema das quatro cores" e a sua resposta é sim. Sua demonstração foi feita por exaustão por um computador em 1976 devido a quantidade enorme de casos que precisavam ser testados.

Proposição 2.25. (Homomorfismo e coloração)

ϕ um homomorfismo entre G e G_1. Se G_1 é n-colorável então G é n-colorável.

Demonstração:

Seja e ψ a n-coloração de G_1. Temos que se u e v são vértices vizinhos de G então $\phi(u)$ e $\phi(v)$ são vizinhos em G_1. Se $\phi(u)$ e $\phi(v)$ são vizinhos em G_1 então $\psi(\phi(u))$ e $\psi(\phi(v))$ são diferentes. Logo $\psi \circ \phi$ é uma n-coloração de G.

■

Claramente podemos escolher uma quantidade de cores igual a ordem do grafo, associando a cada vértice uma cor diferente e única. Isto não é muito interessante, mas pelo menos garante que todo grafo G possui pelo menos uma $|G|$-coloração.

Proposição 2.26. (Extensão de colorações)

Seja $V_1 \subset V$ um subconjunto de vértices, $G_1 = \mathcal{I}_G(V_1)$ o grafo induzido, ϕ uma n-coloração em G_1 e $v \in V - V_1$ um vértice fora de G_1. Considere $V_2 = V_1 \cup \{v\}$ e $G_2 = \mathcal{I}_G(V_2)$. É verdade que:

(1) Se $|v|_{G_2} < n$ então G_2 é n-colorável;

(2) Se $|v|_{G_2} \geq n$ então G_2 é n-colorável ou no máximo $(n+1)$-colorável.

Demonstração:

Iremos mostrar (1). Note que $\mathcal{V}_{G_2}(v) \subseteq V_1$ e portanto $\phi(\mathcal{V}_{G_2}(v)) \subseteq \{1, \ldots, n\}$. Se $|v|_{G_2} < n$ então $M = \{1, \ldots, n\} \setminus \phi(\mathcal{V}_{G_2}(v)) \neq \emptyset$. Tome $m \in M$ e construa a função $\psi : V_2 \to \{1, \ldots n\}$ da seguinte maneira:

$$\psi(u) = \begin{cases} \phi(u), & u \in V_1 \\ m, & u = v \end{cases}$$

Por construção ψ é n-coloração em G_2, já que o valor escolhido para $\psi(v)$ está entre 1 e n e é diferente dos seus vizinhos.

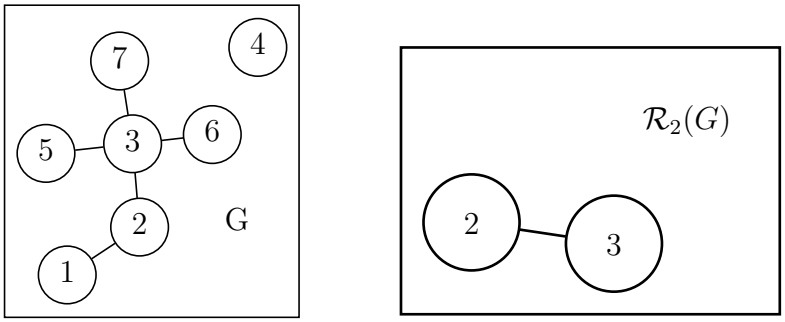
Iremos mostrar (2). Se $|v|_{G_2} \geq n$ então $M = \{1, \ldots, n\} \setminus \phi(\mathcal{V}_{G_2}(v))$ pode ou não ser vazio. Caso $M \neq \emptyset$ então segue de (1) a existência de uma n-coloração em G_2. Se $M = \emptyset$ tome $m = n + 1$ e construa a função $\psi : V_2 \to \{1, \ldots n + 1\}$ como a acima. Por construção ψ é $n + 1$-coloração em G_2, já que o valor escolhido para $\psi(v)$ está entre 1 e $n + 1$ e é diferente dos seus vizinhos.

■

Definição 2.27. (Redução)

Seja n um número natural. A n**-redução** de G, denotada por $\mathcal{R}_n(G)$, é o grafo induzido pelo subconjunto de vértices de G com ordem maior ou igual a n, ou seja, remove-se o os vértices de grau menor que n do grafo. O n**-reduzido** de G, denotado por $\mathcal{R}_n^*(G)$, é o grafo é obtido a partir da iteração da n-redução até que se remova todos os vértices ou o grau mínimo seja igual a n. É verdade que $G \geq \mathcal{R}_1(G) \geq \mathcal{R}_2(G) \geq \cdots$.

Figura 10



Fonte: Autor

A Figura 10 apresenta um exemplo de redução. Note que foram removidos os vértices $\{1, 4, 5, 6, 7\}$ pois os seus graus são menores que 2. Se aplicarmos a 2-redução novamente obteremos o grafo vazio.

Definição 2.28. (Núcleo)

O **núcleo** de G é o menor n-reduzido de G diferente do vazio, ou seja, queremos saber o maior n possível tal que $\mathcal{R}_n^*(G)$ seja não vazio. Na Figura 10 temos que o núcleo de G é $\mathcal{R}_1^*(G)$.

Proposição 2.29. (Redução e coloração)

G é n-colorável se, e somente se, $\mathcal{R}_n(G)$ é n-colorável. Como corolário desta proposição segue que G é n-colorável se, e somente se, $\mathcal{R}_n^*(G)$ é n-colorável.

Demonstração:

A ida é óbvia, pois a redução de G é subgrafo de G. A volta decorre da aplicação da proposição 2.26. Se a n-redução de um grafo está colorida com n cores então todo vértice que não está na redução tem menos de n vizinhos, e portanto é sempre possível escolher uma entre as n cores que seja diferente dos vizinhos.

■

Proposição 2.30. (Majorante para o número cromático)

É verdade que G é $(\Delta(G) + 1)$-colorável.

Demonstração:

Mesmo argumento que a demonstração 2.29. Se todo vértice tem menos que $\Delta(G) + 1$ vizinhos então sempre podemos escolher uma entre as $\Delta(G) + 1$ cores.

■

Mas como encontrar uma coloração de um grafo? Abaixo vamos definir o ‘‘algoritmo ganancioso para coloração’’, que colore um grafo vértice por vértice. Dizemos que o algoritmo é ‘‘ganancioso’’ pois ele tenta resolver o problema sem se preocupar com a solução ótima, apenas buscando minimizar localmente o uso das cores.

Definição 2.31. (Algoritmo ganancioso para coloração)

Seja $\{v_1, \ldots, v_n\}$ uma enumeração de V. Aplicando a proposição 2.26 nesta enumeração, podemos estender uma coloração inicial em G de tal forma que colorimos cada vértice com o menor valor entre os seus vizinhos, garantindo que seja utilizado no máximo $\Delta(G) + 1$ cores.

O resultado do algoritmo depende da ordenação escolhida. Existem diversas possíveis ordenações, desde aleatórias até aquelas que escolhem os vértices em ordem decrescente pelo valor do seu grau.

Proposição 2.32. (Existência de uma ordenação ótima)

Existe uma ordenação para o algoritmo 2.31 que usa $\chi(G)$ cores para colorir G.

Demonstração: Seja ϕ uma $\chi(G)$-coloração de G e ordene os vértices de G tal que os primeiros vértices estão coloridos com a cor 1, depois os vértices com a cor 2 e assim por diante. Seja $V_i \subset V$ os vértices que são coloridos com a cor i. Claramente o algoritmo irá colorir V_1 com a primeira cor, pois eles não podem ser vizinhos entre si pois ϕ é coloração. Os vértices de V_2 que fazem vizinhança com algum de V_1 serão coloridos com a cor 2 pelo algoritmo, caso o contrário serão coloridos com a cor 1. Repita indutivamente este argumento e eventualmente todos os vértices estarão coloridos com no máximo $\chi(G)$ cores.

■

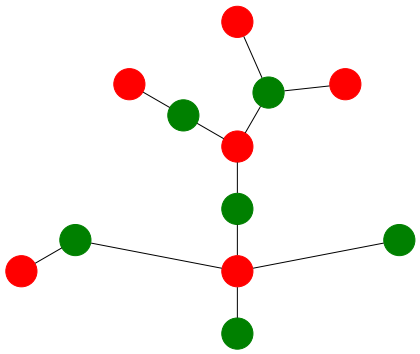
Proposição 2.33. (2-colorabilidade de árvores)

Toda árvore é 2-colorável.

Demonstração:

Seja G uma árvore e $r \in V$ uma raiz de G. Tome $v \in V$ e note que existe um único caminho que conecta r e v. Seja ϕ uma função de V para $\{1, 2\}$ tal que se o caminho que conecta r e v tem comprimento ímpar então $\phi(v) = 1$ e caso tenha comprimento par então $\phi(v) = 2$. Claramente vértices vizinhos não podem ter a mesma cor, fazendo com que ϕ seja uma 2-coloração para G.

Figura 11 – 2-coloração de árvores



Fonte: Autor

■

Proposição 2.34. (Condição para 2-colorabilidade)

G é 2-colorável se, e somente se, não existem ciclos de comprimento ímpar em G.

Demonstração:

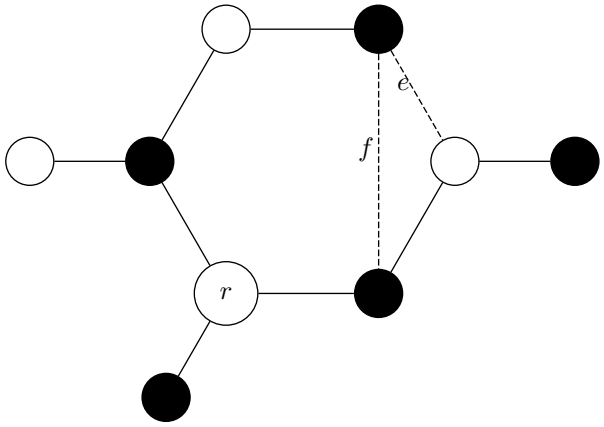
Podemos provar a ida por contrapositiva. Claramente, um grafo gerado por um ciclo de ordem ímpar em G não pode ser 2-colorável, pois as cores devem alternar dentro do ciclo, resultando em dois vértices com a mesma cor. Se G contém um subgrafo que não é 2-colorável então G não pode ser 2-colorável.

Vamos provar a volta. Suponha sem perda de generalidade que G é conectado, pois um grafo é 2-colorável se, e somente se, seus componentes conectados são 2-coloráveis. Portanto, pelas proposições 2.20 e 2.22 existe uma árvore T que gera G e $r \in V$ uma raiz de T. Pela proposição 2.33 sabemos que T deve ser 2-colorável e seja ϕ uma coloração. Tome $e = \{u, v\}$ aresta de G que não está em T (se não houver aresta desta forma então $T = G$ pois T gera G) e note que T é maximalmente acíclico pela proposição 2.18, portanto em $T + e$ existe um ciclo A em G que passa por e. Se os únicos caminhos que conectam r com u e r com v ambos em T têm paridades iguais, então A deve necessariamente ter ordem ímpar, o que é um absurdo. Portanto u e v tem cores diferentes e podemos adicionar as arestas de G uma a uma à T e garantir que a 2-coloração ϕ de T não é quebrada, tornando ϕ uma 2-coloração de G.

■

A Figura 12 abaixo ajuda a entender o argumento da proposição 2.34. Note que se adicionarmos a aresta e na árvore temos que a coloração não é quebrada, mas se adicionarmos f teremos dois vértices vizinhos com a mesma cor.

Figura 12 – 2-coloração de árvores



Fonte: Autor

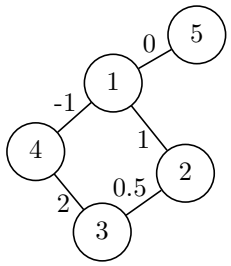
2.8 OUTROS TIPOS DE GRAFOS

Os grafos que acabamos de definir são chamados de **não dirigidos**, **não ponderados** e **sem *loops***[4]. Um grafo **ponderado** é aquele que cada aresta tem um valor associado, um grafo **dirigido** é aquele que a relação de vizinhança não é simétrica e um grafo **com *loops*** é aquele que um vértice pode ser vizinho de si mesmo.

Definição 2.35. (Grafo ponderado)

Seja $f : E \to \mathbb{R}$. O par (G, f) é um **grafo ponderado** com função peso f. No diagrama de um grafo ponderado colocamos o valor da aresta ao lado dela.

Figura 13 – Exemplo de grafo ponderado



Fonte: Autor

Um grafo ponderado pode ser utilizado quando a relação de vizinhança tem um peso associado. Por exemplo, ao fazermos a representação de uma cidade por um grafo, os pesos das arestas podem representar a distância entre as cidades vizinhas. O comprimento de um

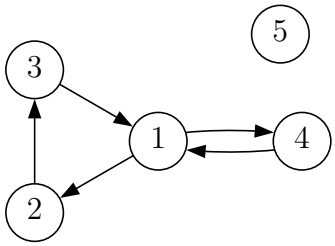
[4] Um *loop* pode ser obtido permitindo que subconjuntos unitários sejam arestas, tornando um vértice vizinho de si mesmo

caminho em um grafo ponderado é a soma dos valores associados para cada aresta que o caminho passa.

Definição 2.36. (Grafo dirigido)

Seja $D \subset V \times V$. O par (V, D) é um **grafo dirigido** com arestas D. Os vértices u e v são vizinhos se $(u, v) \in D$, note que u pode ser vizinho de v mas v pode não ser vizinho de u. No diagrama de um grafo direcionado utilizamos setas indicando a relação de vizinhança, onde se u é vizinho de v então desenhamos uma seta saindo de u indo para v.

Figura 14 – Exemplo de grafo direcionado



Fonte: Autor

No grafo dirigido não temos mais uma relação simétrica. No exemplo da cidade podemos dizer que é possível ir de uma cidade à outra, mas não fazer o caminho de volta. Grafos dirigidos podem ser ponderados.

3 IMPLEMENTAÇÃO COMPUTACIONAL

Em Matemática muitas vezes provamos que um exemplo existe, mas também importante sabermos encontrar a ‘‘cara’’ dele. Este capítulo portanto será dedicado a apresentar algoritmos em grafos que aplicam as proposições demonstradas anteriormente.

3.1 REPRESENTAÇÃO DE UM GRAFO

A definição inicial de grafo funciona muito bem para desenvolver a Teoria dos Grafos com a linguagem de conjuntos, mas para executar algoritmos computacionalmente devemos definir o que é um grafo para um computador. Existem diversas maneiras, mas a que melhor se aplica para o nosso caso é a partir de uma **matriz de incidência**: para um grafo G enumere os vértices $\{v_1, \ldots, v_{|G|}\}$ e monte uma matriz $A = (a_{ij})$ de 0's e 1's tal que se v_i é vizinho de v_j então a $a_{ij} = a_{ji} = 1$, caso contrário $a_{ij} = a_{ji} = 0$.

Utilizando a matriz de incidência, vamos aplicar diversos algoritmos citados nas definições e proposições anteriores. Abaixo uma lista das funções implementadas:

- Encontrar a vizinhança e o grau de um vértice;
- Calcular o maior e menor grau em um grafo;
- Calcular a densidade de um grafo;
- Encontrar o subgrafo descrito na proposição 2.6;
- Calcular a distância entre dois vértices e checar se um grafo é conexo;
- Encontrar a árvore geradora de um grafo conexo;
- Fazer a coloração gananciosa em um grafo e retornar a coloração encontrada;
- Fazer a redução de um grafo e encontrar o seu reduzido.

O código está escrito na linguagem Julia, uma linguagem de alto nível e performance feita para computação científica.

3.2 CÓDIGO

```
struct Grafo
    # Struct para as variáveis de tipo Grafo
    A::Matrix{Int64}
    # Matriz de adjacencia do grafo
    Id::Vector{String}
    # Cada entrada eh o nome de cada vertice
end

function QuantidadeVertices(G::Grafo)::Int64
    return length(G.Id)
```

```
end

function VizinhancaGrauVertice(G::Grafo, v::Int64)::Tuple{Int64,Vector{Int64}}
    # Encontra os vertices vizinhos de v em G e calcula o seu grau
    N::Int64 = QuantidadeVertices(G)
    grau::Int64 = sum(G.A[v,:])
    vizinhanca::Vector{Int64} = zeros(Int64, grau)
    i::Int64 = 1
    for C::Int64 in 1:N
        # Itera sobre a coluna e verifica se eh vizinho
        if G.A[v, C] == 1
            vizinhanca[i] = C
            i += 1
        end
    end
    return (grau, vizinhanca)
end

function GrausMenorMaiorGrafo(G::Grafo)::Tuple{Vector{Int64}, Int64, Int64}
    # Retorna um vetor com o grau de cada vertice e o valor do menor
    # e maior grau
    N::Int64 = QuantidadeVertices(G)
    graus::Vector{Int64} = zeros(Int64, N)
    for L::Int64 in 1:N
        graus[L] = VizinhancaGrauVertice(G, L)[1]
    end
    if N == 0
        return (graus, 0, 0)
    else
        return (graus, minimum(graus), maximum(graus))
    end
end

function QuantidadeArestas(G::Grafo)::Int64
    return div(sum(GrausMenorMaiorGrafo(G)[1]), 2)
end

function DensidadeGrafo(G::Grafo)::Rational
    nArestas::Int64 = QuantidadeArestas(G)
    nVertices::Int64 = QuantidadeVertices(G)
    return nArestas//nVertices
end

function PassoSubgrafoDenso(G::Grafo)::Tuple{Grafo, Bool}
    # Faz o passo de escolher um vértice grau
    # menor que a densidade
    N::Int64 = QuantidadeVertices(G)
    densidade::Rational = DensidadeGrafo(G)
    graus::Vector{Int64} = GrausMenorMaiorGrafo(G)[1]
    reduzido::Bool = false
    for V::Int64 in 1:N
        if graus[V]*denominator(densidade) <= numerator(densidade)
            reduzido = true
            vertices::Vector{Int64} = 1:N
            filter!(v -> v != V, vertices)
            return (Grafo(G.A[vertices, vertices], G.Id[vertices]), reduzido)
        end
    end
    return (G, reduzido)
```

```
end

function SubgrafoDenso(G::Grafo)::Tuple{Grafo, Bool}
    # Encontra um subgrafo mais denso
    nG::Tuple{Grafo, Bool} = PassoSubgrafoDenso(G)
    while nG[2]
        nG = PassoSubgrafoDenso(nG[1])
    end
    return nG
end

function ColoracaoGananciosa(G::Grafo, O::Vector{Int64})::Vector{Int64}
    N::Int64 = QuantidadeVertices(G)
    corMax::Int64 = maximum(GrausMenorMaiorGrafo(G)[1]) + 1
    # Quantidade maxima possivel de cores
    # Segue do segundo majorante para o numero cromatico
    coloracao::Vector{Int64} = zeros(Int64, N)
    for L::Int64 in O
        coresPossiveis::Vector{Int64} = 1:corMax
        for C::Int64 in 1:N
            if G.A[L,C] == 1
                filter!(c -> c != coloracao[C], coresPossiveis)
                # Remove da lista de cores possiveis aquelas que
                # os vertices vizinhos ja estao coloridos
            end
        end
        coloracao[L] = minimum(coresPossiveis)
    end
    return coloracao
end

function NReducao(G::Grafo, N::Int64)::Tuple{Grafo,Bool}
    N1::Int64 = QuantidadeVertices(G)
    graus::Vector{Int64} = GrausMenorMaiorGrafo(G)[1]
    nVertices::Vector{Int64} = zeros(Int64, 0)
    # Vertices que permanecem
    for V::Int64 in 1:N1
        if graus[V] >= N
            push!(nVertices, V)
        end
    end
    reduzido::Bool = false
    # Checa se o grafo foi reduzido
    if length(nVertices) < length(G.Id)
        reduzido = true
    end
    return (Grafo(G.A[nVertices, nVertices], G.Id[nVertices]), reduzido)
end

function NReduzido(G::Grafo, N::Int64)::Tuple{Grafo,Bool}
    nG::Tuple{Grafo,Bool} = NReducao(G, N)
    while nG[2]
        # Checa se o grafo foi reduzido
        # Caso positivo tenta reduzir mais
        nG = NReducao(nG[1], N)
    end
    return nG
end
```

```
function Nucleo(G::Grafo)::Tuple{Grafo, Int64}
    maiorGrau::Int64 = GrausMenorMaiorGrafo(G)[3]
    for I::Int64 in 0:maiorGrau
        nG::Tuple{Grafo,Bool} = NReduzido(G,I)
        if length(nG[1].Id) == 0
            nG = NReduzido(G,I - 1)
            return (nG[1], I - 1)
        end
    end
end

function DistanciaVertices(G::Grafo, u::Int64, v::Int64)::Int64
    N::Int64 = QuantidadeVertices(G)
    distancias::Vector{Int64} = zeros(Int64, N)
    visitado::Vector{Int64} = zeros(Int64, N)
    nVisitado::Vector{Int64} = zeros(Int64, N)
    visitado[u] = 1
    alteracao::Bool = true
    while alteracao
        alteracao = false
        for I::Int64 in 1:N
            if visitado[I] == 1
                for J::Int64 in 1:N
                    if G.A[I,J] == 1 && visitado[J] == 0
                        nVisitado[J] = 1
                        distancias[J] = distancias[I] + 1
                    end
                end
                nVisitado[I] = -1
                alteracao = true
            elseif visitado[I] == -1
                nVisitado[I] = -1
            end
        end
        visitado = copy(nVisitado)
    end
    return distancias[v]
end

function ChecarConexidade(G::Grafo)::Bool
    N::Int64 = QuantidadeVertices(G)
    conexo::Bool = true
    for I::Int64 in 1:N
        for J::Int64 in 1:N
            if I != J
                if DistanciaVertices(G, I, J) == 0
                    conexo = false
                    break
                end
            end
        end
        if !conexo
            break
        end
    end
    return conexo
end

function ArvoreGeradora(G::Grafo)::Grafo
```

```
    N::Int64 = QuantidadeVertices(G)
    nA::Matrix{Int64} = copy(G.A)
    nId::Vector{String} = copy(G.Id)
    for L::Int64 in 1:N
        for C::Int64 in (L + 1):N
            if nA[L,C] == 1
                nA[L,C] = 0
                nA[C,L] = 0
                if ChecarConexidade(Grafo(nA, nId))
                    continue
                else
                    nA[L,C] = 1
                    nA[C,L] = 1
                end
            end
        end
    end
    return Grafo(nA, nId)
end
```

4 CONSIDERAÇÕES FINAIS

Este trabalho trouxe alguns dos principais resultados para um primeiro estudo em Teoria dos Grafos. Outros tópicos, como fluxos e planaridade, também são interessantes para um primeiro contato, mas não foram abordados aqui.

Apesar de simples, as aplicações desta teoria são vastas. Conceitos de Computação como autômatos[3], organização de arquivos, controle de versões e outras coisas podem ser representadas e resolvidas com ferramentas da Teoria dos Grafos.

A motivação para fazer esta pesquisa foi movida pelo meu interesse em estudar colorações em grafos, pois acho um tema fascinante. Ainda pretendo responder algumas perguntas e descobrir novas proposições, principalmente aquelas que envolvem a redução.

Algumas dúvidas ainda permanecem comigo. Será que existem boas ordenações para o algoritmo da ordenação, melhores que escolher os vértices em ordem crescente pelo valor do seu grau? Será que o núcleo é o menor subgrafo que ‘‘fixa’’ uma coloração para o grafo inteiro, ou seja, o menor subgrafo que garante que se é colorável então todo grafo também é? É possível saber quando a coloração gananciosa é ótima e melhorar colorações a partir de alguma já conhecida? Quais as propriedades únicas de colorações obtidas a partir da coloração gananciosa?

Esta dúvidas por pouco não entraram no trabalho, visto que nem todas fui capaz de responder com certeza e demonstrar formalmente. Ainda assim, são com dúvidas em aberto que continuamos pesquisar as áreas que nos interessam.

REFERÊNCIAS

[1] Thomas Cormen. *Algoritmos: Teoria e Prática*. 3ª ed. GEN LTC, 2012.

[2] Reinhard Diestel. *Graph Theory*. 2ª ed. Springer, 2010.

[3] Michael Sipser. *Introduction to the Theory of Computation*. 3ª ed. Cengage Learning, 2012.

[4] William Thomas Tutte. *Graph Theory*. 1ª ed. Cambridge University Press, 2001.