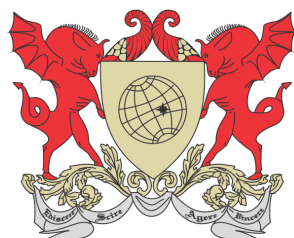


UNIVERSIDADE FEDERAL DE VIÇOSA  
DISSERTAÇÃO DE MESTRADO



DÁRCIO COSTA NOGUEIRA JÚNIOR

# GRAFOS E PROBLEMAS DE CAMINHOS

FLORESTAL  
MINAS GERAIS – BRASIL  
2017

DÁRCIO COSTA NOGUEIRA JÚNIOR

## GRAFOS E PROBLEMAS DE CAMINHOS

Dissertação apresentada à Universidade Federal de Viçosa,  
como parte das exigências do Programa de Pós-Graduação  
Mestrado Profissional em Matemática em Rede Nacional,  
para obter o título *Magister Scientiae*.

FLORESTAL  
MINAS GERAIS – BRASIL  
2017

**Ficha catalográfica preparada pela Biblioteca da Universidade Federal  
de Viçosa - Câmpus Florestal**

T

N778g  
2017 Nogueira Júnior, Dárcio Costa, 1978-  
Grafos e Problemas de Caminhos. / Dárcio Costa Nogueira  
Júnior. – Florestal, MG, 2017.  
x, 89f : il. ; 29 cm.

Inclui apêndice.

Orientador: Luís Felipe Gonçalves Fonseca.

Dissertação (mestrado) - Universidade Federal de Viçosa.

Referências bibliográficas: f.87-89.

1. Teoria dos grafos. 2. Matemática - Ensino médio.  
3. Algoritmos - Processamento de dados. I. Universidade Federal  
de Viçosa. Departamento de Matemática. Programa de  
Pós-graduação Mestrado Profissional em Matemática em Rede  
Nacional. II. Título.

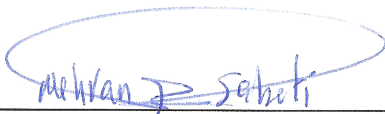
CDD 22 ed. 511.54

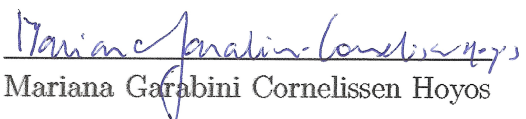
DÁRCIO COSTA NOGUEIRA JÚNIOR

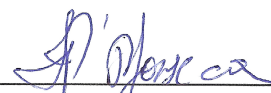
## GRAFOS E PROBLEMAS DE CAMINHOS

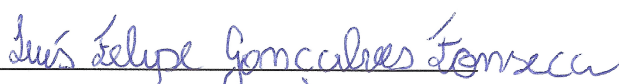
Dissertação apresentada à Universidade Federal de Viçosa,  
como parte das exigências do Programa de Pós-Graduação  
Mestrado Profissional em Matemática em Rede Nacional,  
para obter o título *Magister Scientiae*.

APROVADA: 22 de fevereiro de 2017.

  
\_\_\_\_\_  
Mehran Sabeti

  
\_\_\_\_\_  
Mariana Garabini Cornelissen Hoyos

  
\_\_\_\_\_  
Luis Alberto D'Afonseca  
(Coorientador)

  
\_\_\_\_\_  
Luís Felipe Gonçalves Fonseca  
(Orientador)

# Dedicatória

---

Em cada instante está lá. Desde a primeira busca até a constante procura pelo conhecer a si mesmo. No respirar, em cada sentimento, no pulsar do coração está lá. A Deus. Somente a Deus.

# Agradecimentos

---

Se há memória do coração, certamente ela é a gratidão.

Agradeço ao professor, orientador e amigo, Luís Felipe, pelo excepcional planejamento e condução das orientações. A sua dedicação e contribuições foram fundamentais para essa pesquisa. Cada encontro foi um aprendizado com marcas perenes em minha trajetória como educador.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) por todo apoio financeiro e suporte.

Aos meus pais, Dárcio e Vanilda, pelas orações, presença e momentos inesquecíveis. Vocês são parte dessa conquista.

Aos meus pais, Jane e Oswaldo, pelas intercessões, presença e momentos memoráveis. Cada café e almoço, conversas e partilhas, são como refrigerio, presente de Deus.

Aos meus irmãos Tatiana e Douglas, com respectivas famílias: somos um. Isso é o mais importante.

Andréia: nos momentos mais importantes e felizes, mais difíceis e tristes, você é a prova de que há amigos mais chegados que um irmão. Você é minha irmã de caminhada. Sempre.

À primeira turma do mestrado da UFV - Florestal: apesar dos momentos quentes do verão, vocês são incríveis! Momentos marcantes! Muito obrigado!

Aos professores, em especial Luís Felipe (orientador), Luis D'Afonseca (coorientador) e Mehran (coordenador), obrigado pelas lições e ensinamentos.

Aos professores da banca da defesa: obrigado pelas contribuições, correções e sugestões.

Aos amigos e colegas, que apoiaram esse trabalho, obrigado pela presença e suporte.

Aos alunos do segundo ano do CMBH (2016): obrigado pelo apoio na caminhada e pelas inúmeras contribuições. Sou muito agradecido a cada um de vocês.

# Lista de Símbolos

---

Símbolos e notações utilizadas neste trabalho:

|             |                                 |             |                                |
|-------------|---------------------------------|-------------|--------------------------------|
| $NULL$      | Algo sem valor definido         | $\epsilon$  | Letra grega Épsilon            |
| $O$         | Complexidade                    | min         | Mínimo                         |
| $\emptyset$ | Conjunto vazio                  | $\chi(G)$   | Número cromático de $G$        |
| $V_{IND}$   | Conj. de vértices independentes | $\alpha(G)$ | Número de independência de $G$ |
| $:=$        | Definido por                    | $\in$       | Pertence                       |
| $d(v)$      | Grau do vértice $v$             | $\prec$     | Precede                        |
| $\infty$    | Infinito                        | $\cup$      | União                          |
| $\cong$     | Isomorfismo                     |             |                                |

# Lista de Figuras

---

|      |   |    |
|------|---|----|
| 1.1  | As sete pontes de Königsberg e o grafo associado. [38]  | 1  |
| 1.2  | As oito pontes no centro de Recife e o grafo associado. [14]  | 2  |
| 2.1  | O problema de Euler: as pontes de Königsberg. [44]  | 4  |
| 2.2  | Desenho no artigo de Euler, de 1736, mas publicado apenas em 1741. [45]   | 5  |
| 2.3  | O problema de Kirchhoff. [27]   | 5  |
| 2.4  | O problema de Cayley. [27]  | 5  |
| 2.5  | O problema de Guthrie. [11]   | 6  |
| 2.6  | Definição de grafo. [44]  | 7  |
| 2.7  | Exemplo de grafo com arestas múltiplas e laço   | 7  |
| 2.8  | Grafos simples, valorado e direcionado. No grafo direcionado temos as arestas $(a,b); (b,c); (c,d); (d,e); (a,e)$ . | 9  |
| 2.9  | Grafos planares.  | 9  |
| 2.10 | Grafos regular, completo, bipartido e rotulado.   | 10 |
| 2.11 | Multigrafo e subgrafo.  | 10 |
| 2.12 | Exemplo de isomorfismo em grafos [30].  | 11 |
| 2.13 | Exemplo de isomorfismo em grafos [5].   | 11 |
| 2.14 | Caminhos e ciclos em grafos   | 12 |
| 2.15 | Grafo euleriano e semieuleriano. [21]   | 13 |
| 2.16 | Grafos euleriano e hamiltoniano. [44]   | 14 |
| 2.17 | Grafo: árvore   | 15 |
| 2.18 | Aplicação de coloração de grafos: Sudoku.   | 16 |
| 2.19 | O desafio das três casas. [26]  | 19 |
| 2.20 | Aplicação de coloração de grafos: programação de horários. [15]   | 20 |
| 2.21 | Conjunto independente. [35]   | 23 |
| 2.22 | Um dos ciclos hamiltonianos na solução do problema do cavalo. [4]   | 24 |
| 2.23 | Exemplo de aplicação do algoritmo de Dijkstra. [16]   | 25 |
| 3.1  | Exemplo de algoritmo: recepcionista de cinema. [19]   | 28 |
| 3.2  | Exemplo de fluxograma: recepcionista de cinema.   | 28 |
| 3.3  | Exemplo de algoritmo na computação: multiplicação de dois números inteiros positivos. [19]                          | 29 |
| 3.4  | Hierarquia de funções. [43]   | 30 |



---

|      |   |    |
|------|---|----|
| 3.5  | Exemplo de matriz de adjacência. [33]   | 31 |
| 3.6  | Matriz de adjacência para grafo e para digrafo. [28]  | 32 |
| 3.7  | Lista de adjacências para grafo e para digrafo. [28]  | 32 |
| 3.8  | Implementação de matriz de adjacências. [29]  | 33 |
| 3.9  | Implementação de matriz de adjacências. [29]  | 34 |
| 3.10 | Caminho mínimo do vértice $a$ até $g$ - exemplo do resultado obtido pelo algoritmo de Dijkstra. [8]                     | 34 |
| 3.11 | Caminho mínimo do vértice $a$ até $g$ - tabela com passos e distâncias do algoritmo de Dijkstra. [8]                    | 35 |
| 3.12 | Implementação do algoritmo de Dijkstra. [11]  | 35 |
| 3.13 | Exemplo da execução do algoritmo de Dijkstra. [20]  | 36 |
| 3.14 | Passos obtidos com a modificação do algoritmo de Dijkstra. [20]   | 36 |
| 3.15 | Grafo $G$ . [42]  | 37 |
| 3.16 | Matriz de adjacência $M$ e matriz de roteamento $R$ . [42]  | 38 |
| 3.17 | Matrizes de roteamento com vértices intermediários. [42]  | 38 |
| 3.18 | O algoritmo de Floyd. [11]  | 39 |
| 3.19 | Exemplo de aplicação algoritmo de Kruskal. [13]   | 40 |
| 3.20 | O algoritmo de Kruskal. [11]  | 41 |
| 3.21 | Exemplo de aplicação do algoritmo de Prim. [36]   | 42 |
| 3.22 | O algoritmo de Prim. [11]   | 43 |
|      |   |    |
| 4.1  | Interface do Code::Blocks [2]   | 48 |
| 4.2  | Esquema do laboratório de informática e a disposição dos alunos   | 50 |
| 4.3  | Atividade 1.1.1 apresentada pelo aluno 6  | 52 |
| 4.4  | Atividade 1.2.1 apresentada pelo aluno 1  | 53 |
| 4.5  | Atividade 1.4.1 apresentada pelo aluno 3  | 54 |
| 4.6  | Atividade 2.1.1 apresentada pelo aluno 4  | 55 |
| 4.7  | Atividade 2.1.2 apresentada pelo aluno 15   | 55 |
| 4.8  | Atividade 2.2.1 apresentada pelo aluno 13   | 56 |
| 4.9  | Atividade 2.2.2 apresentada pelo aluno 9  | 56 |
| 4.10 | Atividade 2.2.4 apresentada pelo aluno 13   | 57 |
| 4.11 | Atividade 2.3.1 apresentada pelo aluno 13   | 58 |
| 4.12 | Atividade 2.3.3 apresentada pelo aluno 13   | 58 |
| 4.13 | Atividade 3.1.1: conjunto de arestas dirigidas de um digrafo. Padrão de resposta dos alunos abaixo da figura dada. [18] | 59 |
| 4.14 | Atividade 3.2.1: Padrão de resposta dos alunos na execução do programa. [18]  | 59 |
| 4.15 | Atividade 3.2.1: árvore de caminho mínimo obtida pelo algoritmo de Dijkstra. [18]                                       | 60 |

# Resumo

---

NOGUEIRA JÚNIOR, Dárcio Costa, M.Sc., Universidade Federal de Viçosa, fevereiro de 2017. **Grafos e problemas de caminhos**. Orientador: Luís Felipe Gonçalves Fonseca. Coorientador: Luis Alberto D’Afonseca.

A Teoria dos Grafos está associada a situações que podem ser descritas por meio de diagramas representados por um conjunto de pontos (vértices) e linhas que ligam alguns pares destes pontos (arestas). Seu início remonta a visita de Leonhard Euler à cidade de Königsberg, em 1736, quando foi apresentado a ele um desafio que intrigava os moradores da cidade. Eles se perguntavam se era possível sair de casa, passar em cada ponte, apenas uma vez, e retornar ao ponto inicial. O diagrama montado por Euler para representar o mapa das sete pontes da cidade é um esquema de grafo. O desenvolvimento e a consolidação da Teoria dos Grafos proporcionou significativas contribuições para a Física, Química, Biologia e Ciência da Computação. Os algoritmos associados a problemas de caminho mínimo, coloração e busca de árvore geradora mínima são amplamente utilizados na prática de linguagem de programação. Nessa pesquisa, o problema de caminho mínimo e a busca da árvore geradora mínima são usados para o trabalho de algoritmos envolvendo grafos com alunos do Ensino Médio em uma escola de Belo Horizonte. Uma sequência didática com três aulas foi aplicada, sendo a primeira aula sobre a introdução à teoria dos grafos, a segunda aula sobre algoritmos e grafos e a terceira aula com a implementação desses algoritmos usando a linguagem de programação C. Os algoritmos utilizados foram Dijkstra, Prim, Kruskal e Floyd. Resultados apontam para a possibilidade de inclusão da Teoria dos Grafos no Ensino Médio tendo em vista as interações com Análise Combinatória, Probabilidade e Poliedros. O estudo de grafos por meio de algoritmos e sua aplicação em linguagem de programação é uma nova abordagem a ser considerada para o Ensino Médio.

# Abstract

---

NOGUEIRA JÚNIOR, Dárcio Costa, M.Sc., Universidade Federal de Viçosa, February, 2017. **Graphs and path problems**. Adviser: Luís Felipe Gonçalves Fonseca. Co-adviser: Luis Alberto D'Afonseca.

The Theory of Graphs is associated with situations that can be described by means of diagrams represented by a set of points (vertices) and lines that connect some pairs of these points (edges). Its beginning dates back to Leonhard Euler's visit to the town of Königsberg in 1736, when he was presented with a challenge that intrigued the city's residents. They wondered if it was possible to get out of the house, pass on each bridge only once, and return to the starting point. The diagram assembled by Euler represent the map of the city's seven bridges is a graph diagram. The development and consolidation of Graph Theory provided significant contributions to Physics, Chemistry, Biology and Computer Science. The algorithms associated with minimum path, coloring, and minimum generation tree search algorithms are widely used in programming language practice. In this research, the minimum path problem and the minimum generation tree search are used to work with algorithms involving graphs with high school students in a school in Belo Horizonte. A didactic sequence with three sections was applied, being the first class on the introduction to the graph theory, the second class on algorithms and graphs and the third class with the implementation of these algorithms using the C programming language. The algorithms used were Dijkstra, Prim, Kruskal and Floyd. Results point to the possibility of including the Theory of Graphs in High School in view of the interactions with Combinatorial Analysis, Probability and Polyhedra. The study of graphs through algorithms and their application in programming language is a new approach to be considered for High School.

# Sumário

---

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introdução</b>  | <b>1</b>  |
| <b>2</b> | <b>Noções de Grafos</b>  | <b>4</b>  |
| 2.1      | Primeiras ideias de Grafos . . . . .   | 6         |
| 2.2      | Alguns tipos de grafos . . . . .   | 8         |
| 2.3      | Isomorfismo . . . . .  | 11        |
| 2.4      | Definições relacionadas a caminhos e ciclos . . . . .                        | 12        |
| 2.5      | Árvores . . . . .  | 13        |
| 2.6      | Coloração . . . . .  | 16        |
| 2.7      | Teoremas, problemas e outros resultados . . . . .                            | 17        |
| <b>3</b> | <b>Grafos, problemas e algoritmos</b>  | <b>26</b> |
| 3.1      | Algoritmos . . . . .   | 26        |
| 3.2      | Matriz de Adjacência. . . . .  | 31        |
| 3.3      | Algoritmo de Dijkstra e a busca de caminho mais curto . . . . .              | 34        |
| 3.4      | O algoritmo de Floyd e o problema do menor caminho . . . . .                 | 37        |
| 3.5      | O algoritmo de Kruskal e a árvore geradora mínima . . . . .                  | 39        |
| 3.6      | O algoritmo de Prim e a árvore geradora de custo mínimo . . . . .            | 42        |
| <b>4</b> | <b>A sequência didática</b>  | <b>44</b> |
| 4.1      | Introdução ao estudo de grafos com auxílio de algoritmos de programação      | 45        |
| 4.1.1    | Aula 1: Grafos . . . . .   | 46        |
| 4.1.2    | Aula 2: Grafos - problemas e algoritmos . . . . .                            | 46        |
| 4.1.3    | Aula 3: Grafos - algoritmos de programação . . . . .                         | 48        |
| 4.2      | A aplicação da sequência didática . . . . .                                  | 49        |
| 4.3      | Análise dos dados coletados . . . . .  | 51        |
| 4.3.1    | Os grafos . . . . .  | 52        |
| 4.3.2    | Grafos: problemas e algoritmos . . . . .                                     | 54        |
| 4.3.3    | Grafos: algoritmos de programação . . . . .                                  | 58        |
| 4.3.4    | Aula extra: socialização das impressões sobre a sequência didática . . . . . | 60        |
| <b>5</b> | <b>Considerações finais</b>  | <b>62</b> |
| <b>A</b> | <b>Apêndice - Sequência didática</b>   | <b>64</b> |

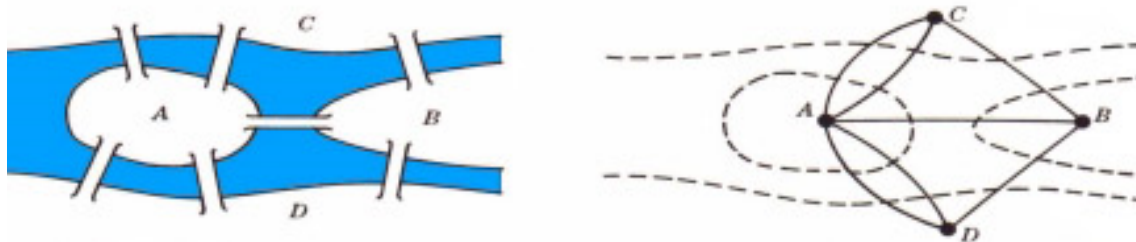


# Introdução

---

Os grafos, devido a sua natureza combinatória, servem de modelos para vários problemas na Matemática. A teoria dos grafos se mostrou um eficiente assunto para abordar aplicação de algoritmos para resolução de problemas práticos.

A história da Matemática teve um episódio específico quando Euler foi visitar a cidade de Königsberg em meados da primeira metade do século XVIII. A teoria dos grafos surgiu de um problema, proposto como um desafio pelos moradores daquela cidade. O desafio, conhecido como o “Problema das Sete Pontes de Königsberg”, consistia em saber se era possível sair de casa, passar uma vez em cada ponte e voltar para casa. Euler resolveu o desafio e publicou a resolução em um artigo no ano de 1741. Após um século sem registros relevantes, além dos publicados por Euler, o século XIX foi um momento de retomada e desenvolvimento dos grafos. Os problemas foram se tornando notórios devido às suas implicações práticas.



**Figura 1.1:** As sete pontes de Königsberg e o grafo associado. [38]

Ao longo da história, muitos dos problemas da teoria dos grafos foram associados a algoritmos, o que resultou em questões de complexidade computacional. O teorema das quatro cores mostra que é possível colorir um mapa com apenas quatro cores, de modo que países vizinhos não sejam coloridos com a mesma cor. Esse teorema exemplifica tal complexidade, já que um período de várias décadas separa a primeira tentativa de demonstração da prova em que se utilizou computadores.

Nesta dissertação, a teoria dos grafos é introduzida por meio de um breve contexto histórico, ressaltando sua origem em problema de aplicação prática. Após a definição, alguns elementos e as suas representações são descritos. Os conceitos fundamentais antecedem os tipos de grafos, que são abordados de acordo com suas utilizações nos

algoritmos propostos. Há uma preocupação em apresentar teoremas e lemas, com suas respectivas demonstrações matemáticas.

Os algoritmos são apresentados de maneira sucinta, seguido da definição e aplicação da matriz de adjacência. Dois algoritmos (Dijkstra e Floyd) são propostos para problemas que buscam o caminho mínimo e dois algoritmos (Kruskal e Prim) para a árvore geradora mínima. Os algoritmos apresentados podem ser adaptados para qualquer linguagem de programação.

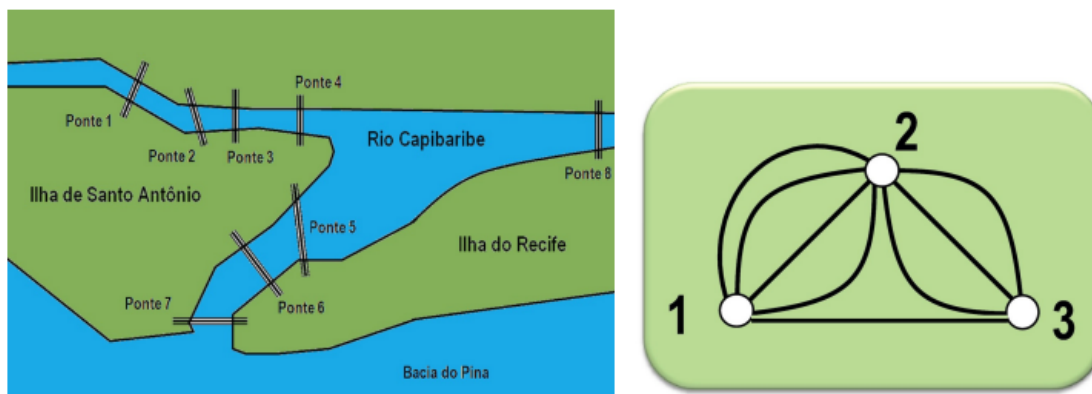
A teoria dos grafos e as possibilidades de interações com outras áreas da Matemática foram balizadores para uma sequência didática. Dividida em três aulas, cada uma de 100 minutos, as atividades permearam cada passo da sequência. A primeira aula trouxe uma introdução à teoria dos grafos, enquanto a segunda aula iniciou a aplicação de algoritmos como Dijkstra e Kruskal, por exemplo. A terceira aula aborda a linguagem de programação C, escolhida para essa pesquisa devido ao seu uso na Olimpíada Brasileira de Informática (OBI).

As aulas ocorreram numa escola federal de Belo Horizonte ao longo de duas semanas. Dezesseis alunos do segundo ano do Ensino Médio foram voluntários para participar das aulas. Os dados coletados por meio de observação das aulas, execução e recolhimento das atividades, assim como as anotações da socialização realizada duas semanas após a última das três aulas, foram analisados com viés de pesquisa qualitativa.

Esses dados foram analisados e os resultados mostraram a possibilidade de inclusão de grafos no Ensino Médio, tendo em vista a série em que se ensina Análise Combinatória, Probabilidades e Poliedros.

O uso da linguagem C para a programação dos algoritmos se mostrou eficiente apenas para alunos com conhecimento prévio. Nesse contexto, destacam-se aqueles que já participaram da OBI ou que estão matriculados em algum curso técnico.

Algumas questões levantadas na observação e análise de dados são apresentadas nas considerações finais. A oportunidade de inserir grafos nos currículos pode trazer benefícios para o estudo de algoritmos, a demonstração de teoremas e a resolução de problemas práticos.



**Figura 1.2:** As oito pontes no centro de Recife e o grafo associado. [14]

O caminho para as aplicações é repleto de possibilidades de contextualização de

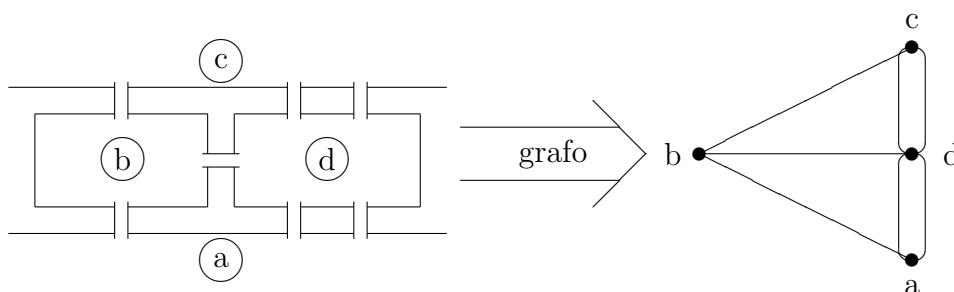
grafos tendo em vista às regionalidades de nossa nação. Uma delas está representada na figura 1.2, que pode ser considerada a Königsberg brasileira do século XXI. A mesma impossibilidade encontrada por Euler pode ser percebida em Recife, uma vez que é impossível obter um ciclo euleriano, isto é, um caminho que passasse por todas as pontes apenas uma vez e retornasse ao ponto inicial. Esse exemplo de contextualização atesta a relevância e as possibilidades de trabalho quando se introduz grafos na Educação Básica, em especial, no Ensino Médio, conforme abordado na pesquisa.



## Noções de Grafos

---

A Teoria dos Grafos é relativamente contemporânea na Matemática. Seus registros iniciais remontam a um problema que foi resolvido por Leonhard Euler ao visitar a cidade de Königsberg em 1736. Localizada numa pequena região da Rússia, a cidade possuía uma intensa atividade intelectual e um relevante comércio marítimo. É nesse contexto que Euler, ao visitar a cidade, se depara com um problema de aparência simples, mas que estava sendo amplamente discutido pela elite intelectual da cidade. O desafio consistia em determinar um trajeto iniciado em uma das margens do Pregel, rio que corta a cidade, e passasse por todas as pontes apenas uma única vez, retornando ao ponto de partida.



**Figura 2.1:** O problema de Euler: as pontes de Königsberg. [44]

Ele mostrou que era impossível realizar tal trajeto, a não ser que cada margem fosse ligada a uma ilha por um número par de pontes. O desafio proposto a Euler resultou num artigo escrito em 1736 [17]. Mais tarde, o esquema apresentado por Euler recebeu o nome de grafo.

Apenas 111 anos mais tarde, o físico alemão Gustav Kirchhoff pensou em algo parecido quando publicou alguns resultados em que se utilizava modelos de grafos. Na figura 2.3, pode ver-se o uso de “árvores matemáticas” para a investigação de circuitos elétricos. A lei das correntes de Kirchhoff pode ser enunciada como “para qualquer rede de parâmetros concentrados, para qualquer de seus cortes, e a qualquer instante, a soma algébrica de todas as correntes através dos braços do corte é zero”. [27]

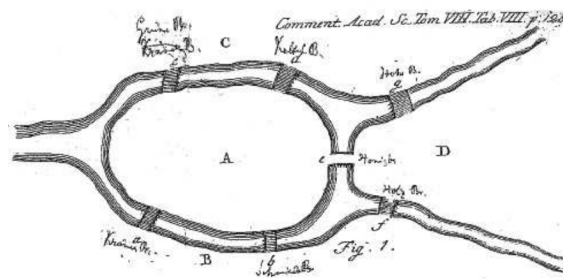


Figura 2.2: Desenho no artigo de Euler, de 1736, mas publicado apenas em 1741. [45]

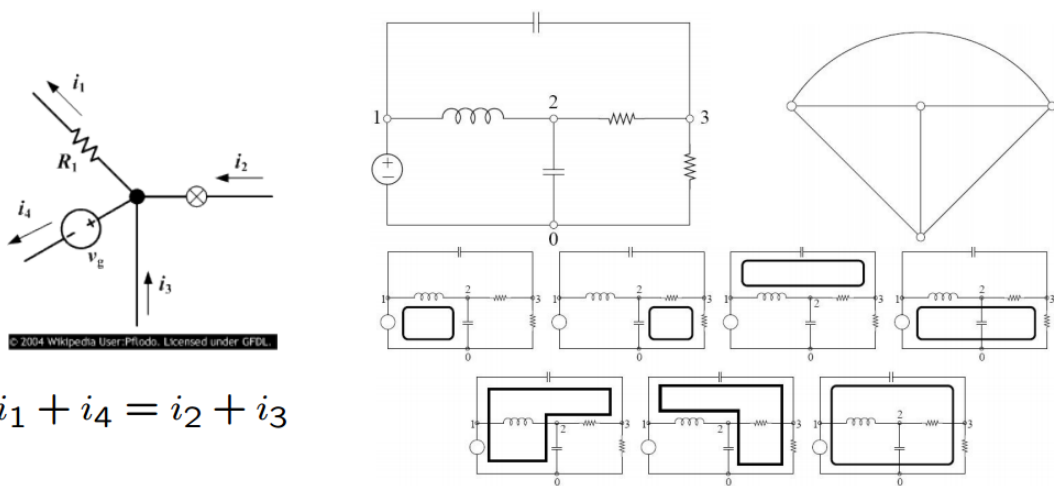


Figura 2.3: O problema de Kirchhoff. [27]

Alguns anos depois, mas precisamente em 1857, o matemático britânico Arthur Cayley usou a ideia de grafos para enumerar todos os isômeros dos hidrocarbonetos alifáticos, sendo estes compostos de carbono e hidrogênio com cadeias abertas, conforme indicado na figura 2.4. Neles, a cadeia carbônica é acíclica e por serem isômeros, possuem a mesma fórmula molecular mesmo com fórmulas estruturais distintas.

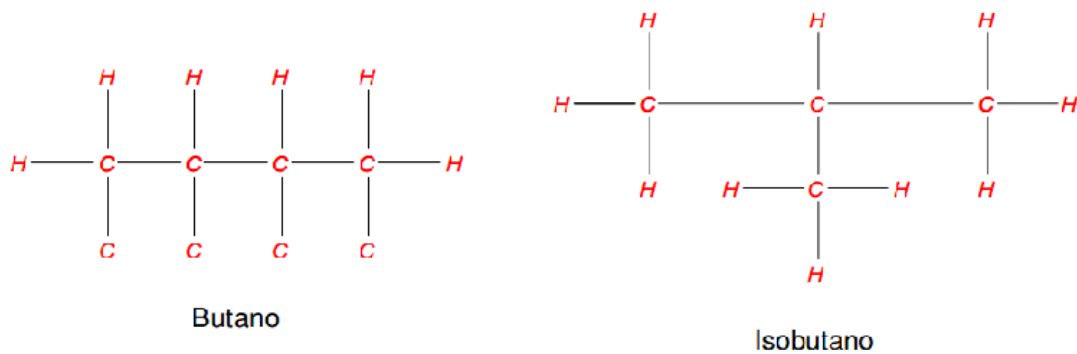


Figura 2.4: O problema de Cayley. [27]

Nessa mesma época, Francis Guthrie apresentou a conjectura das quatro cores enquanto tentava colorir um mapa dos condados da Inglaterra. Ele notou que apenas quatro cores eram suficientes para colorir o mapa, sendo que cada região não apresentava a mesma cor das demais regiões com as quais se fazia fronteira. Associando pontos a cada condado, Guthrie uniu dois pontos por uma linha se as duas regiões faziam fronteira, conforme a figura 2.5. Cerca de 124 anos se passaram e muitos métodos foram desenvolvidos para resolver o problema das quatro cores. Apenas em 1976, com a ajuda de um IBM 360, Kenneth Appel e Wolfgang Haken apresentaram uma demonstração para o teorema das quatro cores [10].



Figura 2.5: O problema de Guthrie. [11]

No século XX, muitos matemáticos contribuíram com Teoria dos Grafos. Diversas áreas do conhecimento como a Ciência da Computação, Biologia e a estrutura do DNA, Química e o projeto de novos compostos químicos foram beneficiadas.

## 2.1 Primeiras ideias de Grafos

A seguir, uma breve apresentação da Teoria Matemática dos Grafos.

**Definição 2.1.1.** Um grafo não direcionado  $G = (V, E)$  é uma estrutura matemática formada por dois conjuntos, o conjunto  $V$  dos vértices e o conjunto  $E$  das arestas. Uma aresta existe se dois vértices estão associados a ela.

A compreensão de um grafo depende de saber quais são os vértices e como eles se interligam dois a dois. Isso proporciona uma rápida visualização gráfica devido à facilidade na sua representação por meio de esquemas. A seguir, alguns conceitos sobre grafos.

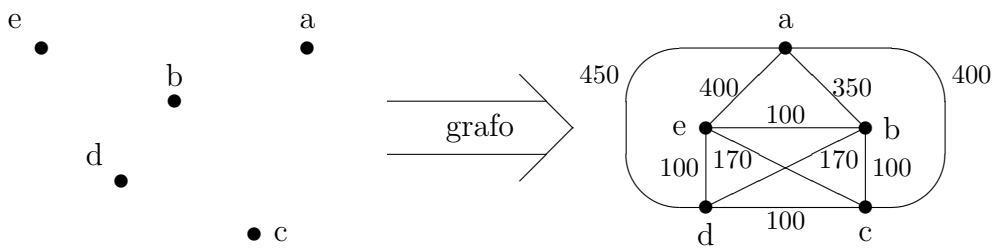


Figura 2.6: Definição de grafo. [44]

- Laço: aresta da forma  $\{v,v\}$ ;
- Vértices adjacentes: vértices de uma mesma aresta;
- Aresta incidente ao vértice  $v$ : aresta que contém  $v$ ;
- Extremidade de uma aresta: vértice da aresta;
- Grafo nulo: aquele para o qual  $\mathbf{V} = \emptyset$ .
- Ordem de um grafo  $G = (V, E)$ : é a cardinalidade do seu conjunto de vértices, ou seja, o número de vértices de  $G$ .
- Tamanho de um grafo  $G = (V, E)$ : é o número de arestas de  $G$ .

O vértice de um grafo é representado por um círculo ou ponto; a aresta é representada por uma linha.

**Exemplo 2.1.2.** No grafo  $G = (V, E)$  temos três vértices e cinco arestas. A ordem de  $G$  é 3.

- $G = (\mathbf{V}, \mathbf{E})$ , em que:
  - $\mathbf{V} = \{a,b,c\}$ ;
  - $\mathbf{E} = \{\{a,b\}, \{a,c\}, \{b,c\}, \{b,c\}, \{c,c\}\}$ .
- Uma representação geométrica:

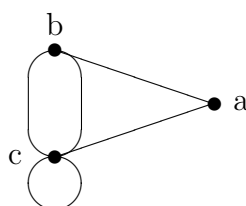


Figura 2.7: Exemplo de grafo com arestas múltiplas e laço

Em um grafo, o grau do vértice  $v$  é o número de vezes que as arestas incidem sobre  $v$ , sendo representado por  $d(v)$ . No exemplo da figura 2.7,  $d(a) = 2$ ,  $d(b) = 3$  e  $d(c) = 5$ . Observe que cada laço é contado duas vezes e que se a figura apresentasse algum vértice isolado, esse vértice apresentaria grau zero. Um resultado, conhecido como o **Lema do Aperto de Mãos** foi provado por Leonhard Euler em 1736. [17]

**Lema 2.1.3. do Aperto de Mãos:** A soma dos graus dos vértices de um grafo é sempre o dobro do número de arestas.

**Prova.** Por definição, basta notar que cada aresta, incluindo os laços, contribui com duas unidades para a soma. ■

**Corolário 2.1.4.** Todo grafo  $G$  possui um número par de vértices de grau ímpar.

**Prova.** Uma soma de números naturais é par, se, e somente se, a quantidade de números ímpares é par. Assim, pelo lema do aperto de mãos, a soma dos graus é par e, portanto, o número de vértices de grau ímpar é par. ■

## 2.2 Alguns tipos de grafos

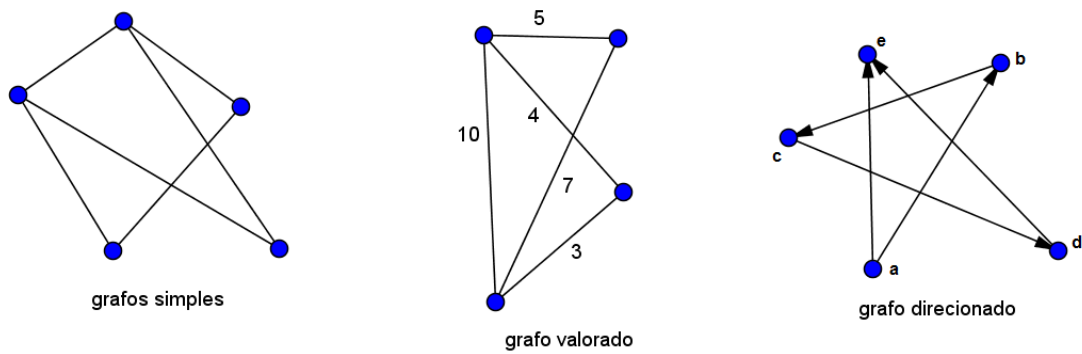
Os conceitos de grafos são relativamente simples e podem ser utilizados na resolução de problemas de Análise Combinatória, Probabilidades, Geometria e Programação de Computadores.

- Um grafo simples é aquele que não apresenta laços e nem arestas múltiplas.
- Um grafo valorado é aquele em que cada aresta possui um valor (peso) associado.
- Um grafo direcionado ou orientado apresenta arestas determinadas por pares ordenados. Nesse caso, a aresta  $(a,b)$  deve ser desenhada por meio de uma seta que vai de  $a$  até  $b$ .

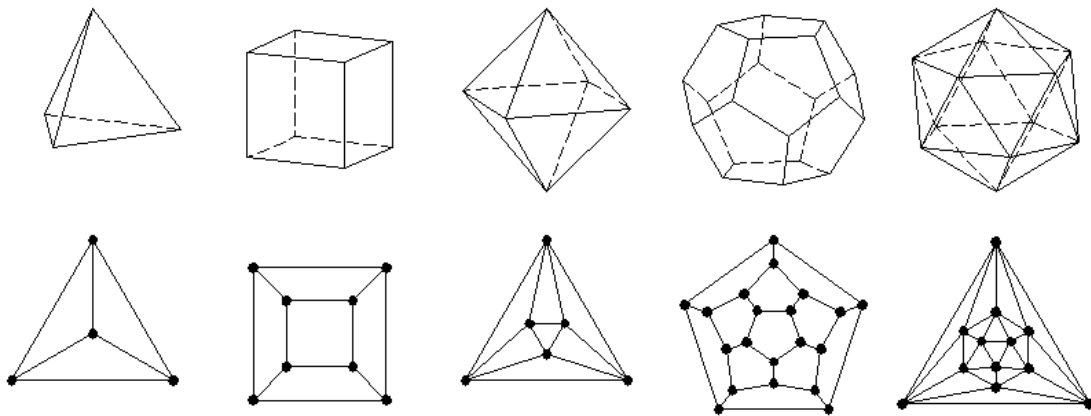
Um dos tipos de grafos que pode ser aplicado à representação de mapas de bairros de uma cidade é o grafo planar. Se o bairro não apresentar nenhum viaduto, as ruas se cruzarão apenas nos cruzamentos. É um caso de grafo em que as arestas se interceptam apenas nas extremidades, quando representado em um plano.

- Quando um grafo  $G$  admite uma representação numa superfície  $S$  sem que existam arestas que se interceptam, diz que ele é realizável em  $S$ . Um grafo diz-se planar se é realizável no plano. [32]

Os poliedros platônicos são exemplos de grafos planares. Da associação com cada planificação dessas figuras espaciais é possível identificar a quantidade de vértices, arestas e faces de um poliedro.



**Figura 2.8:** Grafos simples, valorado e direcionado. No grafo direcionado temos as arestas  $(a,b)$ ;  $(b,c)$ ;  $(c,d)$ ;  $(d,e)$ ;  $(a,e)$ .



**Figura 2.9:** Grafos planares.

- Um grafo é dito regular se todos os seus vértices apresentam o mesmo grau.
- Um grafo é completo se qualquer par de vértices é conectado por uma aresta. São denotados por  $K_n$ , sendo  $n$  a ordem do grafo. Todo grafo completo  $K_n$  é também regular de ordem  $n - 1$ , visto que todos os seus vértices tem grau  $n - 1$ .
- Um grafo é conexo (ou conectado) quando existe um caminho entre qualquer par de vértices. Caso contrário, demonina-se desconexo. Se todos os vértices apresentam grau zero, temos um grafo totalmente desconexo.
- Um grafo é bipartido quando seu conjunto de vértices  $V$  pode ser particionado em dois subconjuntos  $V_1$  e  $V_2$  tais que toda aresta do grafo associa um vértice de  $V_1$  a outro vértice de  $V_2$ . Um grafo é bipartido completo quando todos os vértices de um subconjunto estão ligados a todos os vértices do outro subconjunto.
- Um grafo é rotulado em vértices ou arestas quando a cada vértice ou aresta, respectivamente, um rótulo estiver associado.

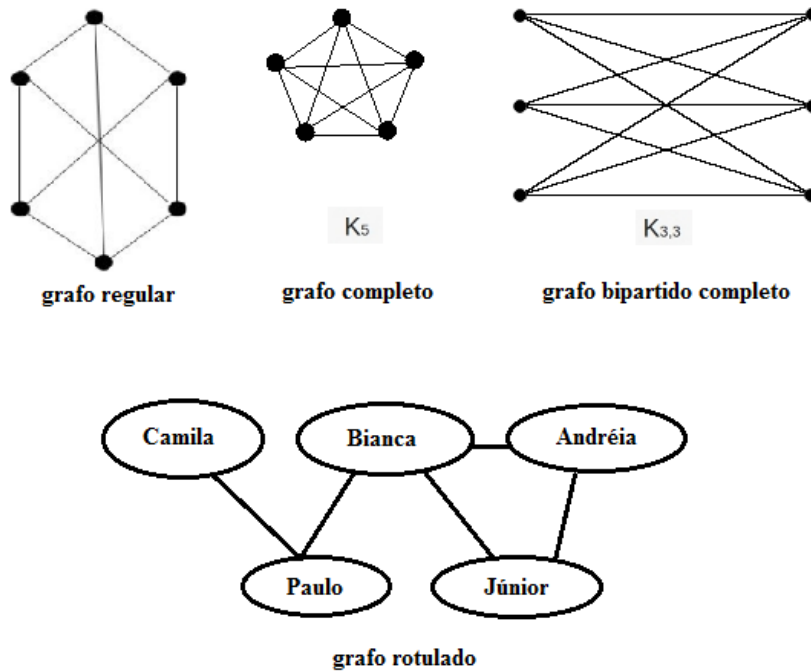


Figura 2.10: Grafos regular, completo, bipartido e rotulado.

- Um multigrafo é um grafo que apresenta múltiplas arestas entre pares de seus vértices.
- Subgrafo de um grafo é aquele em que o conjunto de seus vértices e o conjunto de suas arestas são, respectivamente, subconjuntos do conjunto de vértices e do conjunto de arestas do grafo dado.

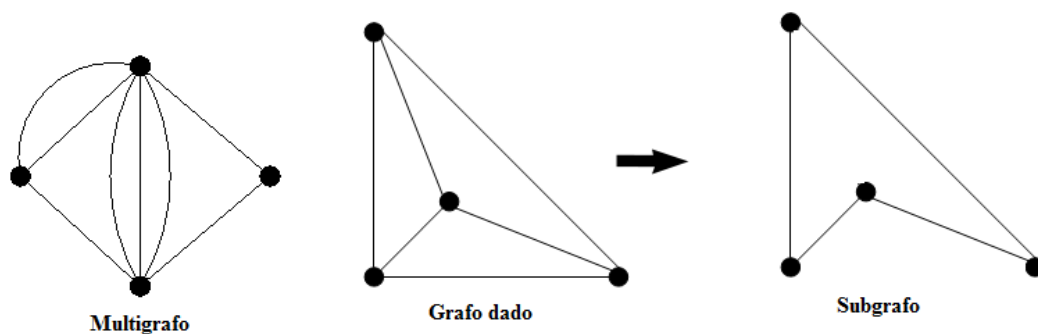


Figura 2.11: Multigrafo e subgrafo.

### 2.3 Isomorfismo

Dois grafos  $G$  e  $G'$  são isomorfos ( $G \cong G'$ ) quando existe uma função bijetora  $f : V(G) \rightarrow V(G')$  tal que se  $(u,v) \in E(G)$ , então  $(f(u), f(v)) \in E(G')$ .

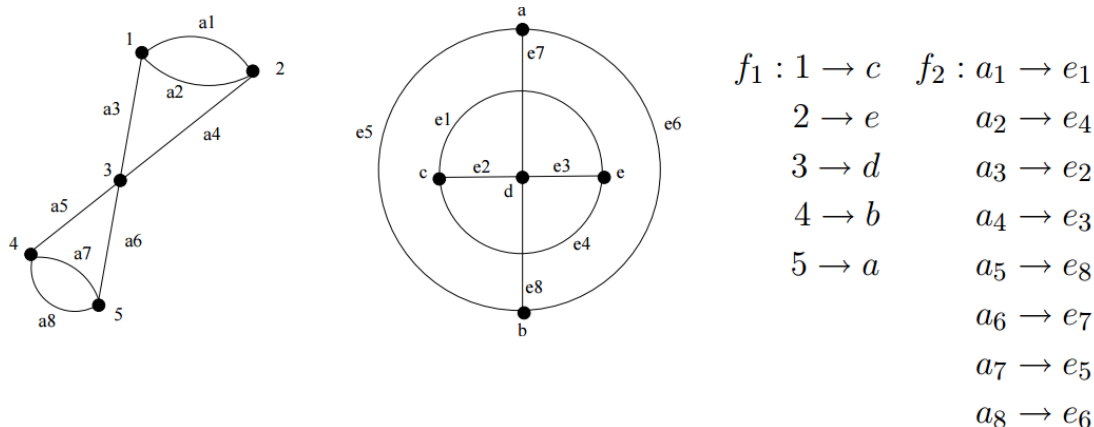


Figura 2.12: Exemplo de isomorfismo em grafos [30].

Ou seja, dois grafos são isomorfos quando existe uma correspondência vértice a vértice, de modo a preservar as adjacências, como visto na figura 2.12

Determinar se dois grafos são isomorfos nem sempre é uma tarefa simples. Apesar da possibilidade de se fazer correspondência vértice a vértice, preservando as adjacências, a quantidade de permutações possível pode revelar uma alta complexidade em problemas desse tipo. No entanto, há uma técnica [5] (não simples de implementar) que mantém os rótulos dos vértices, mas no entanto modifica o seu desenho, obtendo um novo grafo. Na figura 2.13, os vértices  $f, g, h, i, j$  foram colocados para fora e os vértices  $a, b, c, d, e$  foram colocados no meio, obtendo um grafo isomorfo.

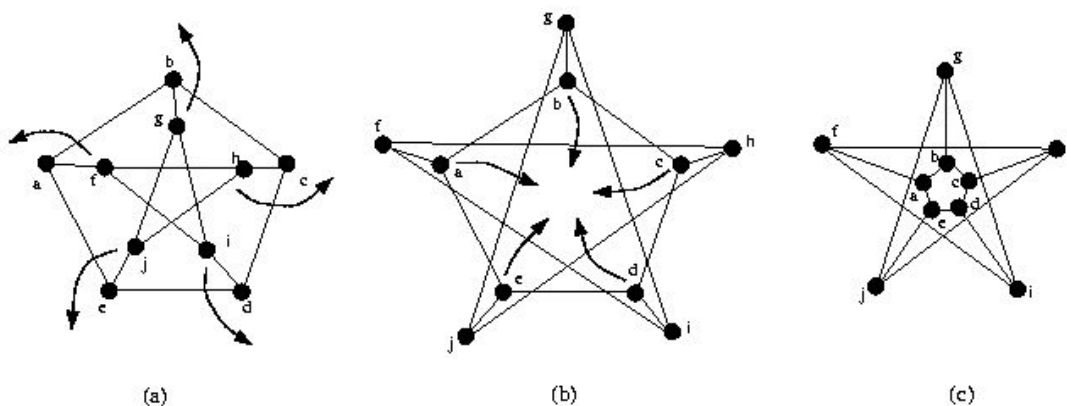


Figura 2.13: Exemplo de isomorfismo em grafos [5].

O isomorfismo, denotado por  $\cong$ , é uma relação de equivalência no conjunto dos grafos, pois satisfaz as propriedades:



- 1) a simetria ( $G_1 \cong G_1$  para todo o grafo  $G_1$ );
- 2) reflexividade (Se  $G_1 \cong G_2$ , então  $G_2 \cong G_1$ , para quaisquer grafos  $G_1$  e  $G_2$ );
- 3) transitividade ( $G_1 \cong G_2$  e  $G_2 \cong G_3 \rightarrow G_1 \cong G_3$ ).

## 2.4 Definições relacionadas a caminhos e ciclos

Um passeio em um grafo  $G = (V, E)$  é uma sequência de vértices consecutivos ligados por meio de arestas ou arcos. Desse modo, uma sequência  $v_1, \dots, v_k \in V$ , é um passeio de  $v_1$  a  $v_k$ , se  $(v_j, v_{j+1}) \in E$  para todo  $j = 1, \dots, m - 1$ . Logo, um passeio com  $j$  vértices apresenta  $j - 1$  arestas. Essa quantidade de arestas é o comprimento do passeio. Se as arestas são todas distintas, esse passeio denomina-se trilha.

Um caminho é um passeio que não contém vértices repetidos. Seu comprimento é dado pelo número de arestas desse passeio.

Um circuito é um passeio em que o vértice de partida coincide com o vértice de chegada. O ciclo, por sua vez, é um circuito com todos os vértices distintos entre o primeiro e o último.

Na figura 2.14, por exemplo, temos o passeio  $a, b, c, d, e, b, c$ , a trilha  $a, b, e, d, a$ , o caminho  $a, d, c, b, e$ , o circuito  $a, c, e, b, c, d, a$ , o ciclo  $a, b, e, c, d, a$  e o triângulo  $c, d, e, c$ .

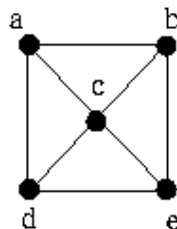


Figura 2.14: Caminhos e ciclos em grafos

Um grafo conexo é euleriano quando existe uma trilha fechada contendo cada uma das suas arestas.

**Teorema 2.4.1. Teorema de Euler [17]:** Um grafo conexo é euleriano se, e somente se o grau de cada vértice é par.

**Prova [21].** ( $\Rightarrow$ ) Seja  $G$  um grafo euleriano. Logo, ele contém um circuito euleriano. Por cada vértice desse caminho, existe uma aresta que chega nesse vértice e outra que sai desse vértice. Uma vez que toda aresta faz parte do caminho, necessariamente o número de arestas por cada vértice é par.

( $\Leftarrow$ ) Suponha que todos os vértices possuem grau par. Seja  $v_i$  um vértice qualquer do grafo. A partir de  $v_i$ , tenta-se construir um caminho que não passa duas vezes pela mesma aresta, até que não seja mais possível continuar. Se os vértices possuem grau par, é possível entrar e sair de um vértice, com exceção de  $v_i$ , onde o caminho vai terminar. Se esse caminho  $C_1$  contém todas as arestas de  $G$ , temos um circuito

euleriano. Caso contrário, retira-se todas as arestas de  $G$  que fazem parte de  $C_1$ . No grafo resultante  $G'$ , todos os vértices tem grau par e um deles necessariamente faz parte de  $C_1$  (a fim de garantir que o grafo seja conexo). Recomeçando o mesmo processo em  $G'$  a partir do vértice comum com  $C_1$ , obtendo  $C_2$ . Portanto, temos um circuito único que contém todas as arestas de  $G'$ . ■

Um circuito ou caminho é euleriano quando todas as suas arestas são contempladas no circuito ou caminho uma única vez, respectivamente. Ora, se um grafo não apresenta circuito euleriano, mas apresenta um caminho euleriano, esse grafo é denominado semieuleriano.

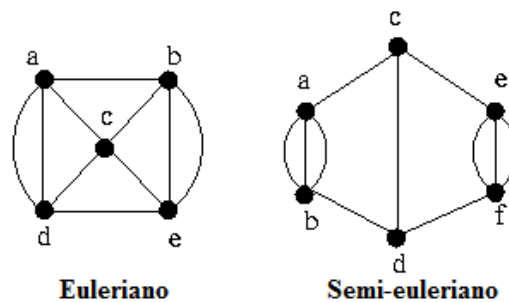


Figura 2.15: Grafo euleriano e semieuleriano. [21]

Com a definição de grafos eulerianos e o teorema 2.4.1, o problema das sete pontes de Königsberg se torna trivial. Como o grafo não é euleriano, fica impossível partir de um lugar, atravessar todas as pontes uma única vez e voltar ao ponto de partida, conforme indicado na figura 2.1.

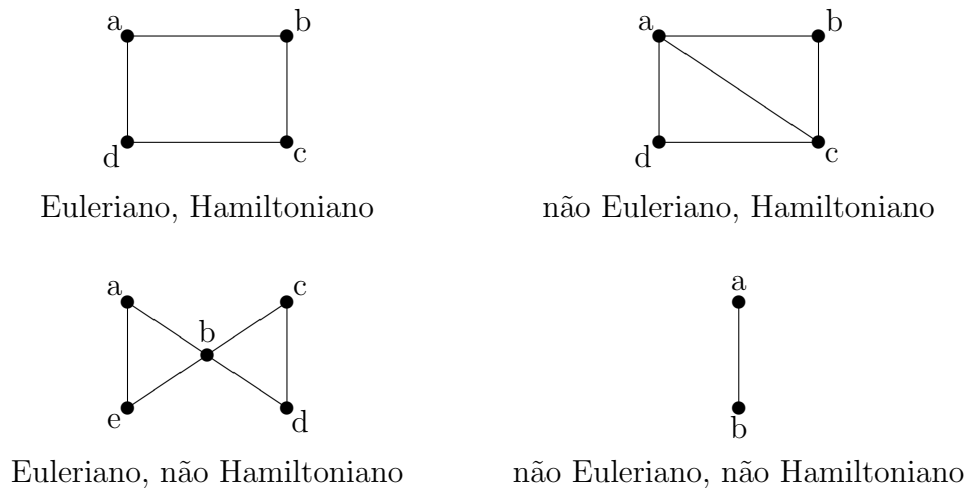
Um caminho é hamiltoniano quando é possível passar por todos os vértices sem repetição. No caso em que começa e termina no mesmo ponto, temos um ciclo hamiltoniano. Se um grafo possui um ciclo desse tipo, ele é denominado grafo hamiltoniano.

## 2.5 Árvores

Uma árvore é um grafo conexo que não apresenta ciclos. Uma floresta é um grafo não necessariamente conexo acíclico, isto é, sem ciclos. Segundo Jurkiewicz (2009) [24], “para um dado número de vértices  $n$ , uma árvore é o grafo conexo com menor número de arestas.”O teorema 2.5.1 reúne as várias caracterizações das árvores.

**Teorema 2.5.1.** Seja  $T$  um grafo com  $n$  vértices. As seguintes afirmações são equivalentes:

- I.  $T$  é uma árvore.
- II.  $T$  não contém ciclos e tem  $n - 1$  arestas.
- III.  $T$  é conexo e tem  $n - 1$  arestas.



**Figura 2.16:** Grafos euleriano e hamiltoniano. [44]

- IV.  $T$  é conexo e toda aresta é uma ponte <sup>1</sup>.
- V. Todo par de vértices de  $T$  é ligado por um único caminho.
- VI.  $T$  não contém ciclos, mas a adição de uma aresta produz um único ciclo.

**Prova** [24].

(I)  $\Rightarrow$  (II) Se  $T$  é uma árvore com  $n$  vértices, por definição,  $T$  não contém ciclos. Ao retirar uma aresta  $uv$ , separando os vértices  $u$  e  $v$ , tem-se um par de árvores  $T'$  e  $T''$  com  $n'$  e  $n''$  vértices respectivamente. Logo,  $n = n' + n''$ . Por indução, os números de arestas de  $T'$  e  $T''$  são  $n' - 1$  e  $n'' - 1$  arestas, respectivamente. Acrescentando a aresta  $uv$ , conclui-se que o número de arestas de  $T$  é  $(n' - 1) + (n'' - 1) + 1 = (n' + n'') - 1 = n - 1$ .

(II)  $\Rightarrow$  (III) Se  $T$  fosse desconexo, cada componente seria uma árvore. Por indução, o número de arestas em cada componente é inferior em uma unidade ao número  $n$  de vértices e o número total de arestas seria inferior a  $n - 1$ . Portanto,  $T$  é conexo e tem  $n - 1$  arestas.

(III)  $\Rightarrow$  (IV) A retirada de qualquer aresta separa o grafo, uma vez que  $n - 2$  arestas não são suficientes para conectar o grafo. Portanto,  $T$  é conexo e toda aresta é uma ponte.

(IV)  $\Rightarrow$  (V) Supondo que exista mais de um caminho entre dois vértices, o grafo apresentaria um ciclo e uma aresta que não separaria o grafo. Portanto, todo par de vértices de  $T$  é ligado por apenas um caminho.

(V)  $\Rightarrow$  (VI) Supondo que  $T$  contém um ciclo, obteria-se um par de vértices ligado por mais de um caminho. A adição de uma aresta  $uv$  encadeada com o único caminho entre  $u$  e  $v$  resulta num ciclo. Se este ciclo não fosse único, a retirada da aresta  $uv$  deixaria dois caminhos distintos entre  $u$  e  $v$ .

(VI)  $\Rightarrow$  (I) Cada aresta  $e$  de  $T$  é uma ponte, pois não está contida em qualquer ciclo. Logo, todo par de vértices de  $T$  é ligado por um único caminho. Se  $T$  contivesse um ciclo, haveria um par de vértices com mais de um caminho. Assim,  $T$  não contém

<sup>1</sup>Uma aresta de um grafo é uma ponte se não pertence a um circuito

ciclos, mas a adição de uma aresta produz um único ciclo. Portanto  $T$  é conexo e não contém ciclos. Logo,  $T$  é uma árvore. ■

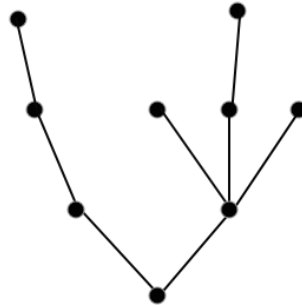


Figura 2.17: Grafo: árvore

**Definição 2.5.2.** Seja  $G = (V, E)$  um grafo conexo não orientado. Uma árvore geradora  $A = (V', E')$  de  $G$  é um subgrafo conexo sem ciclos que possui todos os vértices de  $G$ .

Árvore geradora é definida para componentes conexos de grafos. Um subgrafo de um grafo  $G$  é gerador se contém todos os vértices de  $G$ .

**Teorema 2.5.3.** Todo grafo conexo contém uma árvore geradora.

**Prova:** [3] Seja  $G$  um grafo conexo. Se  $G$  não apresenta circuitos, temos uma árvore geradora. Se  $G$  tem um circuito, então podemos tirar uma aresta do circuito, resultando um subgrafo conexo de  $G$ . Repetindo esse processo até não restar nenhum circuito, obtém-se uma árvore geradora de  $G$ . ■

**Teorema 2.5.4.** Toda árvore geradora de um grafo  $G = (V, E)$  com  $n$  vértices tem exatamente  $n - 1$  arestas.

**Prova:** A prova será feita por indução em  $n$ .

(i) Para  $n = 1$ . Tem-se uma árvore com apenas um vértice e nenhuma aresta.

(ii) Suponha que o teorema é válido para  $n = k, k \geq 1$  (hipótese de indução). Se a árvore tem  $k$  vértices, então ela terá  $k - 1$  arestas. Seja  $A$  uma árvore geradora qualquer com  $k + 1$  vértices e seja  $|E_A|$  o seu número de arestas. Como  $k \geq 1$ , temos  $|E_A| \geq 1$ , a fim de que  $A$  seja conexo. Retirando uma aresta qualquer de  $A$ , temos uma nova árvore geradora  $A'$  com  $k$  vértices e  $|E_A| - 1$  arestas. Se a árvore tem  $k$  vértices, pela hipótese de indução ela deve ter  $k - 1$  arestas. Portanto,  $|E_A| - 1 = k - 1$  e, assim,  $|E_A| = k$ . ■

## 2.6 Coloração

Os problemas de coloração estão associados com a coloração de mapas, de modo que regiões adjacentes não podem ter a mesma cor. Considera-se cada região como um vértice do grafo e cada fronteira entre duas regiões como aresta que liga os dois vértices correspondentes.

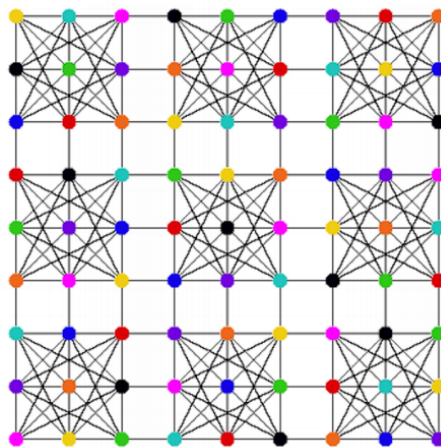
Se  $G$  é um grafo simples, a atribuição de cores para cada um dos vértices, de modo que vértices adjacentes apresentem cores diferentes, é uma coloração para  $G$ . O número cromático de  $G$  é o menor número de cores necessário para colorir  $G$  e é denotado por  $\chi(G)$ . Se  $\chi(G) = 1$ , temos um grafo nulo. Se  $\chi(G) = 2$  temos um grafo bipartido não nulo como mostra o próximo teorema.

**Teorema 2.6.1.** Um grafo  $G$  é bipartido se, e só se,  $\chi(G) = 2$ .

**Prova:** ( $\Rightarrow$ ) Se  $G$  é bipartido, basta corresponder cada um dos dois subconjuntos de vértices a uma cor distinta. ( $\Leftarrow$ ) Se um grafo é tal que  $\chi(G) = 2$ , então basta separar os vértices em dois subconjuntos de acordo com as duas cores usadas na coloração do grafo. Por definição, não podemos ter dois vértices adjacentes com a mesma cor. Desse modo, só haverá arestas de um vértice de um dos subconjuntos a um vértice do outro subconjunto. Logo, o grafo é bipartido. ■

Em termos de coloração de grafos, o teorema das quatro cores foi um desafio que perdurou por mais de cem anos. Uma consequência do Teorema das Quatro Cores é que o número cromático de um grafo planar não é maior que 4. Uma prova do teorema foi feita por K. Appel e W. Haken em 1976. [10]

Uma das aplicações mais conhecidas de coloração está no Sudoku [15]. A resolução de uma instância equivale a encontrar uma 9-coloração num grafo de 81 vértices, conforme indicado na figura 2.18.



**Figura 2.18:** Aplicação de coloração de grafos: Sudoku.

## 2.7 Teoremas, problemas e outros resultados

Alguns teoremas e resultados da Teoria de Grafos tem contribuído para a resolução de problemas de localização, organização de campeonatos esportivos, simulação de redes elétricas, estruturas moleculares e novos compostos químicos.

Leonhard Euler, no século XVIII, provou um teorema que originalmente tratava de arestas, faces e vértices de poliedros convexos. Esse teorema pode também ser aplicado a grafos. A correspondência entre vértices de um poliedro com os vértices de um grafo já foi vista antes. No caso das faces, em um grafo se trata das regiões obtidas pela representação geométrica de grafos com pelos menos um ciclo.

**Teorema 2.7.1. (Relação de Euler).** Se um grafo planar conexo apresenta  $v$  vértices,  $a$  arestas e  $f$  faces, então  $v + f = a + 2$ .

**Prova:** A prova será feita por indução em  $a$  (número de arestas).

(i) Para  $a = 1$ , tem-se  $f = 1$  e  $v = 2$ .

(ii) Suponha que o teorema seja válido para um grafo planar conexo qualquer com menos de  $a + 1$  arestas. Se for uma árvore,  $f = 1$  (apenas uma região, pois árvores são acíclicas) e  $v = a + 1$ . Temos  $v + f - a = a + 1 + 1 - a = 2$ . Se apresentar um ciclo, ao retirar uma aresta, o grafo fica com  $f - 1$  faces. Pela hipótese de indução, a relação vale para o novo grafo. Assim,  $(f - 1) + v = (a - 1) + 2$  e finalmente,  $f + v = a + 2$ . ■

Outra relação importante ocorre em grafo maximal planar, que se trata de um grafo no qual não se pode acrescentar uma aresta sem comprometer sua planaridade.

**Teorema 2.7.2.** Em um grafo planar conexo  $G$ , temos  $a \leq 3v - 6$ . Se  $a = 3v - 6$ , então  $G$  é maximal planar.

**Prova:** Cada aresta é contada duas vezes quando se conta as faces do grafo. Como o grafo é conexo, então cada face tem pelo menos 3 arestas. Assim:

$$3f \leq 2a$$

Pela relação de Euler:

$$f + v - a = 2,$$

$$3f + 3v - 3a = 6$$

Logo,

$$2a - 3a + 3v \geq 6$$

Portanto

$$a \leq 3v - 6$$

■

**Corolário 2.7.3.** Se  $n > 4$ , então  $K_n$  não é planar

**Prova:** Em um grafo completo  $K_n$  temos  $n$  vértices e  $(n-1)n/2$  arestas. Pelo teorema 2.7.2, sabe-se que  $3v - a \geq 6$  em um grafo conexo e planar com  $v$  vértices e  $a$  arestas. Assim,  $3n - n(n-1)/2 = (-n^2 + 7n)/2 \geq 6$ . Os únicos valores naturais que verificam essa desigualdade são 3 e 4. Portanto, todos os grafos completos  $K_n$ , com  $n \geq 5$ , não são planares. ■

Sobre grafos bipartidos, seguem dois teoremas, sendo o primeiro necessário para a compreensão do segundo.

**Teorema 2.7.4.** Um grafo  $G$  é bipartido se, e somente se, não contém ciclos ímpares.

**Prova:** ( $\Rightarrow$ ) Seja  $G$  bipartido. Sejam  $V_1$  e  $V_2$  dois subconjuntos de vértices de  $G$  independentes e disjuntos. Se  $G$  contém um ciclo, este alterna os vértices de  $V_1$  e  $V_2$ . Partindo de um desses subconjuntos, para voltar ao ponto de partida, tem-se um número par de arestas. Nesse caso, o ciclo tem comprimento par. Se  $G$  não apresenta ciclo, não há o que mostrar.

( $\Leftarrow$ ) Seja  $G$  um grafo onde todo ciclo é de comprimento par. Particione os vértices em dois subconjuntos  $V_1$  e  $V_2$  independentes e disjuntos. Se não tiver nenhuma aresta ligando dois vértices de  $V_1$  ou dois vértices de  $V_2$ ,  $G$  é bipartido. Se existir uma outra aresta entre dois vértices  $u$  e  $w$  de  $V_1$  ou de  $V_2$ , obtém-se um ciclo ímpar, o que contraria a hipótese. Portanto, não pode existir outra aresta entre qualquer par de vértices que está em  $V_1$  (igualmente para  $V_2$ ). Logo,  $G$  é bipartido. ■

**Teorema 2.7.5.** Em um grafo planar bipartido conexo  $G$ , temos  $a \leq 2v - 4$ .

**Prova:** Um grafo bipartido possui apenas ciclos pares. Desse modo, cada face tem no mínimo 4 arestas. Assim:

$$4f \leq 2a$$

Pela relação de Euler:

$$f + v - a = 2$$

$$4f + 4v - 4a = 8$$

Logo,

$$2a - 4a + 4v \geq 8$$

Portanto

$$a \leq 2v - 4$$

■

A partir deste teorema, percebe-se que o grafo bipartido completo de seis vértices  $K_{3,3}$  não é planar. Esse resultado soluciona um problema que muitas vezes é apresentado como desafio ou passatempo. Trata-se do problema de conectar três

casas a cada uma de três infraestruturas básicas como energia, água e telefone. Pelo teorema 2.7.5, percebe-se que não há solução para tal desafio, pois  $a = 9$  e  $2v - 4 = 2 \times 6 - 4 = 8$ .

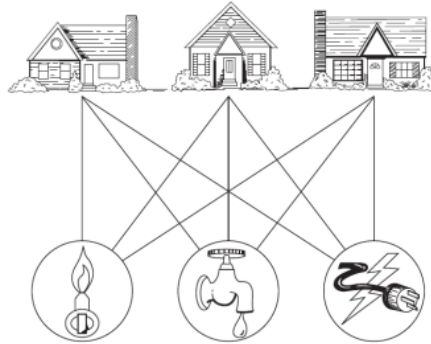


Figura 2.19: O desafio das três casas. [26]

**Corolário 2.7.6.** Se  $G$  é um grafo simples conexo e planar, então  $G$  tem um vértice de grau menor ou igual a 5.

**Prova:**

1º caso:  $G$  com um ou dois vértices. A afirmativa é válida.

2º caso:  $G$  com pelo menos três vértices. Pelo teorema 2.7.2, sabe-se que  $a \leq 3v - 6$  e, daí,  $2a \leq 6v - 12$ . Supondo que o grau de cada vértice fosse maior ou igual a 6, o grafo teria um grau total maior ou igual a  $6v$ . A representação do grau total do grafo com a quantidade de arestas resultaria em  $2a \geq 6v$ . Tem-se uma contradição. Portanto, um vértice, pelo menos, deve ter grau menor ou igual a 5. ■

Considere a hipótese de que os vértices de um grafo possuam grau maior ou igual a 3. Tal consideração é essencial para o resultado a seguir.

**Teorema 2.7.7.** Sendo o mapa um grafo plano, todo mapa tem ao menos duas faces com número máximo de 5 arestas cada

**Prova:** [40] Considere um grafo planar com  $v$  vértices,  $a$  arestas e  $f$  faces. Se cada vértice é extremidade de pelo menos três arestas e cada aresta possui dois vértices como extremidades, ao contar o menor número possível de extremidades de arestas do mapa, conta-se o triplo do número total de vértices.

Assim,  $3v \leq 2a$  e, daí,  $3v - 2a \leq 0$ . Pela Relação de Euler, temos:

$$6 = 3v - 3a + 3f \leq 2a - 3a + 3f = 3f - a$$

$$3f - a \geq 6$$

Sendo  $f_n$  o número de faces delimitadas por  $n$  arestas, temos:

$$f_1 + f_2 + f_3 + \dots = f$$



Ao contar as arestas de todas as faces, cada aresta será contada duas vezes:

$$f_1 + 2f_2 + 3f_3 + \dots = 2a$$

Levando em conta que

$$6f - 2a \geq 12,$$

temos

$$(6f_1 + 6f_2 + 6f_3 + 6f_4 + \dots) - (f_1 + 2f_2 + 3f_3 + \dots) \geq 12$$

Portanto

$$5f_1 + 4f_2 + 3f_3 + 2f_4 + f_5 - f_7 - 2f_8 - 3f_9 - \dots \geq 12$$

Desse modo,  $f_1, f_2, f_3, f_4, f_5$  não podem ser todos nulos. Se  $5f_1 + 4f_2 + 3f_3 + 2f_4 + f_5 \geq 12$ , na lista  $f_1, f_2, f_3, f_4, f_5$  há pelo menos dois não nulos ou um deles é maior ou igual a 12. A conclusão imediata é que o mapa tem ao menos duas faces com um máximo de 5 arestas. ■

Essa prova mostra que todo mapa tem dois países com no máximo 5 vizinhos cada. Essa propriedade é essencial para provar o teorema das cinco cores que diz que um grafo  $G$  é planar simples quando seu número cromático é menor ou igual a 5 [23].

| D.A.   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|--------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|
| Mat.   | • |   |   |   |   |   |   | • |   |    |    | •  |    |    |    | •  |
| Port.  |   |   |   | • |   |   |   |   |   |    | •  |    |    |    |    | •  |
| Inglês |   |   |   |   | • |   | • |   |   | •  |    |    |    |    |    | •  |
| Geo.   |   |   |   | • | • |   | • |   |   |    |    |    | •  |    |    |    |
| Hist.  |   |   | • |   |   |   |   |   |   | •  |    |    |    |    |    | •  |
| Fis.   |   |   | • |   | • |   |   |   |   |    |    | •  |    |    |    |    |
| Qui.   |   | • |   |   |   |   |   | • | • |    |    | •  |    |    |    | •  |
| Bio.   |   | • |   |   |   | • |   |   |   |    |    |    |    |    |    |    |

Tabela indicando os alunos matriculados por disciplina.

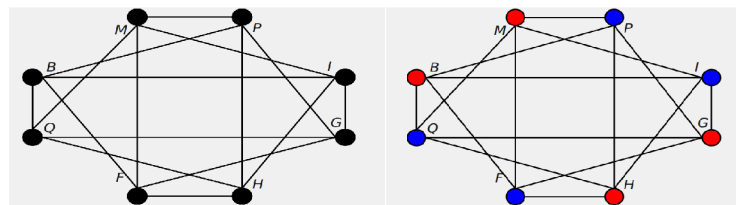


Figura 2.20: Aplicação de coloração de grafos: programação de horários. [15]

Uma das aplicações de coloração de grafos está na programação de horários. Esse tipo de problema consiste em determinar os horários das disciplinas de modo que todos os alunos matriculados assistam as aulas sem conflitos de horários. No grafo que representa a situação problema, as disciplinas são os vértices. Quando existir alunos em comum, os vértices são adjacente, como ilustrado no exemplo da figura 2.20.

Sobre os poliedros de Platão, os grafos platônicos estão associados a cada um deles: tetraedro, hexaedro, octaedro, dodecaedro e icosaedro. Afirma-se que existem somente cinco tipo de poliedros de Platão ou regulares.

**Teorema 2.7.8.** Há somente cinco poliedros regulares ou de Platão.

**Prova:** [41] Seja  $x$  o número de lados de cada face do poliedro e  $y$  o número de arestas incidentes em cada vértice. Sejam  $A$ ,  $F$  e  $V$ , respectivamente, o número de

arestas, faces e vértices do poliedro. Sabe-se que:

$$2A = xF = yV.$$

Assim,

$$A = xF/2$$

e

$$V = xF/y$$

Pela relação de Euler, tem-se:

$$xF/2 - xF/y + F = 2$$

de onde segue que

$$F = 4y/(2y + 2x - yx)$$

Uma vez que o numerador  $4y$  é sempre positivo, deve-se ter o denominador  $2y + 2x - yx$  maior que zero. Logo,  $2x/(x - 2) > y$ .

Sabe-se que  $y \geq 3$ , o que implica em  $x < 6$ . Desse modo, as possibilidades são:

$$x = 3 \rightarrow F = 4y/(6 - y) \rightarrow y = 3 \rightarrow F = 4(\text{Tetraedro})$$

$$x = 3 \rightarrow F = 4y/(6 - y) \rightarrow y = 4 \rightarrow F = 8(\text{Octaedro})$$

$$x = 3 \rightarrow F = 4y/(6 - y) \rightarrow y = 5 \rightarrow F = 20(\text{Icosaedro})$$

$$x = 4 \rightarrow F = 4y/(8 - 2y) \rightarrow y = 3 \rightarrow F = 6(\text{Hexaedro})$$

$$x = 5 \rightarrow F = 4y/(10 - 3y) \rightarrow a = 3 \rightarrow F = 12(\text{Dodecaedro})$$

■

Outro problema é o de conexão de peso mínimo, associado à definição 2.5.2 de árvore geradora. Dado  $G$  um grafo valorado qualquer, qual é a árvore geradora de menor valor? A resposta desse problema é obtida por meio do Algoritmo de Kruskal. Nesse procedimento, escolhe-se uma aresta de menor valor, de modo que não se forme ciclo, e acrescenta-se à árvore. A árvore será finalizada quando  $a - 1$  arestas tiverem sido tomadas. Essa é a árvore ótima, isto é, a árvore geradora minimal.

Em relação aos grafos eulerianos, o problema Chinês do Carteiro é uma importante aplicação. Dado um grafo pesado <sup>2</sup> conexo  $G = (V, E)$ , deseja-se encontrar um caminho fechado de peso mínimo que atravesse cada aresta de  $G$  ao menos uma vez. Chama-se percurso do carteiro qualquer caminho fechado que atravesse cada

<sup>2</sup>Se às arestas tiverem associado um peso ou custo, o grafo passa a chamar-se grafo pesado

aresta de  $G$  pelo menos uma vez. Esse caminho não precisa ser necessariamente de peso mínimo. Um caminho nessas condições é denominado percurso ótimo do carteiro chinês.

Se o grafo  $G$  for euleriano, então qualquer circuito euleriano é um percurso ótimo do carteiro chinês. Se  $G$  não for euleriano, constrói-se um grafo euleriano  $G_1$  duplicando algumas arestas de  $G$ . Tais arestas devem ser escolhidas de modo a obter um grafo euleriano com peso mínimo. A ideia de fazer o carteiro percorrer ruas repetidas de forma econômica reforça a ideia de que não há necessidade de visitar cada aresta mais do que duas vezes. [39]

**Corolário 2.7.9.** Um grafo  $G$  conexo é semieuleriano se, e somente se, possui no máximo um par de vértices de grau ímpar.

**Prova:**

( $\Rightarrow$ ) Seja  $G$  um grafo de comprimento  $m$  semieuleriano.  $G$  representa uma trilha máxima aberta de comprimento  $m$  que começa em  $v_k$  e termina em  $v_p$ . Como a trilha é aberta, temos que  $v_k \neq v_p$ . Logo, tanto  $v_k$  e  $v_p$  possuem graus ímpares, pois a trilha não volta por onde começou.

( $\Leftarrow$ ) Seja  $G$  o grafo conexo com um par de vértices de grau ímpar,  $v_k$  e  $v_p$ . Pelo teorema de Euler sobre grafos eulerianos, acrescentando uma aresta de  $v_k$  a  $v_p$ , os graus de todos os vértices se tornam pares. Assim, existe uma trilha fechada de comprimento  $m + 1$  que começa em  $v_k$  e termina em  $v_p$ . Também existe uma trilha aberta de comprimento  $m$  que começa em  $v_k$  e termina em  $v_p$ , descrevendo assim um caminho semieuleriano. ■

Um grafo hamiltoniano é a solução do problema do Caixeiro Viajante. Nesse problema, supõe-se que a área de venda de um caixeiro viajante inclua várias cidades. Muitas delas estão conectadas aos pares por rodovias. O caixeiro precisa visitar cada cidade. Procura-se estabelecer uma viagem que o leve novamente para o ponto de partida, de modo que ele visite cada cidade apenas uma vez.

O problema do Caixeiro Viajante pode ser modelado por meio de um grafo  $G(V,E)$ , sendo:

$$V = \{x/x \text{ é uma cidade}\}$$

$$E = \{(x_1, x_2)/\text{existe uma estrada que conecta as cidades } x_1 \text{ e } x_2\}$$

A estrada que conecta as cidades  $x_1$  e  $x_2$  não passa por nenhuma outra cidade. Desse modo, resolve-se o problema verificando se  $G$  é hamiltoniano. O teorema a seguir fornece uma condição suficiente para um grafo ser hamiltoniano.

**Teorema 2.7.10. (Teorema de Dirac)** Se  $G$  é um grafo simples de ordem  $n \geq 3$  e  $d(v) \geq n/2$  para todo  $v \in V(G)$ , então  $G$  é hamiltoniano.

**Prova:**

A demonstração será feita por absurdo. Supondo que a afirmação seja falsa. Assim, existe um grafo simples não hamiltoniano maximal  $G$  de ordem  $n \geq 3$  que

satisfaz a condição do teorema. Em um grafo maximal não hamiltoniano, qualquer par de vértices não adjacentes  $u, v$  em  $G$  proporciona um grafo  $G + uv$  hamiltoniano, o que implica na existência de um caminho que percorre todos os vértices.

É óbvio que  $G$  não é completo, pois caso contrário seria hamiltoniano. Portanto, existem vértices  $u$  e  $v$  não adjacentes em  $G$ . Seja  $H := G + uv$ . Pela maximalidade de  $G$ , temos que  $H$  é hamiltoniano. Assim, todo circuito hamiltoniano em  $H$  precisa conter a aresta  $uv$ . Então  $G$  tem um caminho hamiltoniano ( $u = v_1, v_2, v_3, \dots, v_n = v$ ). Se  $v_i$  é adjacente a  $u$ , então  $v_{i-1}$  não é adjacente a  $v$ . Caso contrário, teríamos um circuito hamiltoniano em  $G$ , contrariando a escolha de  $G$ .

Portanto, qualquer que seja o vértice adjacente a  $u$ , existe um vértice pertencente a  $V(G) \setminus \{v\}$  que não é adjacente a  $v$ . Assim,  $d(v) \leq n - 1 - d(u)$ . Sendo  $d(u) \geq n/2$ , temos  $d(v) \leq n - 1 - n/2 = n/2 - 1$ . Dessa contradição, conclui-se que a afirmação é verdadeira. ■

Em um grafo  $G = (V, E)$ , um conjunto independente de vértices  $V_{IND}$  de  $G$  é um subconjunto de  $V$  em que não existe nenhuma aresta entre qualquer par de elementos de  $V_{IND}$ , conforme exemplo da figura 2.21. Um conjunto independente (maximal) em um grafo é um conjunto de vértices não adjacentes entre si que não está estritamente contido em outros conjuntos independentes. O número de independência, denotado por  $\alpha(G)$ , é o tamanho do maior conjunto independente. [35]

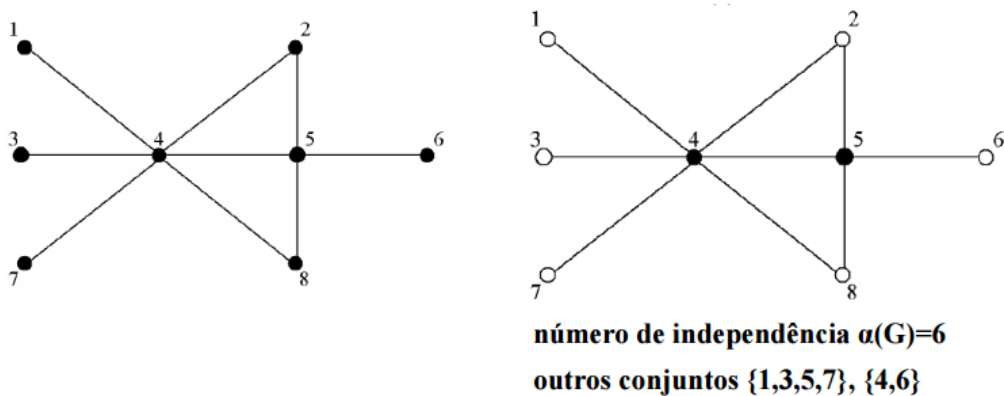


Figura 2.21: Conjunto independente. [35]

**Teorema 2.7.11.** Se  $\alpha(G) > n/2$ , então  $G$  não é hamiltoniano ( $n$  é o número de vértices).

**Prova:** Em um conjunto independente, dois vértices fazem parte de um mesmo percurso somente se houver pelo menos um vértice intermediário. Esse vértice deverá ser vizinho de ambos. Se o grafo  $G$  apresenta um conjunto independente com mais de  $n/2$  vértices, não existirá vértices intermediários em quantidade suficiente de modo que exista um percurso fechado. Logo,  $G$  não é hamiltoniano. ■

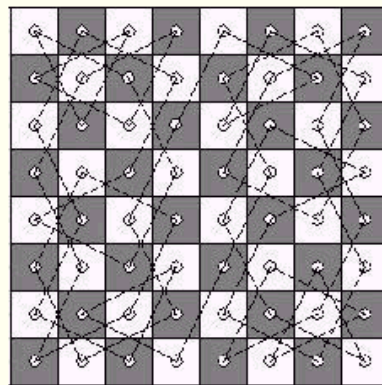
Outro problema cuja solução poder ser modelada com grafos está relacionado ao jogo de xadrez. O problema do cavalo questiona se é possível percorrer todo o

tabuleiro visitando cada casa uma única vez e retornando à casa de origem. No jogo de xadrez, o movimento do cavalo ocorre duas casas em um sentido (horizontal ou vertical) e uma casa no outro sentido (vertical ou horizontal). Esse problema pode ser modelado por um grafo  $G(V, E)$  sendo:

$$V = \{x/x \text{ é uma casa do tabuleiro de xadrez}\}$$

$$E = \{(x_1, x_2)/x_2 \text{ é atingida por um único movimento de cavalo}\}$$

A solução do problema consiste em verificar se  $G$  é hamiltoniano. O grafo  $G$  contém 64 vértices e 168 arestas, com inúmeros ciclos hamiltonianos, um dos quais está indicado na figura 2.22.



**Figura 2.22:** Um dos ciclos hamiltonianos na solução do problema do cavalo.

[4]

O problema do menor caminho consiste em obter o caminho de menor custo entre dois vértices  $v_1$  e  $v_2$  em um grafo  $G$  com pesos nas arestas. O algoritmo que soluciona este problema foi determinado pelo cientista da computação Edsger Wybe Dijkstra em 1952. O algoritmo de Dijkstra considerava que para ir do vértice  $v_1$  ao vértice  $v_2$  deve-se partir do vértice  $v_1$  e, ao longo do processo, adicionar a ele os vértices cujos menores custos (ou distâncias de  $v_1$ ) já foram determinadas. Se o vértice  $v_i$  está fora desse conjunto, toma-se o seu custo a partir de  $v_1$  e avalia-se se esse custo efetivamente é maior ou menor que vértice atual (vértice que está sendo considerado como o de menor custo até o momento). O procedimento se encerra ao chegar em  $v_2$ .

No exemplo da figura 2.23, foram necessárias seis iterações para determinar o menor caminho. No grafo em questão, deseja-se sair do vértice 5 e chegar ao vértice 2. A solução é dada pela sequência de vértices 5, 6, 4, 1, 3, 2 com custo correspondente igual a 9.

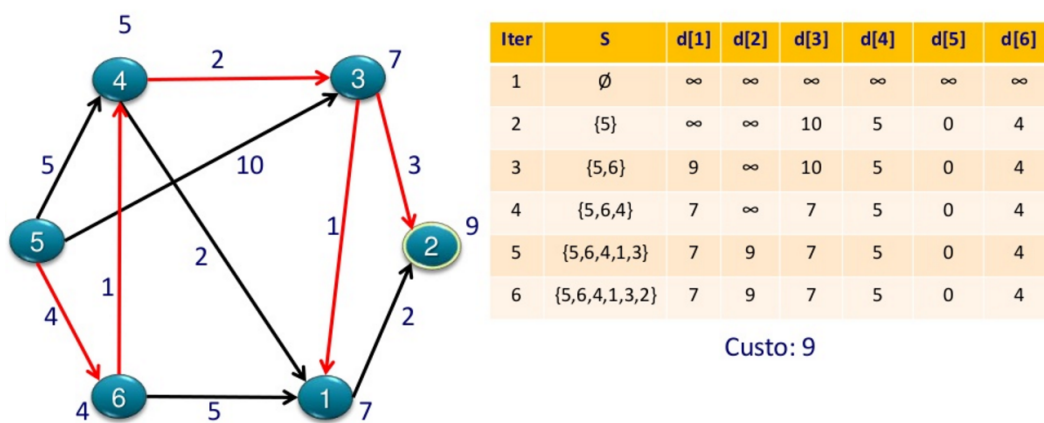


Figura 2.23: Exemplo de aplicação do algoritmo de Dijkstra. [16]

# Grafos, problemas e algoritmos

---

Os grafos são estruturas discretas que podem ser utilizadas na representação e resolução de vários problemas em Topologia, Programação, Lógica, Cartografia, entre outras. Os algoritmos associados a esses problemas permitem verificar a existência de caminhos para ir de um objeto a outro seguindo determinadas conexões.

O modelo básico de grafo permite verificar quais são os elementos envolvidos (vértices) e quais elementos estão associados (arestas). “Um modelo é uma estrutura de uma realidade com a qual nos interessa trabalhar, construída de modo a conter aquilo que mais nos interessa e de forma que nos permita obter as respostas de que necessitamos”. [11] Os modelos podem se apresentar prontos, podem ser adaptados ou mesmo construídos. Resolver um modelo consiste em encontrar respostas ao problema que a ele está associado. Porém, nem todo modelo tem solução e, mesmo sendo solucionável, o procedimento pode ser difícil de se encontrar.

Com o avanço da Ciência da Computação, alguns problemas, como o da coloração de grafos usando apenas quatro cores, passaram a ser abordados com êxito por meio de algoritmos.

## 3.1 Algoritmos

**Definição 3.1.1.** Um algoritmo é um conjunto de passos para realizar uma tarefa, aplicado em etapas repetitivas e com eventuais desvios lógicos (aberturas de caminhos para duas opções, conforme a condição que ele contém seja satisfeita ou não). [11]

Exemplos simples de algoritmos podem ser encontrados no cotidiano como uma receita de bolo ou quem vai da casa para escola:

- passo 1: pegar ônibus até a estação.
- Passo 2: pegar o metrô.
- Passo 3: andar da estação até a escola.

Na Ciência da Computação, um algoritmo é um conjunto de passos para que um programa de computador, utilizado determinada linguagem, possa realizar uma tarefa. Encontrar bons algoritmos e saber quando aplicá-los é fundamental para a escrita de programas relevantes.

Alguns conceitos são essenciais para a compreensão e prática na implementação dos algoritmos envolvendo grafos que serão apresentados nesse capítulo.

- **Variável:** espaço reservado na memória do computador para guardar informações que serão utilizadas ao longo do código do programa. Pode ter seu valor alterado ao longo do programa de acordo com a conveniência.
- **Matriz (array):** coleção de variáveis de mesmo tipo, acessíveis com um único nome e armazenados contiguamente na memória. Esse conceito é diferente do conceito de matriz em Álgebra Linear.
- **Vetor:** matriz de uma só dimensão. A individualização de cada variável de um vetor é feita por meio do uso de índices.
- **Instância de um problema computacional:** é uma caso particular do problema, espécie de exemplo, amostra ou ilustração. Cada instância é definida por um conjunto de dados.
- **Loop:** estruturas de repetição executadas até que alguma condição seja atendida.
- **Programação dinâmica:** nome fantasia para recursão com uma tabela. Em vez de resolver subproblemas recursivamente, resolve-se sequencialmente e armazena-se suas soluções em uma tabela.
- **Complexidade de tempo:** medida que expressa a eficiência em termos de tempo de execução. Na análise de algoritmos verifica-se o número de operações consideradas relevantes que foram realizadas pelo algoritmo. Em seguida, expressa-se esse número como uma função de  $n$ , sendo  $n$  um parâmetro que caracteriza o tamanho da entrada do algoritmo.

**Exemplo 3.1.2.** Um recepcionista de cinema precisa conferir os bilhetes e mostrar qual é a sala que vai passar o filme escolhido. Caso o cliente esteja 30 minutos adiantado, o recepcionista informa que a sala ainda não está aberta. Em caso de atraso de 30 minutos, ele é informado que não pode entrar.

Nesse algoritmo, há um fluxo que pode seguir diferentes caminhos de acordo com a situação encontrada. Tratam-se dos desvios lógicos, aqui representados pelo “se”.

Esse algoritmo possui uma representação gráfica, denominada fluxograma. Nele, os losangos indicam as decisões que são tomadas a fim de executar o passo seguinte, conforme indicado na figura 3.2.

**Exemplo 3.1.3.** Algoritmo da multiplicação de dois números inteiros positivos.



**Algoritmo Recepcionista de Cinema****Início**

1 – Solicitar ao cliente o bilhete do filme.

2 – Conferir a data e o horário do filme no bilhete.

**Se** data/hora atual > data/hora do filme + 30 minutos **Então**

3 – Informar ao cliente que o tempo limite para entrada foi excedido.

4 – Não permitir a entrada.

**Senão Se** data/hora atual < data/hora do filme – 30 minutos **Então**

5 – Informar ao cliente que a sala do filme ainda não foi liberada para entrada.

6 – Não permitir a entrada.

**Senão**

7 – Permitir a entrada.

8 – Indicar ao cliente onde fica a sala do filme.

**Fim-Se****Fim**

Figura 3.1: Exemplo de algoritmo: recepcionista de cinema. [19]

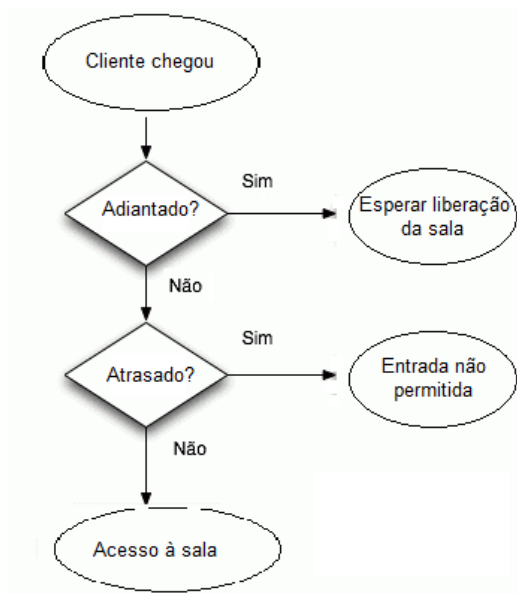


Figura 3.2: Exemplo de fluxograma: recepcionista de cinema.

A boa definição do algoritmo, isto é, clareza nos procedimentos, é essencial para a execução eficiente. Uma calculadora, ao realizar a multiplicação de dois números positivos, executa um algoritmo que efetua somas até um número determinado de vezes conforme indicado na figura 3.3.

```
Algoritmo Multiplicação de números inteiros positivos  
Declaração de variáveis  
    numero1, numero2, resultado, contador: Inteiro  
Início  
    ler(numero1)  
    ler(numero2)  
    resultado <- 0  
    contador <- 0  
    Enquanto contador < numero2 Faça  
        resultado <- resultado + numero1  
        contador <- contador + 1  
    Fim-Enquanto  
    escrever(resultado)  
Fim
```

**Figura 3.3:** Exemplo de algoritmo na computação: multiplicação de dois números inteiros positivos. [19]

No algoritmo da figura 3.3 temos quatro variáveis:

*numero1, numero2, resultado, contador : Inteiro.*

A variável armazena dados em um espaço alocado na memória. O símbolo `<-` atribui valor à variável enquanto o comando `ler(numero1)` indica que o algoritmo está lendo o que o usuário digita e, em seguida, armazena na variável indicada. O comando `Enquanto` é uma estrutura de repetição para controle de fluxo. Finalmente, o comando `escrever(resultado)` exibe o valor da variável `resultado`.

A análise de algoritmo proporciona uma medida objetiva de desempenho que é proporcional ao tempo de execução do algoritmo. Sabe-se que o tempo de execução depende do algoritmo, do conjunto de instruções do computador, da qualidade do compilador e da habilidade de quem realiza a programação. Entretanto, o tempo de execução de um algoritmo, pode ser dimensionado pela quantidade de operações primitivas executadas. Por essa razão, a complexidade também é conhecida como esforço requerido ou quantidade de trabalho. No pior caso, considera-se a instância que faz o algoritmo funcionar mais lentamente. A sua média é dada pela consideração de todas as possíveis instâncias e seus respectivos tempos.

O cálculo da complexidade se dá através do espaço de memória que está sendo utilizado pelo algoritmo ou pelo tempo necessário para computador o resultado segundo uma instância do problema de tamanho  $n$  (parâmetro que caracteriza o

tamanho da entrada do algoritmo). A lista a seguir apresenta uma comparação entre as complexidades (notação  $O$ ).

- $O(1)$  : constante, isto é, não cresce com o tamanho do problema.
- $O(\log_2(\log_2(n)))$  : muito rápido.
- $O(\log_2(n))$  : logarítmico (muito bom).
- $O(n)$  : linear (se algo não pode ser determinado sem o exame total da entrada, é o melhor que se pode esperar).
- $O(n \log_2(n))$  : considerado o limite de diversos problemas práticos como a ordenação de uma coleção numérica, por exemplo.
- $O(n^k)$  : polinomial (aceitável se  $n$  for pequeno).
- $O(k^n)$ ,  $O(n!)$ ,  $O(n^n)$  : exponencial (deve ser evitado).

A seguinte hierarquia de funções pode ser definida do ponto de vista assintótico:

$$1 \prec \log_2(\log_2(n)) \prec \log_2(n) \prec n^\epsilon \prec n^c \prec n(\log_2(n)) \prec c^n \prec n^n \prec c^{c^n}.$$

onde  $\epsilon$  e  $c$  são constantes em que  $0 < \epsilon < 1 < c$ .

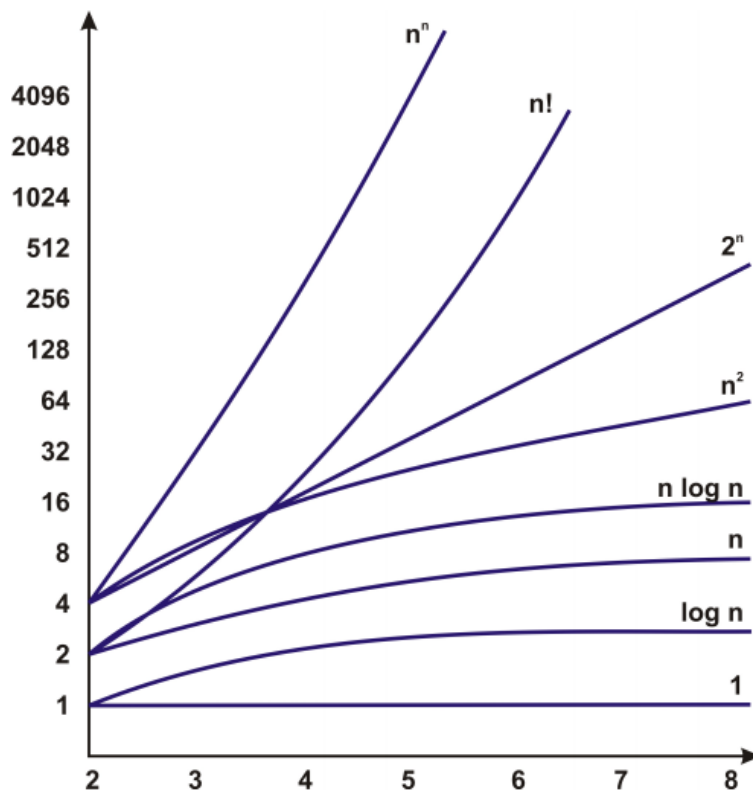


Figura 3.4: Hierarquia de funções. [43]

### 3.2 Matriz de Adjacência.

A representação computacional de um grafo é feita através de uma estrutura que corresponde de forma única a um grafo dado, além da possibilidade de se armazenar e manipular em um computador. No entanto, a representação por meio de diagramas de pontos de um grafo, não é viável de armazenar e nem manipular. Nesse caso, a matriz de adjacência e a lista de adjacências são estruturas usadas como forma de representação computacional de um grafo.

**Definição 3.2.1.** Seja  $G$  um grafo com  $n$  vértices e  $m$  arestas. A matriz de adjacência, de ordem  $n \times n$ , denotada por  $X = [x_{ij}]$  é definida como  $x_{ij} = 1$  se existe uma aresta entre os vértices  $v_i$  e  $v_j$  ou  $x_{ij} = 0$  caso contrário.

Para grafos com pesos em suas arestas, cada termo  $a_{ij}$  contém o rótulo ou peso correspondente a cada aresta. Caso não exista uma aresta de  $i$  para  $j$ , atribui-se zero para o termo  $a_{ij}$ .

A matriz de adjacência deve ser usada para grafos densos, isto é, grafos em que o número de arestas é próximo do número máximo. É muito útil para algoritmos nos quais precisa-se saber com agilidade se existe uma aresta ligando dois vértices. Nesse caso, examinar a matriz tem complexidade de tempo  $O(n^2)$  tendo em vista que a ordem da matriz de adjacência é  $n \times n$ . Observa-se também que os elementos da diagonal principal são todos nulos quando o grafo não possui laços. Se há um laço em um vértice  $v_i$  temos  $x_{ii} = 1$ . E em um grafo simples, o grau de um vértice é igual à soma dos elementos de sua fila (linha ou coluna) correspondente.

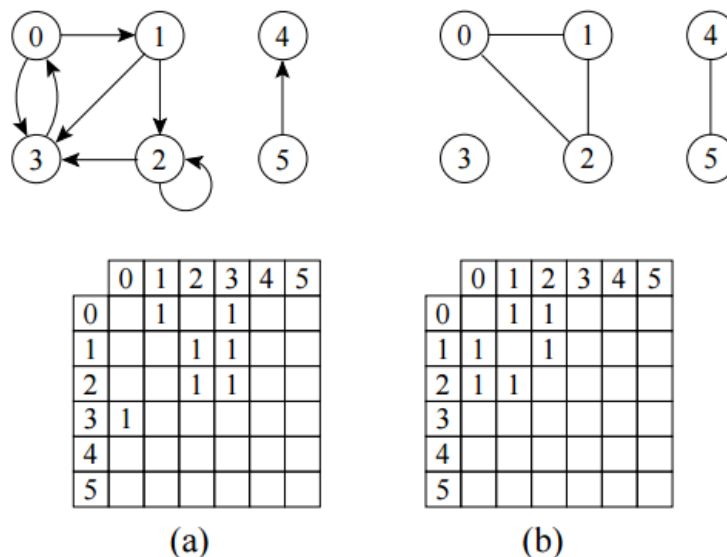


Figura 3.5: Exemplo de matriz de adjacência. [33]

Além do grafo, existe o conceito de digrafo ou *quiver*. A distinção entre os conceitos consiste no fato de que as arestas do grafo são simplesmente formadas por

dois vértice, enquanto que no digrafo, as arestas possuem um vértice inicial e um vértice final (arestas dirigidas). Sendo assim, é possível afirmar que dado um digrafo  $D$  com  $n$  vértices, cada termo da matriz de adjacência  $X$  é definido por  $x_{ij} = 1$  se existe uma aresta direcionada do vértice  $v_i$  para o vértice  $v_j$  ou  $x_{ij} = 0$  caso contrário.

No caso do digrafo, a matriz só será simétrica se o digrafo for simétrico. O grau de saída de um vértice  $v_i$  é dado pela soma dos elementos da linha  $i$  e o grau de entrada, pela soma dos elementos da coluna  $i$ . A transposta da matriz de adjacência de um digrafo  $D$  é a matriz de adjacência do digrafo obtido pela inversão da orientação das arestas de  $D$ .

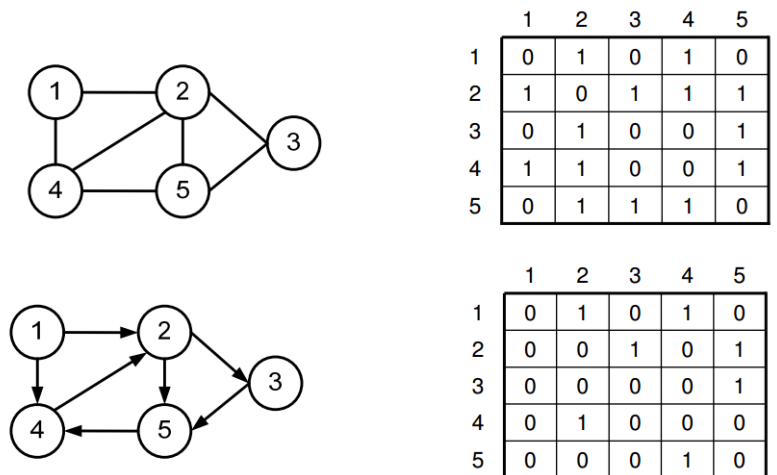


Figura 3.6: Matriz de adjacência para grafo e para digrafo. [28]

A lista de adjacência consiste em um vetor de  $n$  listas para um grafo de  $n$  vértices. Em cada posição  $i$  existe uma lista onde cada elemento é um vértice adjacente ao vértice  $i$ . Para grafos não orientados, as adjacências são armazenadas em ambos os vértices de adjacência. Em digrafos, somente as arestas de saída devem ser armazenadas, conforme exemplo da figura 3.6.

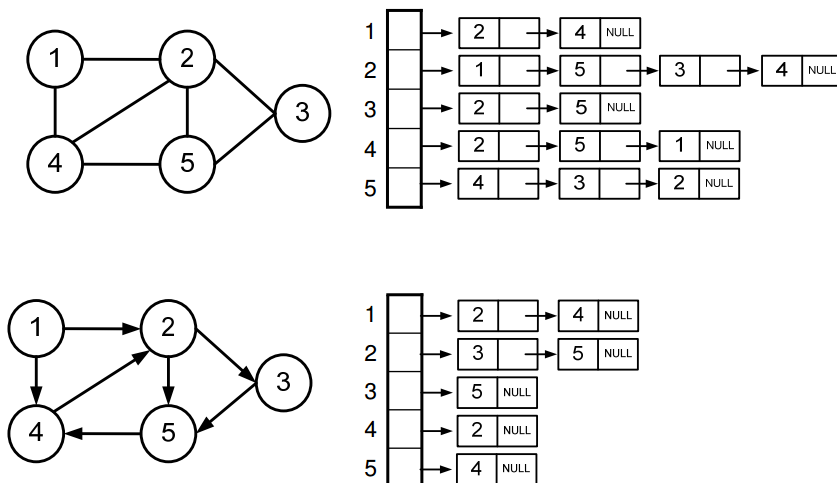


Figura 3.7: Lista de adjacências para grafo e para digrafo. [28]

Na figura 3.7, NULL representa algo sem um valor definido. Nas listas de adjacências, os vértices, em geral, são armazenados em uma ordem arbitrária. Por ser compacta, é muito utilizada em grande parte das aplicações. Sua complexidade de espaço é  $O(n + m)$  para um grafo com  $n$  vértices e  $m$  arestas. Por essa razão, é indicada para grafos esparsos (aqueles cujos complementos são densos). Um ponto negativo é a possibilidade de complexidade de tempo  $O(n)$  para determinar a existência ou não de uma aresta entre os vértices  $i$  e  $j$  uma vez que pode existir  $n$  vértices na lista de adjacentes do vértice  $i$ .

Ao comparar a matriz de adjacência com a lista de adjacências, em termos de complexidade, a matriz requer espaço suficiente para armazenar todas as arestas do grafo completo ( $O(n^2)$ ). A lista requer espaço apenas para armazenar as arestas existentes ( $O(n + m)$ ). Desse modo, grafos pequenos usualmente são representados por matrizes. Os grafos com arestas paralelas (aquelas que começam num mesmo vértice e terminam em um mesmo vértice) não pode, ser armazenados por meio de matriz de adjacência.

A escolha da representação depende do algoritmo que será implementado e da natureza do grafo (se ele é disperso ou denso). Nas figuras 3.8 e 3.9, são apresentados exemplos de implementação do preenchimento de uma matriz de adjacência considerando uma entrada de vários  $x, y$ , indicando relação entre  $x - y$  ( $y - x$ ) até que  $x = 0$  e  $y = 0$ . Observa-se que o grafo não é orientado.

```

para  $i \leftarrow 1$  até  $N$ , faça

    para  $j \leftarrow 1$  até  $N$ , faça

         $matriz[i][j] \leftarrow 0$ 

    fim-para

fim-para

enquanto (recebe  $x, y$  e  $x \neq 0$ ), faça

     $matriz[x][y] \leftarrow 1$ 

     $matriz[y][x] \leftarrow 1$ 

fim-enquanto

```

Figura 3.8: Implementação de matriz de adjacências. [29]

Na figura 3.9, o operador ++ consiste em incrementar a variável depois da instrução atual.

```

para  $i \leftarrow 1$  até  $N$ , faça
     $grau[i] \leftarrow 0$ 
fim-para

enquanto (recebe  $x, y$  e  $x \neq 0$ ), faça
     $lista[x][grau[x]++] \leftarrow y$ 
     $lista[y][grau[y]++] \leftarrow x$ 
fim-enquanto

```

Figura 3.9: Implementação de matriz de adjacências. [29]

### 3.3 Algoritmo de Dijkstra e a busca de caminho mais curto

O algoritmo de Dijkstra calcula o caminho com menor custo entre os vértices de um grafo. Dado um vértice como origem da busca, o procedimento calcula o custo mínimo deste vértice até todos os demais vértices do grafo. Apresenta pontos positivos como a simplicidade e boa performance. Um ponto negativo é falta de garantia de uma solução exata com arestas de pesos negativos.

O algoritmo se inicia por meio de uma estimativa para o custo mínimo. Em seguida, vai ajustando a estimativa durante a execução. Para isso, considera o fechamento de um vértice apenas quando obtém-se um caminho de custo mínimo do vértice escolhido como origem até o vértice em questão. Desse modo, mantém-se dois conjuntos de vértices:

- $S$  é o conjunto dos vértices para os quais já foram determinados o caminho mais curto.
- $Q = V - S$  é o conjunto dos vértices que ainda estão na fila.

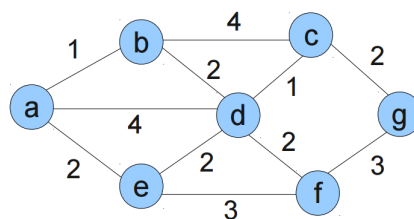


Figura 3.10: Caminho mínimo do vértice  $a$  até  $g$  - exemplo do resultado obtido pelo algoritmo de Dijkstra. [8]

No exemplo da figura 3.10 tem-se um grafo em que busca-se o menor caminho a partir do vértice  $a$  até o vértice  $g$ . Na figura 3.11 está a tabela com passos e distâncias correspondentes à aplicação do algoritmo de Dijkstra no grafo dado. Percebe-se a manutenção de dois conjuntos, sendo que o  $S$  (vértices já determinados) está indicado e o  $Q$  (vértices em fila) está implícito. A indicação das distâncias em cada passo permite conhecer o menor caminho até aquele momento para cada vértice. Só assim o vértice é adicionado ao conjunto  $S$ , mantendo assim, as distâncias atualizadas até o final. No grafo da figura 3.10 o menor caminho tem distância igual a 6 e não passa pelos vértices  $e$  e  $f$ .

| Conjunto S      | d(a) | d(b) | d(c) | d(d) | d(e) | d(f) | d(g) |
|-----------------|------|------|------|------|------|------|------|
| {}              | 0    | inf  | inf  | inf  | inf  | inf  | inf  |
| {a}             | -    | 1    | inf  | 4    | 2    | inf  | inf  |
| {a,b}           |      | -    | 5    | 3    | 2    | inf  | inf  |
| {a,b,e}         |      |      | 5    | 3    | -    | 5    | inf  |
| {a,b,e,d}       |      |      | 4    | -    |      | 5    | inf  |
| {a,b,e,d,c}     |      |      |      | -    |      | 5    | 6    |
| {a,b,e,d,c,f}   |      |      |      |      |      | -    | 6    |
| {a,b,e,d,c,f,g} |      |      |      |      |      |      | -    |

Figura 3.11: Caminho mínimo do vértice  $a$  até  $g$  - tabela com passos e distâncias do algoritmo de Dijkstra. [8]

O algoritmo de Dijkstra está formalmente exposto na figura 3.12. O vetor utilizado é denominado *anterior*, uma vez que trata da última linha da tabela que contém os passos e distâncias atualizadas. O conjunto  $A$ , denominado *aberto* contém todos os vértices do início. O conjunto  $F$ , denominado *fechado* é vazio. Os índices numéricos são utilizados para determinar os vértices.

```

Algoritmo
início
     $d_{11} \leftarrow 0; d_{1i} \leftarrow \infty \forall i \in V - \{1\};$  < distância origem-origem zero; distâncias a partir da origem infinitas >
     $A \leftarrow V; F \leftarrow \emptyset; anterior(i) \leftarrow 0 \forall i;$ 
    enquanto  $A \neq \emptyset$  fazer
        início
             $r \leftarrow v \in V \mid d_{1r} = \min_{i \in A} [d_{ij}]$  < acha o vértice mais próximo da origem >
             $F \leftarrow F \cup \{r\}; A \leftarrow A - \{r\};$  < o vértice r sai de Aberto para Fechado >
             $S \leftarrow A \cap N^+(r)$  < S são os sucessores de r ainda abertos >
            para todo  $i \in S$  fazer
                início
                     $p \leftarrow \min [d_{1i}^{k-1}, (d_{1r} + v_{ri})]$  < compara o valor anterior com a nova soma >
                    se  $p < d_{1i}^{k-1}$  então
                        início
                             $d_{1i}^k \leftarrow p; anterior(i) \leftarrow r;$  < ganhou a nova distância ! >
                        fim;
                    fim;
                fim;
            fim;
        fim.
    
```

Figura 3.12: Implementação do algoritmo de Dijkstra. [11]



Percebe-se no algoritmo formalizado que as distâncias sempre são consideradas a partir da origem. Os vértices que estão em fila vão sendo tomados com base na proximidade com o vértice de origem (partindo do mais próximo). O vértice sai da fila para o conjunto  $S$  quando, após a comparação das distâncias atualizadas, percebe-se que o menor caminho foi encontrado até aquele vértice. O processo é finalizado quando o último vértice é contemplado.

Na figura 3.13 é apresentado um exemplo de grafo direcionado e as etapas de execução do algoritmo.

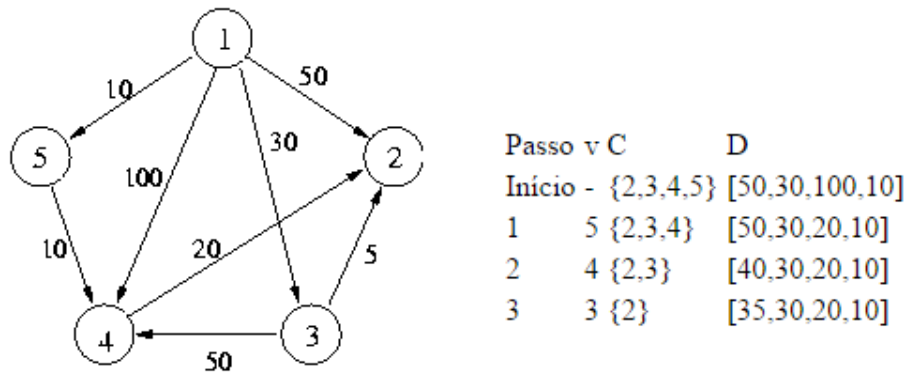


Figura 3.13: Exemplo da execução do algoritmo de Dijkstra. [20]

A partir da descrição apresentada na figura 3.12 é possível identificar o caminho mínimo. Para tal, acrescenta-se mais um vetor  $P[2..n]$ , onde  $P[v]$  indica o vértice que precede  $v$  no caminho mais curto. As alterações são simples e consistem em inicializar o vetor  $P$  e atualizá-lo no mesmo momento em que  $D$  é atualizado. As etapas de execução estão indicadas na figura 3.14.

| Passo  | v | C         | D              | P         |
|--------|---|-----------|----------------|-----------|
| Início | - | {2,3,4,5} | [50,30,100,10] | [1,1,1,1] |
| 1      | 5 | {2,3,4}   | [50,30,20,10]  | [1,1,5,1] |
| 2      | 4 | {2,3}     | [40,30,20,10]  | [4,1,5,1] |
| 3      | 3 | {2}       | [35,30,20,10]  | [3,1,5,1] |

Figura 3.14: Passos obtidos com a modificação do algoritmo de Dijkstra. [20]

No final, o estado do vetor  $P$  é  $[3,1,5,1]$ . Para saber qual é caminho mais curto entre os vértices 1 e 2, procura-se o valor na posição 2 desse vetor (sendo que  $P$  e  $D$  são indexados a partir de 2). O vetor indica que o último vértice antes do vértice 2 é o vértice 3. Repete-se o processo para ver o caminho mais curto entre 1 e 3. No vetor, a posição 3, tem-se o valor 1, que é a origem. Então, o caminho mais curto é 1,3,2. [20]

A análise do algoritmo mostra que a sua inicialização exige um tempo em  $O(n)$ . O *loop* é executado  $n - 2$  vezes e os vértices de  $Q$  são visitados a cada iteração. Se na primeira iteração, visita-se  $n - 1$  vértices, na segunda,  $n - 2$  vértices e assim

sucessivamente. Sendo a soma dos termos 1 a  $n - 1$  igual a  $n(n - 1)/2$ , deduz-se que o tempo de execução é em  $O(n^2)$ .

### 3.4 O algoritmo de Floyd e o problema do menor caminho

O algoritmo de Floyd resolve o problema do menor caminho considerando todos os pares de vértices de um grafo orientado e ponderado (grafo que apresenta pesos em suas arestas). Seus resultados são os valores de tais caminhos sem se preocupar com a listagem da sequência de arestas que foram percorridas.

Possui algumas aplicações como a verificação se um dado grafo não dirigido é bipartido e a determinação do vértice central (aquele que é acessível a partir de um vértice  $v$ , isto é, existe um caminho de  $v$  até ele).

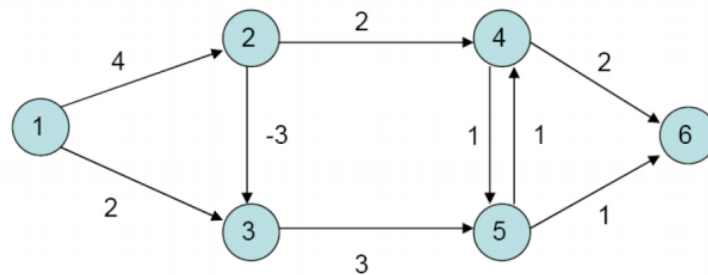


Figura 3.15: Grafo  $G$ . [42]

Uma das vantagens do algoritmo de Floyd em relação ao de Dijkstra é a possibilidade de peso negativo na aresta, conforme grafo  $G$  indicado na figura 3.15. Os seguintes critérios serão utilizados para atribuição dos valores na aplicação do algoritmo:

- Valor infinito: no caso das posições das arestas não existirem.
- Valor nulo: em todos os termos da diagonal principal.
- Pesos das arestas: correspondentes aos valores das arestas que existem. [11]

A matriz de adjacência é obtida direto do grafo, enquanto a matriz de roteamento busca descrever o caminho mínimo entre cada par de vértice, conforme indicado na figura 3.16. Na matriz de roteamento, todos os elementos de uma coluna devem ter o valor de seus índices, exceto aqueles que correspondem ao valor infinito e aqueles que recebem o valor nulo.

Em seguida, o algoritmo verifica se existem caminhos mais curtos do que os que estão nas arestas, utilizando cada vértice como sendo um intermediário. Se  $M_{n \times n}$  é a matriz de adjacência e  $R_{n \times n}$  é a matriz de roteamento, a verificação é feita por

|   |   |    |   |   |   |
|---|---|----|---|---|---|
| 0 | 4 | 2  | ∞ | ∞ | ∞ |
| ∞ | 0 | -3 | 2 | ∞ | ∞ |
| ∞ | ∞ | 0  | ∞ | 3 | ∞ |
| ∞ | ∞ | ∞  | 0 | 1 | 2 |
| ∞ | ∞ | ∞  | 1 | 0 | 1 |
| ∞ | ∞ | ∞  | ∞ | ∞ | 0 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 2 | 3 | ∞ | ∞ | ∞ |
| ∞ | 0 | 3 | 4 | ∞ | ∞ |
| ∞ | ∞ | 0 | ∞ | 5 | ∞ |
| ∞ | ∞ | ∞ | 0 | 5 | 6 |
| ∞ | ∞ | ∞ | 4 | 0 | 6 |
| ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

Figura 3.16: Matriz de adjacência  $M$  e matriz de roteamento  $R$ . [42]

meio da condição:  $m_{ik} + m_{kj} < m_{ij}$ , sendo  $k$  é vértice utilizado como intermediário,  $i$  e  $j$  são, respectivamente, os índices da linha e coluna. A verificação é feita em todos os elementos da matriz de adjacência, sendo que os vértices são utilizados como intermediários de modo ordenado, começando do vértice número 1 e prosseguindo para os vértices seguintes. Se a condição estabelecida é verdadeira, o elemento  $m_{ij}$  recebe o valor de  $m_{ik} + m_{kj}$ , assim como o elemento  $r_{ij}$  da matriz de roteamento recebe o valor de  $r_{kj}$ . Caso contrário, as matrizes de adjacência e roteamento não sofrerão alterações. [42]

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 2 | 3 | ∞ | ∞ | ∞ |
| ∞ | 0 | 3 | 4 | ∞ | ∞ |
| ∞ | ∞ | 0 | ∞ | 5 | ∞ |
| ∞ | ∞ | ∞ | 0 | 5 | 6 |
| ∞ | ∞ | ∞ | 4 | 0 | 6 |
| ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 2 | 3 | ∞ | ∞ | ∞ |
| ∞ | 0 | 3 | 4 | ∞ | ∞ |
| ∞ | ∞ | 0 | ∞ | 5 | ∞ |
| ∞ | ∞ | ∞ | 0 | 5 | 6 |
| ∞ | ∞ | ∞ | 4 | 0 | 6 |
| ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 2 | 3 | ∞ | ∞ | ∞ |
| ∞ | 0 | 3 | 4 | ∞ | ∞ |
| ∞ | ∞ | 0 | ∞ | 5 | ∞ |
| ∞ | ∞ | ∞ | 0 | 5 | 6 |
| ∞ | ∞ | ∞ | 4 | 0 | 6 |
| ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 2 | 3 | ∞ | ∞ | ∞ |
| ∞ | 0 | 3 | 4 | ∞ | ∞ |
| ∞ | ∞ | 0 | ∞ | 5 | ∞ |
| ∞ | ∞ | ∞ | 0 | 5 | 6 |
| ∞ | ∞ | ∞ | 4 | 0 | 6 |
| ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 2 | 3 | ∞ | ∞ | ∞ |
| ∞ | 0 | 3 | 4 | ∞ | ∞ |
| ∞ | ∞ | 0 | ∞ | 5 | ∞ |
| ∞ | ∞ | ∞ | 0 | 5 | 6 |
| ∞ | ∞ | ∞ | 4 | 0 | 6 |
| ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

|   |   |   |   |   |   |
|---|---|---|---|---|---|
| 0 | 2 | 3 | ∞ | ∞ | ∞ |
| ∞ | 0 | 3 | 4 | ∞ | ∞ |
| ∞ | ∞ | 0 | ∞ | 5 | ∞ |
| ∞ | ∞ | ∞ | 0 | 5 | 6 |
| ∞ | ∞ | ∞ | 4 | 0 | 6 |
| ∞ | ∞ | ∞ | ∞ | ∞ | 0 |

Figura 3.17: Matrizes de roteamento com vértices intermediários. [42]

Para calcular o caminho mínimo entre cada par de vértices de um grafo, o algoritmo faz  $n$  iterações. A cada iteração corresponde uma matriz  $n \times n$  em que os valores são modificados por meio da seguinte recorrência:

$$d_{ij}^k = \min\{d_{ik}^{k-1} + d_{kj}^{k-1}, d_{ij}^{k-1}\}$$

Nessa recorrência,  $d_{ij}^k$  é o caminho entre os vértices  $i$  e  $j$  na  $k$ -ésima matriz de iteração. O fundamento desse procedimento é verificar a cada iteração se ao incluir

um vértice intermediário  $k$  no caminho de  $i$  para  $j$  tem-se uma redução do tamanho do caminho já determinado.

Nesse algoritmo, o grafo pode conter arestas com pesos negativos, porém não pode conter ciclos negativos. Além disso, o algoritmo utiliza técnica de programação dinâmica. Nesse contexto, temos a matriz de roteamento, cujo princípio consiste em tornar um vértice  $k$  pertencente a um caminho  $d_{ij}$  se, e somente se  $d_{ik} + d_{kj} = d_{ij}$ .

A estrutura do algoritmo apresenta dois passos: [13]

- 1º passo: Os vértices são numerados de 1 a  $n$ . Define-se uma matriz  $D^0$ , cujos valores  $d_{ij}^0$  correspondem ao valor das arestas  $ij$ , caso exista a aresta no grafo. Caso contrário,  $d_{ij} = \infty$ . Os elementos da diagonal são nulos para todo  $i$ .
- 2º passo: Para cada  $k = 1 \dots n$  são determinados sucessivamente elementos da matriz  $D^k$  a partir dos elementos da matriz  $D^{k-1}$  por meio de recorrência.

Na matriz  $D^n$ , o valor do caminho mínimo de todos os pares de vértices  $ij$  do grafo estarão definidos. Nesse resultado é possível determinar o centro de um grafo, que é dado pelo vértice que apresenta a menor das distâncias máximas aos demais vértices do grafo. Na figura 3.18 segue a formalização do algoritmo de Floyd.

```

início <dados  $G = (V, E)$ ; matriz de valores  $V(G)$ ; matriz de roteamento  $R = [r_{ij}]$ ;
 $r_{ij} \leftarrow j \quad \forall i; D^0 = [d_{ij}] \leftarrow V(G)$ ;
para  $k = 1, \dots, n$  fazer [  $k$  é o vértice-base da iteração ]
    início
        para todo  $i, j = 1, \dots, n$  fazer
            se  $d_{ik} + d_{kj} < d_{ij}$  então
                início
                     $d_{ij} \leftarrow d_{ik} + d_{kj}$ ;
                     $r_{ij} \leftarrow r_{ik}$ ;
                fim;
        fim;
    fim.
    
```

Figura 3.18: O algoritmo de Floyd. [11]

Como o algoritmo de Floyd toma um vértice como base da iteração e examina todos os pares de vértices em relação a ele, sua complexidade é  $O(n^3)$ . Como há  $n$  vértices e o número de pares orientados é  $n(n - 1)$ , temos  $n^2(n - 1)$  testes e, portanto, ordem cúbica.

### 3.5 O algoritmo de Kruskal e a árvore geradora mínima

Se  $G(V, E)$  é um grafo conexo não direcionado e valorado, o algoritmo de Kruskal encontra um subconjunto  $T$  de  $E$ , tal que  $T$  forme uma árvore em que a soma dos

pesos é a menor possível. A árvore geradora mínima é aquela que liga todos os vértices do grafo usando as arestas com um custo total mínimo.

**Lema 3.5.1.** Uma aresta de peso mínimo  $e$  de um grafo estará em pelo menos uma floresta com uma ou mais arestas.

**Prova.** [34] Por contradição. Suponha que  $e$  não esteja em nenhuma floresta mínima (isto é, grafo cujas componentes conexas são árvores com a menor extensão possível em cada uma delas). Ao inserir  $e$  em uma dessas florestas, temos duas possibilidades:

- 1. Forma-se um ciclo no grafo. Assim, basta retirar qualquer outra aresta do ciclo de modo a formar uma floresta de custo menor ou igual à original.
- 2. Não se forma um ciclo no grafo. Nesse caso, a retirada de qualquer aresta da floresta vai formar uma solução de custo que não supera a original.

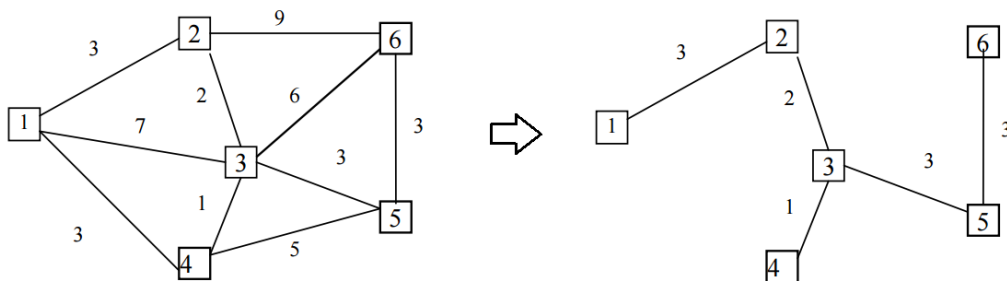
Em qualquer um dos casos, temos uma contradição. ■

**Teorema 3.5.2.** Sabe-se encontrar uma floresta  $F = (V, E)$  contendo arestas, isto é,  $|E| = a$ , cuja soma de suas arestas é mínima.

**Prova.** [34] Por indução no número de arestas  $a$ . Para  $a = 0$  o caso é trivial. Fixando o número de vértices, existe uma única floresta com nenhuma aresta e peso nulo. Suponha que seja possível determinar uma floresta de peso mínimo com  $a = k - 1$  arestas,  $1 \leq k < |V|$  (hipótese de indução). Deseja-se provar que o teorema é válido para  $a = k$ . Pelo lema 3.5.1 conclui-se a prova. ■

A prova desse teorema fornece o algoritmo porque a árvore geradora mínima é o mesmo que uma floresta mínima com  $|V| - 1$  arestas.

No algoritmo de Kruskal, a escolha da aresta é feita a cada passo, iniciando pela de menor peso. Em seguida, acrescenta-se arestas em ordem crescente de valores, obtendo-se uma árvore. O processo é finalizado na conexão da  $(n - 1)$ -ésima aresta.



**Figura 3.19:** Exemplo de aplicação algoritmo de Kruskal. [13]

Na figura 3.19, o número de vértices é 6, deseja-se uma árvore com 5 arestas. As arestas do grafo dado, colocadas em ordem crescente, constituem o conjunto  $B = \{(3,4), (3,2), (1,2), (3,5), (6,5), (1,4), (3,6), (3,1), (2,6)\}$ . Para formar o conjunto  $A$  com a sequência de arestas que foram a árvore mínima, inicia-se pela aresta de

menor peso (3,4). Passo a passo, as arestas vão sendo incluídas até obter  $n - 1$  arestas, nesse caso 5 arestas. O próximo passo é a inclusão da aresta (3,2). No passo seguinte existem duas possibilidades: (1,4) ou (1,2). Ambas apresentam o mesmo peso. As duas simultaneamente não podem ser escolhidas pois aí formaria um ciclo. Toma-se arbitrariamente a aresta (1,2). Os dois últimos passos, considerando a ordem crescente das arestas em  $B$ , consistem em acrescentar as arestas (3,5) e (5,6). Pelo algoritmo de Kruskal, a sequência de arestas (3,4),(3,2),(1,2),(3,5),(5,6) é uma árvore mínima.

Nesse algoritmo, é melhor representar o grafo por meio de um conjunto de arestas. Por se tratar de um algoritmo guloso, deseja-se visitar todas as arestas de uma vez. O algoritmo guloso é usado em problemas de otimização com o objetivo de encontrar um conjunto de candidatos que otimizam o valor de uma função objetivo. A partir do conjunto vazio, o algoritmo continua passo a passo, sempre tentando pegar o melhor pedaço, porém não se preocupa com as consequências da escolha. Na figura 3.20, o algoritmo de Kruskal está formalizado.

```

início [ dados: grafo  $G = (V,E)$  valorado nas arestas ]
para todo  $i$  de 1 a  $n$  fazer  $v(i) \leftarrow i$ ;  $t \leftarrow 0$ ;  $k \leftarrow 0$ ;  $T \leftarrow \emptyset$ ; [  $T$ : arestas da árvore ]
ordenar o conjunto de arestas em ordem não-decrescente;
enquanto  $t < n - 1$  fazer [  $t$ : contador de arestas da árvore ]
  início
     $k \leftarrow k + 1$ ; [  $k$ : contador de iterações ;  $u(k) = (i,j)$  aresta da vez ]
    se  $v(i) \neq v(j)$  então
      início
        para todo  $v(q) \mid v(q) = \max [ v(i), v(j) ]$  fazer  $v(q) = \min [ v(i), v(j) ]$ 
         $T \leftarrow T \cup (i,j)$ ; [ adiciona a aresta à árvore ]
         $t \leftarrow t + 1$ ;
      fim;
    fim;
fim.
  
```

Figura 3.20: O algoritmo de Kruskal. [11]

**Teorema 3.5.3.** Se  $G'$  é um subgrafo construído pelo algoritmo de Kruskal, então  $G'$  é uma árvore ótima de  $G$ .

**Prova:** Seja  $G'$  um subgrafo construído pelo algoritmo de Kruskal.  $G'$  é um subgrafo sem ciclos maximal de  $G$ , isto é, não pode mais ser expandido. Portanto  $G'$  é uma árvore geradora de  $G$ . Basta provar que  $G'$  é uma árvore ótima. Seja  $G'$  uma árvore geradora obtida sem peso mínimo. Assim, existe uma árvore geradora  $G''$  com peso menor que  $G'$ .

Seja  $e$  a primeira aresta escolhida para  $G'$  de modo que  $e \notin G''$ . Se  $e$  for adicionado a  $G''$  tem-se um ciclo que contém uma aresta  $e_k \notin G'$ . Retirando  $e_k$ , obtém-se uma árvore  $T$  com peso menor que  $G'$ . Assim, pelo algoritmo de Kruskal,  $e_k$  seria a aresta escolhida em lugar de  $e$ . Efetivamente, a árvore obtida seria a de menor peso. ■

A complexidade do algoritmo de Kruskal, considerando um grafo com  $n$  vértices e  $a$  arestas, leva em conta o número de operações. Para ordenar as arestas, tem-se

$O(a \cdot \log(n))$ . Para inicializar os conjuntos distintos de cada componente conexa, tem-se  $O(n)$ . No pior caso, tem-se  $O((2a + n - 1)\log(n))$  para determinar e misturar as componentes conexas. O restante das operações, obtém-se  $O(a)$ . Assim, em um grafo que apresenta  $a$  arestas, tem-se uma complexidade  $O(a \cdot \log(n))$ . [12]

### 3.6 O algoritmo de Prim e a árvore geradora de custo mínimo

O algoritmo de Prim inicia em um vértice qualquer. Em seguida, escolhe-se a aresta  $(u,v)$  de menor custo tal que  $u$  pertença à árvore e  $v$  não pertence à essa árvore. A expansão da árvore ocorre por adição de arestas e respectivos vértices sucessivamente. A cada vértice adicionado, procura-se aumentar o custo o menos possível. Esse procedimento se assemelha com o algoritmo de Dijkstra.

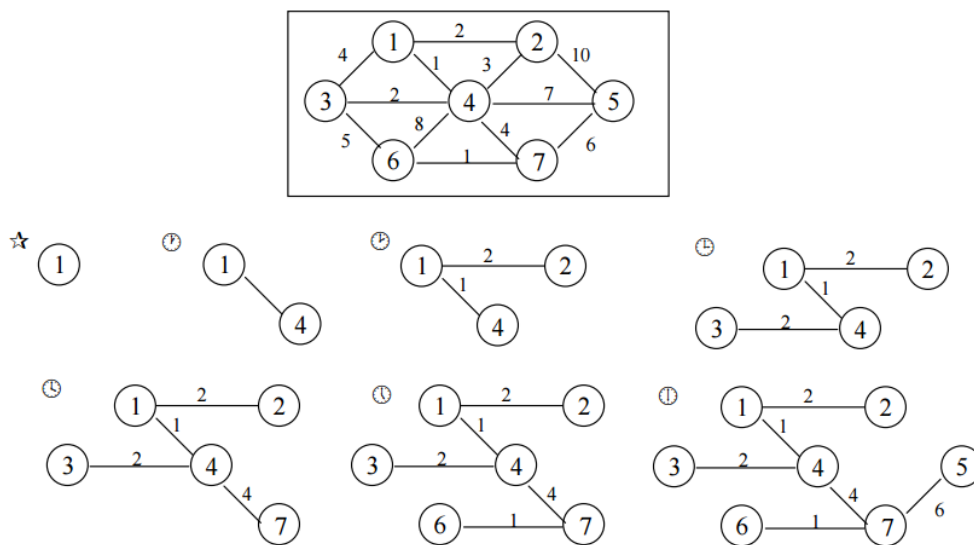


Figura 3.21: Exemplo de aplicação do algoritmo de Prim. [36]

O problema só apresenta solução se o grafo for conexo. O algoritmo faz crescer uma árvore até que ela se torne geradora. A escolha da aresta mais barata em uma determinada etapa não leva em conta o efeito global dessa escolha. Desse modo, o algoritmo de Prim tem perfil guloso. A diferença entre os algoritmos de Prim e de Kruskal está na forma como cada um escolhe a aresta em cada passo. No algoritmo de Kruskal, o conjunto das arestas tomadas é uma floresta em que as componentes conexas vão se unindo a cada iteração. No algoritmo de Prim, esse conjunto nunca deixa de ser uma árvore, já que cada aresta acrescentada une um vértice que está fora da árvore com um vértice dentro dela.

Inicialmente considere o conjunto  $B$  com um vértice arbitrário do grafo  $G$ . Em cada passo, o algoritmo seleciona a aresta de menor peso a partir de todas as arestas

que incidem em algum vértice do conjunto  $B$ . A outra extremidade da aresta escolhida passa a fazer parte de  $B$ . O procedimento encerra quando todos os vértices de  $G$  pertencerem a  $B$ .

```

início           [ dados: grafo  $G = (V,E)$  valorado nas arestas ] ; valor  $\leftarrow \infty$ ; custo  $\leftarrow 0$ ;
   $T \leftarrow \{1\}$ ;  $E(T) \leftarrow \emptyset$ ;                                $T$  e  $E(T)$ : vértices e arestas da árvore ]
  enquanto  $|T| < n - 1$  fazer
    início
      para todo  $k \in T$  fazer                                           [ examinar vértices já escolhidos ]
        início
          para todo  $i \in V - T$  fazer [ examinar vértices ainda não escolhidos ]
            se  $v_{ki} < \text{valor}$  então
              início
                valor  $\leftarrow v_{ki}$ ;  $\text{vesc} \leftarrow k$ ;  $\text{vnovo} \leftarrow i$ ;
              fim;
            fim;
          fim;
        custo  $\leftarrow \text{custo} + \text{valor}$ ;  $T \leftarrow T \cup \{\text{vnovo}\}$ ;  $E(T) \leftarrow E(T) \cup (\text{vesc}, \text{vnovo})$ ; valor  $\leftarrow \infty$ ;
      fim;
    fim.

```

**Figura 3.22:** O algoritmo de Prim. [11]

A análise da complexidade desse algoritmo necessita do conceito de *heap*. Uma *heap* é uma estrutura de dados organizada como uma árvore binária equilibrada, implementando uma fila de prioridade. Por essa razão, faz-se  $O(\log(n))$  operações. [37] Portanto, a complexidade é  $O(a \log(n))$  com uso de *heaps* e listas de adjacências de um grafo com  $n$  vértices e  $a$  arestas. No uso de matriz de adjacência, pela similaridade com o algoritmo de Dijkstra, a complexidade é  $O(n^2)$ .



## A sequência didática

---

A teoria dos grafos pode ser uma oportunidade de introdução mais significativa da Matemática Discreta no Ensino Básico. Os diferentes tipos de grafos e os problemas correlatos, bem como os algoritmos correspondentes, podem contribuir para a aprendizagem de Análise Combinatória, Probabilidades e Poliedros de Platão, assuntos previstos nos Parâmetros Curriculares Nacionais. [\[7\]](#) [\[31\]](#) [\[6\]](#)

Uma análise desses Parâmetros Curriculares Nacionais (PCN) evidencia que a teoria dos grafos não é formalmente prevista como um dos temas curriculares. Mas há competência que favorece a inclusão da teoria de grafos na Educação Básica, como por exemplo, “selecionar diferentes formas de representar um dado ou conjunto de dados e informações, reconhecendo as vantagens e limites de cada uma delas”. [\[7\]](#)

A utilização de algoritmos também está contemplada em competência como “identificar as relações envolvidas e elaborar possíveis estratégias para enfrentar uma dada situação problema”. E ainda, “interpretar, fazer uso e elaborar modelos e representações matemáticas para analisar situações”.[\[7\]](#)

A expansão do alcance da Olimpíada Brasileira de Informática (OBI) nas escolas da Educação Básica é um sinal de que há possibilidade de trabalho com linguagem e lógica de programação.

A OBI está organizada em três modalidades. [\[9\]](#)

- Modalidade Iniciação, com resolução de problemas de lógica e de computação, usando apenas lápis e papel.

Nível 1 - alunos até sétimo ano do Ensino Fundamental.

Nível 2 - alunos até nono ano do Ensino Fundamental.

- Modalidade Programação, composta de tarefas de programação com níveis variados de dificuldade. Há a presença de tarefas que envolvem algoritmos e grafos.

Nível Júnior - alunos do Ensino Fundamental.

Nível 1 - alunos até o primeiro ano do Ensino Médio.

Nível 2 - alunos até o terceiro ano do Ensino Médio.

- Modalidade Universitária com para alunos que estão cursando, pela primeira vez, o primeiro ano de um curso de graduação.

Assim como ocorre com a Olimpíada Brasileira de Matemática (OBM), há a distribuição de medalhas de ouro, prata e bronze. Uma equipe dos quatro melhores alunos da modalidade programação é formada, anualmente, para representar o país na Olimpíada Internacional de Informática.

A estratégia para resolução de problemas que envolvam grafos é similar à estratégia proposta por Lima (1998) [25] na resolução de problemas combinatórios.

1. Postura. Coloque-se no papel da pessoa que irá realizar a tarefa proposta e assim, traçar estratégias para a sua resolução.
2. Divisão. Sempre que possível, divida as decisões a serem tomadas em decisões mais simples, similar a subitens.
3. Não adiar dificuldades. Se determinada decisão a ser tomada for mais restrita que as demais, essa decisão deve ser tomada em primeiro lugar.

As competências aqui citadas, bem como a estratégia apresentada em três passos para a resolução de problemas que envolvem grafos, são referenciais para a sequência didática que será apresentada a seguir.

## 4.1 Introdução ao estudo de grafos com auxílio de algoritmos de programação

As atividades sobre grafos e os principais algoritmos para resolver problemas envolvendo grafos foi estabelecida na perspectiva de sequência didática. Segundo Zabala (1998) [46], a sequência didática é “um conjunto de atividades ordenadas, estruturadas e articuladas para a realização de certos objetivos educacionais, que têm um princípio e um fim conhecidos tanto pelos professores como pelos alunos”. A sequência de atividades é organizada e fundamentada para a realização de certos objetivos educacionais.

A sequência didática foi planejada para ser realizada em três aulas de 100 minutos. O objetivo geral consistia em introduzir o estudo de grafos por meio de problemas e algoritmos já bem estabelecidos para suas respectivas resoluções. Como objetivos específicos, esperava-se utilizar a representação de grafos como representação auxiliar de modelos que resolvessem alguns problemas combinatórios e probabilísticos. Além disso, esperava-se uma aprendizagem mais significativa de poliedros de Platão e a relação de Euler. Finalmente, a implementação de algoritmos com uso de linguagem de programação.

Desse modo, a atividade foi direcionada para o segundo ano do Ensino Médio devido à matriz curricular vigente dessa série abranger Análise Combinatória, Probabilidade e Poliedros de Platão.

### 4.1.1 Aula 1: Grafos

A aula 1, denominada Grafos, tem como objetivo apresentar a definição de grafo, suas representações, conceitos fundamentais e os tipos essenciais para a segunda seção.

A primeira parte inicia-se com um breve histórico sobre o desafio das sete pontes de Königsberg, proposto para Euler, quando este visitou a cidade em 1736. O problema inicial proposto aos alunos é o mesmo que foi proposto para Euler: existe algum passeio que atravesse cada ponte uma única vez e volte ao ponto de partida?

A partir do esquema usado por Euler, é definido grafo, apresentada sua representação gráfica e estabelecido o conjunto de vértices e o conjunto de arestas. Uma breve atividade sobre enumeração desses conjuntos finaliza essa parte.

As noções básicas de grafos inicia a segunda parte da aula, com a noção de vértices adjacentes, laço e grau de um vértice. A ideia de matriz de adjacência é apresentada no decorrer da atividade que encerra a segunda parte.

A terceira parte reforça alguns conceitos importantes como ordem e tamanho de um grafo, percurso, caminho, percursos simples e complementar, ciclo, circuito, grafo complementar e subgrafo. Uma atividade de identificação encerra a terceira parte e reforça os fundamentos para a última parte.

Na quarta parte, alguns grafos especiais, que serão essenciais para os algoritmos da segunda seção, são apresentados. O grafo completo é o primeiro, seguido de uma breve atividade de esboço do grafo completo  $K_6$ . Os grafos bipartido, orientado, conexo,  $k$ -regular, árvore e planar são definidos sucessivamente com seus respectivos exemplos, encerrando a primeira seção.

As atividades que compõem a aula foram programadas para a duração de aproximadamente 100 minutos, incluindo a discussão das ideias, perguntas e eventuais intervenções da aplicação.

### 4.1.2 Aula 2: Grafos - problemas e algoritmos

A aula 2, envolvendo problemas com grafos e seus respectivos algoritmos para solução, tem como objetivo apresentar os problemas fundamentais e os respectivos algoritmos usados para solucioná-los. Desse modo, espera-se explorar o uso de algoritmos e expandir horizontes para eventuais aplicações envolvendo linguagens programação, como já ocorreu em alguns problemas propostos pela OBI.

A aula foi dividida em três partes, buscando agrupar os algoritmos de acordo com o tipo de problema proposto. Sempre que possível, foram apresentados algoritmos diferentes para resolução de cada tipo de problema.

A primeira parte, trata dos problemas de caminho mínimo. O problema é apresentado usando grafos e, em seguida, o primeiro algoritmo é apresentado.

O algoritmo de Dijkstra é apresentado numa linguagem acessível e uma atividade é proposta. A partir de um grafo, deseja-se preencher uma tabela, obtendo-se assim, uma trajetória de menor caminho. Em seguida, as etapas são apresentadas de modo detalhado, preparando para o uso de linguagem de programação.

Em seguida, o algoritmo de Floyd é apresentado com a sugestão do uso de matrizes. Na atividade seguinte, buscou-se determinar todas as distâncias dos menores caminhos entre todos os pares de vértices do grafo dado. Assim, por meio da matriz de adjacência, das matrizes de roteamento e o roteamento com vértice intermediário, busca-se solucionar o problema.

Os problemas de interligação são tratados na segunda parte. Por meio da definição de árvore, um problema de probabilidade é proposto para o uso da árvore de decisão. A árvore de decisão é uma ferramenta que ao ser utilizada, dá ao aluno a capacidade de aprender e tomar decisões. O problema proposto é sobre três moedas lançadas ao ar com a intenção de saber a probabilidade de saírem exatamente duas vezes o mesmo lado (seja cara ou coroa).

Em seguida, apresenta-se a diferença entre problema de caminho mínimo e problema de árvore e interligação. Enquanto no primeiro procura-se saber o menor custo para se ir de um vértice ao outro, no segundo se procura chegar ao destino de modo mais econômico possível.

Apresenta-se então o problema da árvore parcial de custo mínimo com a aplicação do algoritmo de Kruskal. O procedimento, que consiste em incluir a aresta de menor valor, sem formar ciclo, até que todos os vértices estejam ligados, é usado na segunda atividade dessa parte. Em seguida, o algoritmo é apresentado na forma detalhada para a explicação de sua caracterização como do tipo guloso. Um teorema é enunciado com sua respectiva prova, com o objetivo de mostrar que o algoritmo funciona bem. Tendo em vista a demonstração do teorema, uma atividade de ilustração é proposta logo a seguir.

Finalmente, essa parte se encerra com o algoritmo de Prim e uma atividade sobre a árvore parcial de custo mínimo de um grafo dado. Para evidenciar as diferenças em relação ao algoritmo anterior, a atividade apresenta o mesmo grafo usado no algoritmo de Kruskal.

Na terceira e última parte da aula são apresentados os problemas de coloração. Explicam-se os significados de número cromático de um grafo e número cromático das arestas de um grafo. Em seguida, uma atividade contextualizada é proposta, com vistas a apresentar o teorema das quatro cores. Após enunciar o teorema e apresentar um exemplo, a mesma atividade é proposta com a utilização desse teorema.

Apesar da extensão da aula, espera-se concluir as atividades em 100 minutos, uma vez que os conceitos e propriedades essenciais foram abordadas na seção anterior. Essa aula, assim como a anterior, não necessitam do uso de computador, sendo

opcional a consulta via internet.

### 4.1.3 Aula 3: Grafos - algoritmos de programação

Na terceira aula foi usado o mesmo programa utilizado na OBI pela escola pesquisada. O programa Code::Block é um ambiente integrado de desenvolvimento de código aberto para a linguagem C/C++ com recursos voltados para programadores iniciantes. Também conta com realce de sintaxe, múltiplos compiladores, depurador personalizado, além da possibilidade de utilização de plugins que se encontram disponíveis na página do projeto Codeblocks. [1]

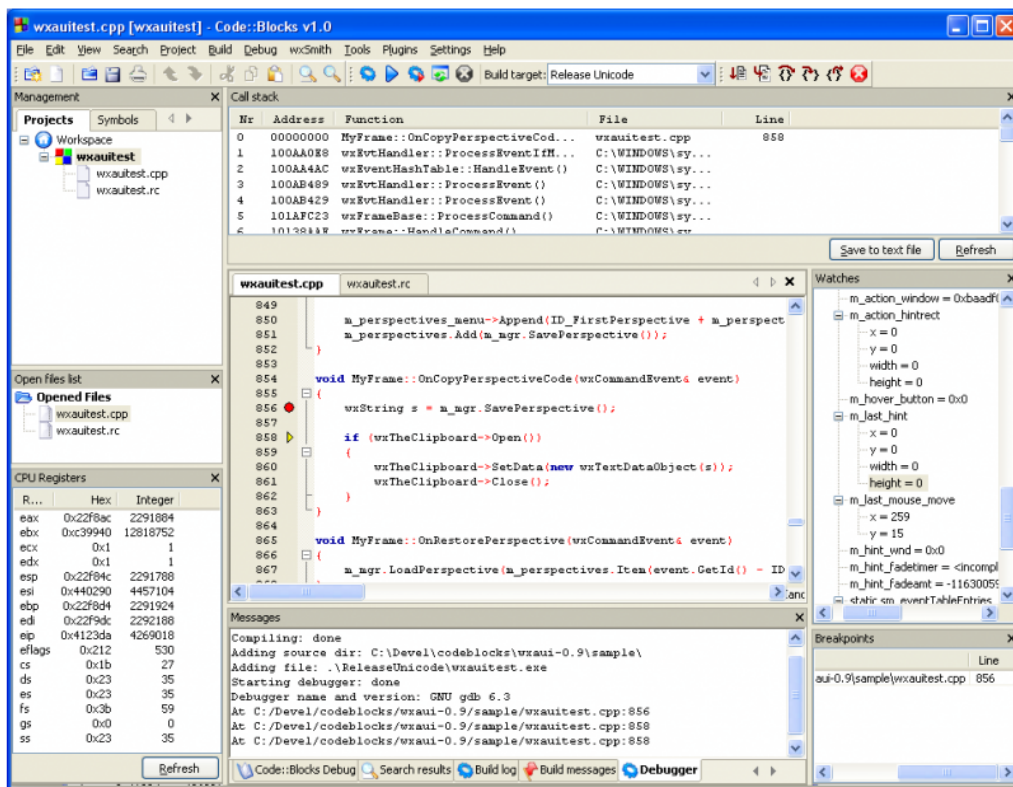


Figura 4.1: Interface do Code::Blocks [2]

A terceira aula foi adaptada do trabalho realizado por Paulo Feofiloff com algoritmos para grafos em C e disponível na página da internet desenvolvida pelo Instituto de Matemática e Estatística e o Departamento de Ciência da Computação da Universidade de São Paulo (USP). [18]

O tema das notas disponíveis na página da internet [18] é a construção de programas eficientes, capazes de resolver grandes problemas em pouco tempo. Nesse tipo de problema é dado um grafo e o objetivo é encontrar algum tipo de objeto dentro dele.

Todos os programas são escritos em linguagem C. Entretanto, os objetos tratados são mais gerais que grafos: os digrafos ou grafos dirigidos. Sendo assim, alguns pontos

mais importantes foram transcritos tendo em vista o público alvo das atividades (segundo ano do Ensino Médio) e os problemas propostos na seção 2. Evidentemente, não há a intenção de esgotar o assunto e nem a pretensão de aprofundar nas aplicações e implementações dos algoritmos.

A terceira aula se inicia com a definição de digrafo e uma atividade para especificar um digrafo por meio de seu conjunto de arestas dirigidas (ou arcos). Em seguida, lista-se uma série de procedimentos para representação de um digrafo por meio da matriz de adjacência.

A segunda parte se inicia com a implementação do algoritmo de Dijkstra, seguida de uma atividade de programação para encontrar a árvore de custo mínimo com raiz zero (vértice de origem da busca, com estimativa do custo mínimo igual a zero).

Logo em seguida, o algoritmo de Kruskal tem sua implementação indicada. Uma atividade para encontrar a árvore geradora de custo mínimo no grafo dado finaliza a atividade.

## 4.2 A aplicação da sequência didática

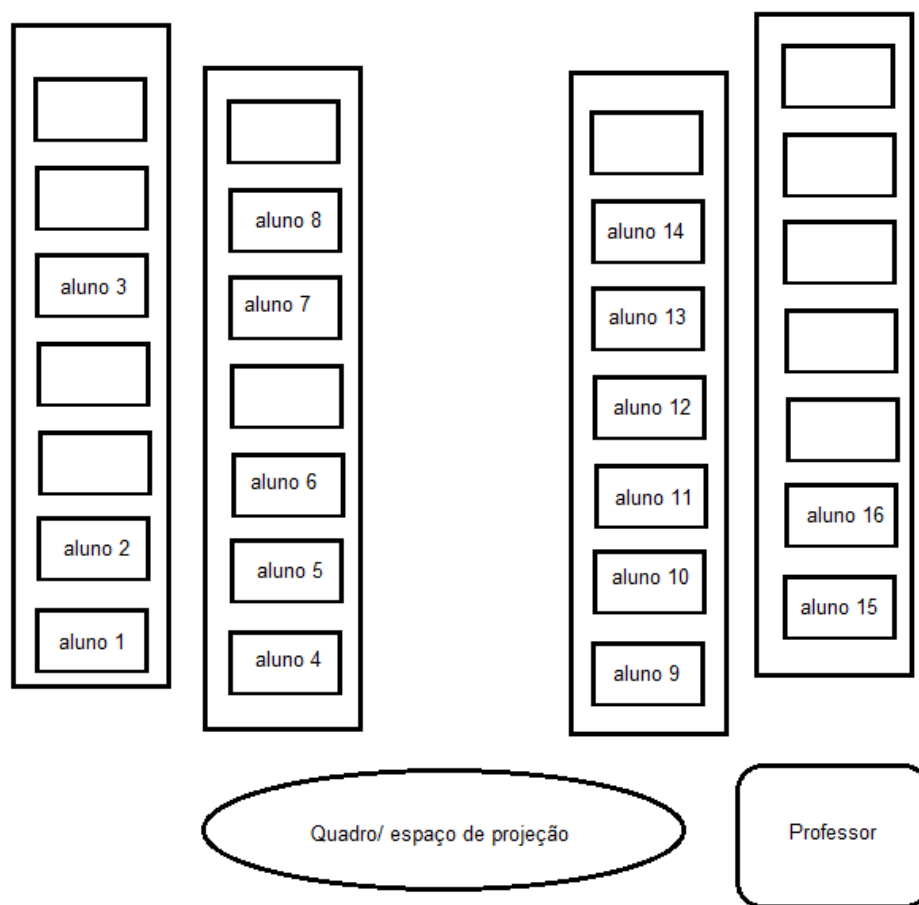
A sequência didática começou a ser aplicada após o convite extensivo à todas as turmas do segundo ano do Ensino Médio da escola. O público foi formado por 16 alunos que se voluntariaram para participar das atividades, tendo em vista o universo de 112 alunos do ano pesquisado.

Entre os alunos voluntários indicados na figura 4.2, temos dois medalhistas de Olimpíadas de Matemática (alunos 9 e 16), sendo que nenhum deles tinha conhecimento prévio do assunto, nem mesmo nas atividades do Programa de Iniciação Científica Jr (PIC). Dois outros alunos (10 e 11) haviam participado de duas Olimpíadas Brasileira de Informática (OBI) e tinham conhecimento da linguagem de programação C. Apenas mais um aluno (14) tinha noção dessa linguagem de programação, porém sem nenhuma participação registrada na OBI.

Entre os demais alunos, temos perfis diversificados, sendo cinco alunos com desempenho maior que oito (alunos 1, 3, 7, 8 e 12), quatro alunos com desempenho maior que cinco e não superior a oito (alunos 5, 6, 13 e 15), dois alunos (2 e 4) com dificuldades cognitivas e desempenho menor que cinco (numa escala de zero a dez, onde a média para aprovação na escola é cinco).

As três aulas foram realizadas no Laboratório de Informática da escola, onde cada um dos 16 alunos tinham à sua disposição um computador com sistema operacional Windows/ Linux e a multiplataforma Code::Blocks. O laboratório conta com projetor multimídia, 29 computadores (sendo 1 para o professor) e sistema de som integrado. As máquinas ficam dispostas em 7 filas conforme esquema da figura 4.2, em que está indicado a disposição de cada um dos 16 alunos.

A aplicação das aulas ocorreram durante três dias alternados, ao longo de duas semanas, no contraturno (turno da tarde), com o acompanhamento do pesquisador e



**Figura 4.2:** Esquema do laboratório de informática e a disposição dos alunos

o monitor do laboratório da instituição.

A primeira aula foi aplicada numa terça-feira, de 13h30min até 15h10min. Os alunos entraram no laboratório, ligaram suas respectivas máquinas e receberam a primeira seção do professor. Foram repassadas as instruções da sequência didática e após dez minutos, a exposição do assunto se iniciou, sempre com a participação dos alunos envolvidos. As atividades foram projetadas no quadro branco do laboratório para fins de discussão ao longo da resolução. Apesar de não ter a necessidade do uso de computador nas duas primeiras sessões, muitos alunos procuraram informações complementares na internet e alguns buscaram a fonte das figuras, a fim de coletar informações.

Na aula seguinte, ocorrida dois dias após (numa quinta-feira), no mesmo horário do primeiro dia, os mesmos alunos compareceram e realizaram as atividades na mesma dinâmica do primeiro dia. As atividades ocorreram com pequenos debates entre uma solução e outra. Pelo fato de ter sido finalizada em oitenta e cinco minutos, nos instantes finais, o professor explicou como seriam as próximas atividades, envolvendo a linguagem de programação C. Apenas três alunos, já com conhecimento prévio, manifestaram interesse imediato em participar. Entre os demais alunos, nenhum mostrou interesse em continuar ou mesmo aprender os fundamentos da linguagem de

programação.

No último dia, ocorrido na segunda-feira da semana seguinte, entre 13h30min e 15h00min, participaram apenas os alunos que já utilizaram pelo menos uma vez a linguagem de programação C, sendo que dois deles já haviam participado da OBI. Nessa atividade, o pesquisador apenas distribuiu a atividade e ficou observando o desempenho dos alunos participantes. Os alunos consultaram o site que serviu de inspiração para as atividades para eventuais complementos de códigos ou implementações. [18] O trabalho com o programa Code::Block ocorreu com boa fluidez, uma vez que o programa é utilizado pela escola na fase de programação da OBI.

### 4.3 Análise dos dados coletados

A pesquisa qualitativa, com alguns poucos dados relativos a pesquisa quantitativa, é a base da coleta e análise de dados dessa pesquisa. Segundo Garnica (2004) [22], a pesquisa qualitativa é caracterizada por

1. “ a transitoriedade de seus resultados;
2. a impossibilidade de uma hipótese a priori, cujo objetivo da pesquisa será comprovar ou refutar;
3. a não neutralidade do pesquisador que, no processo interpretativo, vale-se de suas perspectivas e filtros vivenciais prévios dos quais não consegue se desvencilhar;
4. que a constituição de suas compreensões dá-se não como resultado, mas numa trajetória em que essas mesmas compreensões e também os meios de obtê-las podem ser (re)configuradas; e
5. a impossibilidade de estabelecer regulamentações, em procedimentos sistemáticos, prévios, estáticos e generalistas”.

Tais características não tem efeito de regras, uma vez que o conceito de pesquisa qualitativa é dinâmico. Desse modo, optou-se pela observação das atividades e anotações do comportamento, recolhendo ao final de cada seção, a atividade para posterior análise qualitativa e, quando for o caso, quantitativa. Uma socialização com todos os alunos foi realizada após os três dias de atividades, proporcionando mais dados para serem analisados.




### 4.3.1 Os grafos

Na aula 1, denominada Grafos, as definições foram apresentadas, bem como características e propriedades. Os alunos participaram atentamente e as atividades foram realizadas ao longo do tempo previsto. A curiosidade em relação ao tema logo deu lugar para a associação com problemas que envolvem a ideia de grafo. O mais citado pelos alunos foi o diagrama de árvore (árvore de decisão) usado em probabilidades. Outros lembraram de matrizes e apenas um aluno associou com poliedros.

A primeira atividade <sup>1</sup>, sobre as pontes de Königsberg, a totalidade dos alunos percebeu que não é possível encontrar uma solução sem acrescentar pelo menos uma ponte. Alguns realizaram a tarefa por tentativa e erro até perceber a insolubilidade do problema. O aluno 6 realizou a atividade por meio da consulta da fonte da figura e transcreveu as informações para sua folha de respostas, conforme indicado na figura 4.3. Não foi percebido prejuízo na aprendizagem devido às fontes disponíveis abaixo das figuras. A busca pelos links se mostrou como mais uma fonte de consulta.

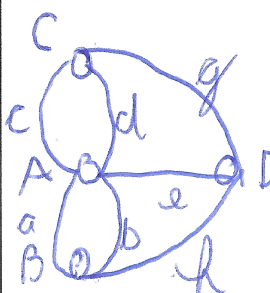
A ilustração a seguir mostra a situação que estava sendo proposta e um esquema que é uma representação gráfica do que hoje em dia chamamos de grafo.



Disponível em: <https://goo.gl/ZZLm9> (acesso em 5/10/16).

**ATIVIDADE 1.1.1:** Existe algum passeio que atravesse cada ponte uma única vez e volte ao ponto de partida?

$V = \{m_1, m_2, p\}$   $\exists$  uma ilha ou margem  
 $A = \{m_1, m_2, p\}$   $\exists$  existe uma ponte unindo as margens ou ilha  $m_1$  e  $m_2$



[1] BOAVENTURA NETTO, Paulo Oswaldo & JURKIEWICZ, Samuel. *Grafos: introdução e prática*. São Paulo: Blucher, 2009.

Para toda  $v \in V$ ,  $\deg(v)$  é ímpar, ou seja, não é um grafo de Euler

**Não tem solução**

Figura 4.3: Atividade 1.1.1 apresentada pelo aluno 6

A totalidade dos alunos também encontrou facilmente as respostas para a atividade 1.1.2. Porém a atividade 1.2.1 teve aproveitamento médio de 83%, sendo o item que

<sup>1</sup>As atividades se encontram no apêndice.

trata da matriz de adjacência o item com maior número de pequenos equívocos. Isso decorre de outros pequenos equívocos que aconteceram de itens anteriores e que após pequenos esclarecimentos, foram amplamente compreendidos e sinalizados durante a atividade.

**ATIVIDADE 1.2.1:** Observe o grafo a seguir.

Disponível em: <https://goo.gl/gzv7EK>. Acesso em 05/10/16.

a) Determine o grau de cada vértice.

$d(v_1) = 3$        $d(v_4) = 1$   
 $d(v_2) = 4 \times 5$        $d(v_5) = 0$   
 $d(v_3) = 3$

d) Podemos tomar os rótulos dos vértices e associá-los às linhas e colunas de uma matriz quadrada. Quando um vértice é adjacente a outro, a posição  $a_{ij}$  deve ser igual a 1, indicando que os vértices  $i$  e  $j$  estão associados por meio de uma aresta. Caso contrário, deve ser igual a 0. Essa matriz se denomina **matriz de adjacência**. Determine a matriz de adjacência do grafo acima.

$a_{11} = 0$   
 $a_{12} = 1$   
 $a_{13} = 1$   
 $a_{14} = 0$   
 $a_{15} = 0$

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

Figura 4.4: Atividade 1.2.1 apresentada pelo aluno 1

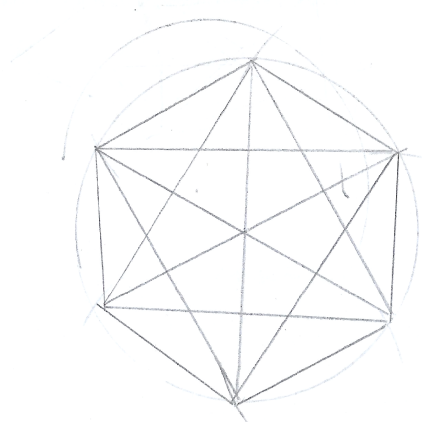
Na figura 4.4, percebe-se um caso em que o laço não foi compreendido adequadamente. Entretanto, a concepção de matriz de adjacência não fica comprometida e o aluno 1 determina a matriz esperada.

A atividade 1.3.1 ocorreu sem alterações, com pequenas dúvidas apresentadas por alguns alunos e esclarecidas por colegas próximos, sem a necessidade de intervenção do pesquisador.

Na parte sobre grafos especiais, a atividade 1.4.1 propunha um esboço de um grafo completo  $K_6$ . O aluno 3 percebeu que, assim como o  $K_5$  tinha como esboço um pentágono com todas as diagonais traçadas, o  $K_6$  poderia ser algo similar. A partir desse raciocínio, usando compasso e régua, o aluno traçou um hexágono regular inscrito numa circunferência e, em seguida, traçou todas as suas diagonais a fim de obter o grafo solicitado, conforme indicado na figura 4.5.

À medida que os demais tipos grafos foram sendo mostrados, alguns alunos procuravam verificar as características contidas nos respectivos exemplos, encerrando a atividade com uma série de perguntas sobre as aplicações desse assunto, em especial,

**ATIVIDADE 1.4.1:** Esboce um grafo completo  $K_6$ .



**Figura 4.5:** Atividade 1.4.1 apresentada pelo aluno 3

na Ciência da Computação.

### 4.3.2 Grafos: problemas e algoritmos

Com os fundamentos já trabalhados na aula anterior, os alunos foram apresentados aos algoritmos que resolvem problemas envolvendo grafos. Cada tipo de problema é apresentado, seguido de seu algoritmo de resolução e uma atividade de aplicação.

O primeira atividade abordava o algoritmo de Dijkstra para a solução de problema de caminho mínimo. A princípio, houve certo estranhamento dos alunos que não participaram das Olimpíadas de Matemática ou Informática. Após uma breve intervenção, algumas soluções foram apresentadas, conforme a apresentada pelo aluno 4, indicada na figura 4.6.

O aluno 4, usando uma notação própria, procura resolver o problema apresentando, passo a passo, o procedimento do algoritmo de Dijkstra. Apesar da dificuldade em Matemática relatada pelo próprio aluno, ele mesmo disse sentir-se confiante com esses tipos de problemas, que em sua percepção, não são convencionais.

A atividade seguinte, sobre o algoritmo de Floyd, apresentou uma dificuldade crescente, à medida que as matrizes de roteamento consideravam determinado vértice intermediário. Por se tratar de um problema de caminho mínimo, onde se determina todas as distâncias dos menores caminhos entre todos os pares de vértices do grafo dado, alguns alunos apresentaram pequenos equívocos. A intervenção foi mais incisiva, a ponto de ser proposto que todos discutissem juntos a solução do problema.

Na figura 4.7, o aluno 15 constrói corretamente a matriz de roteamento considerando o vértice intermediário 1. Ao passar para os vértices intermediários 2 e 3, as dificuldades surgem na montagem da matriz, resultando em pequenos equívocos, que foram amplamente debatidos durante a atividade. As discussões se iniciaram entre colegas próximos e logo tomou alcance de toda a turma, sendo necessário mediar as

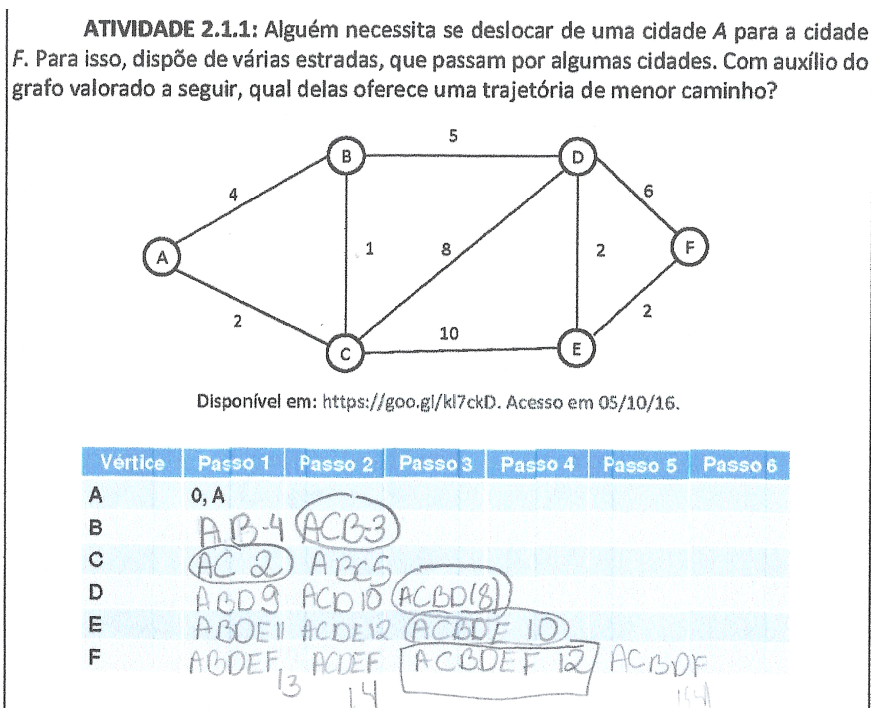


Figura 4.6: Atividade 2.1.1 apresentada pelo aluno 4

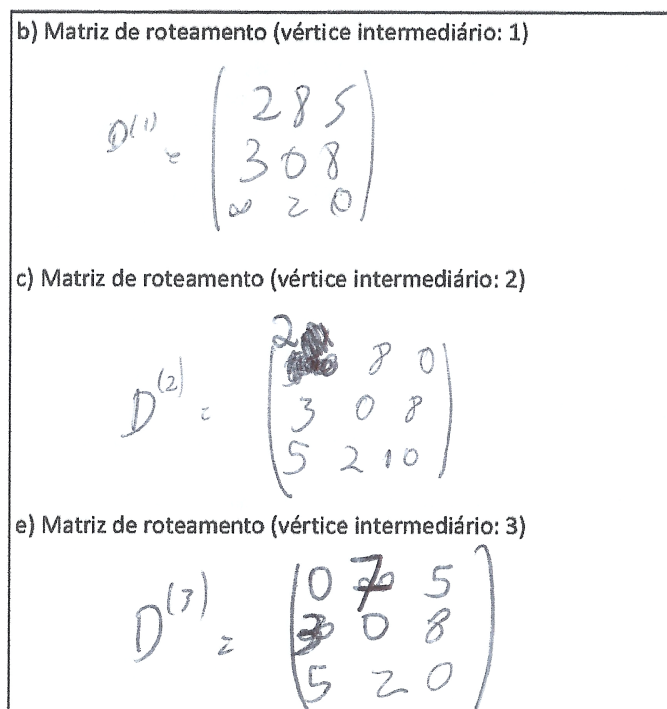


Figura 4.7: Atividade 2.1.2 apresentada pelo aluno 15

dúvidas de valores que foram surgindo.

A segunda parte envolvia os problemas de interligação. Inicialmente, foi abordada a árvore de decisão, utilizada para cálculos de probabilidades. Por ser uma atividade com assunto amplamente trabalhado poucos meses antes, 91% dos alunos observados conseguiram fazer a atividade usando o diagrama de árvore proposto. Apenas um

aluno respondeu por meio de lista, sendo que os demais responderam conforme indicado na figura 4.8.

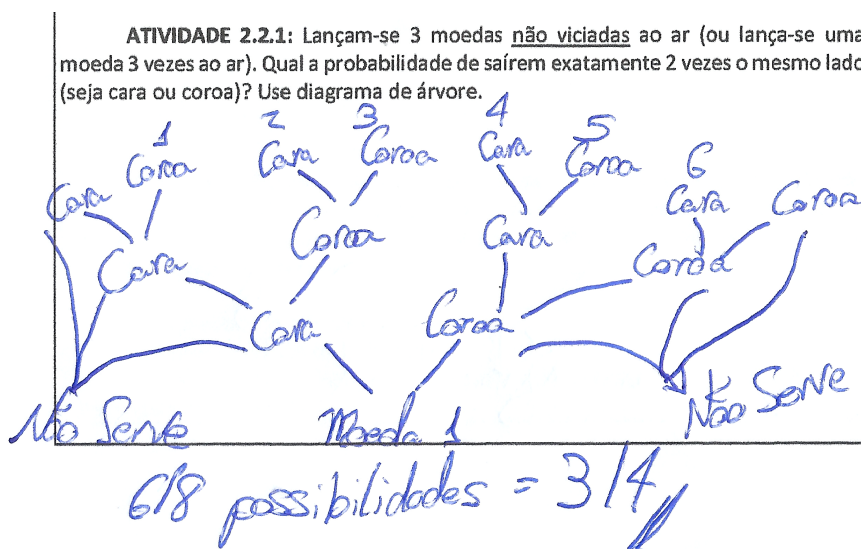


Figura 4.8: Atividade 2.2.1 apresentada pelo aluno 13

O algoritmo de Kruskal foi abordado logo em seguida e grande parte da dificuldade observada ocorreu devido ao não cumprimento do procedimento correto. Nesse algoritmo, procura-se incluir a aresta de menor valor, sem formar ciclos, até que todos os vértices estejam ligados. Dos dezesseis participantes da atividade, cinco formaram ciclos e quatro não contemplaram todos os vértices. Sete alunos resolveram corretamente o problema, conforme indicado na figura 4.9.

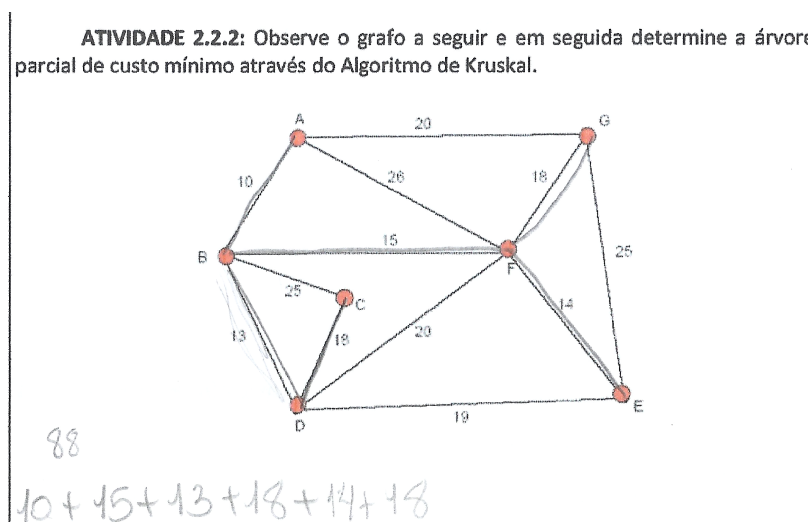


Figura 4.9: Atividade 2.2.2 apresentada pelo aluno 9

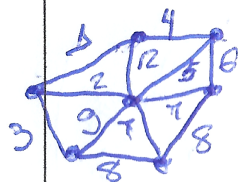
Após a demonstração do teorema 2.2.3 sobre esse algoritmo, a atividade 2.2.4 propõe uma ilustração para a demonstração do teorema, tendo-se destacado o aluno 13 com seus três desenhos indicados na figura 4.10.

A última atividade dessa parte retoma o mesmo problema do algoritmo de Kruskal, só que dessa vez com a perspectiva de utilização do algoritmo de Prim. O

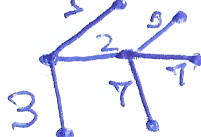
**TEOREMA 2.2.3:** O algoritmo de Kruskal fornece uma solução ótima (solução possível que otimiza a função objetivo) para o problema da interligação a custo mínimo.

**Demonstração:** Suponha que  $T$  não tenha menor peso. Nesse caso existe uma árvore  $T'$  tal que seu peso é menor que o peso de  $T$ . Seja  $e$  a primeira aresta para  $T$  de modo que ela não pertença a  $T'$ . Adicionando  $e$  a  $T'$  obtemos um ciclo que contém uma aresta  $f$  que não está em  $T$ . Retirando  $f$  temos uma árvore  $T''$  com peso menor que  $T$ . Nessa situação a aresta  $f$  teria sido escolhida no lugar de  $e$ . Logo, o algoritmo constrói efetivamente uma árvore de menor peso.

Desenho 1



Desenho 2



Desenho 3

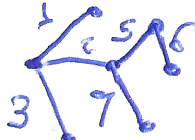


Figura 4.10: Atividade 2.2.4 apresentada pelo aluno 13

uso do mesmo problema provocou comparação imediata entre ambos os algoritmos. A maioria considerou o algoritmo de Prim mais simples de lidar. Isso pode ser observado pela quantidade de acertos, uma vez que o algoritmo de Prim teve 30% mais acertos que o de Kruskal.

Na terceira parte, foram tratados os problemas de coloração. Um problema contextualizado inicia a atividade e o grau de dificuldade foi considerado elevado por uma parte dos alunos. Ao serem questionados o motivo, apresentaram como justificativa a resolução por tentativa e erro. Entretanto, alguns alunos, como, por exemplo, o aluno 13, conseguiram fazer com certa agilidade esse mesmo problema (figura 4.11). O problema foi um facilitador para o teorema das 4 cores, que foi apresentado logo a seguir.

Após o teorema ser enunciado, grande parte dos alunos conseguiu resolver o mesmo problema com auxílio de grafos. Usando o teorema das quatro cores, o aluno 13 resolveu conforme indicado na figura 4.12. Apesar de um pequeno equívoco na solução apresentada, o aluno 13 logo reconheceu o problema e identificou que o vértice  $C_7$  deveria ser preto.

No final do segundo dia, com uma pequena sobra de tempo, foi apresentada para os alunos a parte inicial da terceira atividade. Apenas três alunos, com conhecimentos prévios em linguagem de programação C, manifestaram o desejo de participar.

**ATIVIDADE 2.3.1:** Uma companhia industrial deseja armazenar sete diferentes produtos farmacêuticos  $C_1, C_2, \dots, C_7$ , mas alguns não podem ser armazenados juntos por motivos de segurança. A tabela a seguir mostra os produtos que não podem estar no mesmo local. Encontre o número mínimo de localizações necessárias para colocar estes produtos e escreva uma combinação possível dos medicamentos compatíveis.

|    | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|----|----|----|----|----|----|----|----|
| C1 |    | X  |    |    |    | X  | X  |
| C2 | X  |    | X  | X  |    |    |    |
| C3 |    | X  |    | X  | X  |    |    |
| C4 |    | X  | X  |    | X  | X  |    |
| C5 |    |    | X  | X  |    | X  | X  |
| C6 | X  |    |    | X  | X  |    | X  |
| C7 | X  |    |    |    | X  | X  |    |

Disponível em: <https://goo.gl/5sobLQ>. Acesso em 05/10/16.

*1ª filtragem:*  
 $C_1$   
 $* C_3$   
 $** C_4$   
 $C_5$

*2ª filtragem:*  
 $C_1$   
 $C_5$

$C_2$   
 $C_6$   
 $C_7$

$C_3$   $C_4$

$C_1$   $C_2$   $C_3$   $C_4$   
 $C_5$   $C_6$   $C_7$

*4 caixas //*

Figura 4.11: Atividade 2.3.1 apresentada pelo aluno 13

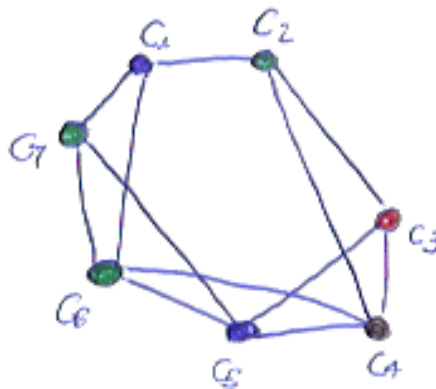


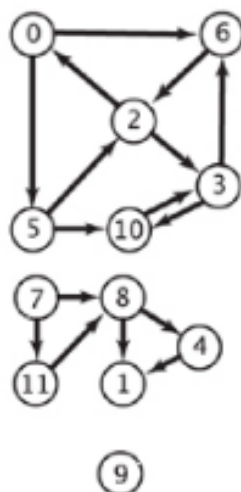
Figura 4.12: Atividade 2.3.3 apresentada pelo aluno 13

### 4.3.3 Grafos: algoritmos de programação

A última aula contou com a participação de apenas três alunos. Com o conhecimento prévio da linguagem utilizada e pleno domínio do Code::Blocks, a atividade não teve nenhum tipo de interferência. Os três alunos permaneceram concentrados durante atividade. Todos consultaram o site de autoria do professor Paulo Feofiloff em busca de mais recursos ou facilitadores para implementação. [18]

A primeira atividade do dia, sobre digrafo, foi registrada no computador, a pedido dos alunos. Os três alunos especificaram corretamente o que a atividade solicitava. Um aluno percebeu que, exibindo seu conjunto de arestas dirigidas, o digrafo ficava bem especificado, conforme exemplo da figura 4.13. Cada aresta dirigida é um par ordenado de vértices.

A atividade apresentou um conjunto de ferramentas fundamentais para a construção e manipulação de digrafos. Dois dos alunos testaram algumas dessas ferramentas,



0-5 0-6 2-0 2-3 3-6 3-10 4-1 5-2 5-10  
6-2 7-8 7-11 8-1 8-4 10-3 11-8

**Figura 4.13:** Atividade 3.1.1: conjunto de arestas dirigidas de um digrafo. Padrão de resposta dos alunos abaixo da figura dada. [18]

mas logo partiram para a atividade seguinte, sobre a aplicação do algoritmo de Dijkstra para a determinação de uma árvore de custo mínimo a partir do vértice 0. Essa atividade demandou mais tempo. Algumas dúvidas sobre o que é franja (isto é, o conjunto de todas as arestas do grafo dado que tem uma ponta inicial na árvore e a ponta final fora da árvore) e sobre a utilização de vetores logo foram esclarecidas com o suporte da página do professor Feofiloff [18]. Percebeu-se a partir dessa atividade, um certo automatismo a partir das ferramentas dadas e pequenas dúvidas sobre a compilação e posterior execução. Pequenos erros de sintaxe promoveram, em determinado momento, uma pequena troca de ideias para eventuais ajustes, a fim de verificar a execução, indicada na figura 4.14.

```
0-2 0-3 0-4 2-4 3-4 3-5 4-1 4-5 5-1 1-2
.7 .2 .4 .1 .1 .3 .4 .1 .2 .0

pai      frj      dist
0 1 2 3 4 5  0 1 2 3 4 5  0 1 2 3 4 5

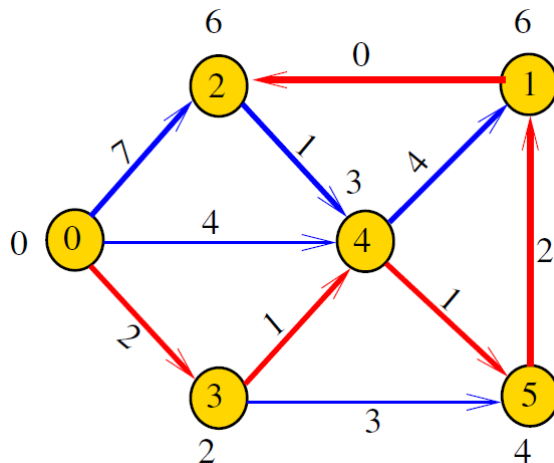
0 . . . . .  . . 0 0 0 .  .0 - .7 .2 .4 -
0 . . 0 . .  . . 0 0 3 3  .0 - .7 .2 .3 .5
0 . . 0 3 .  . 4 0 0 3 4  .0 .7 .7 .2 .3 .4
0 . . 0 3 4  . 5 0 0 3 4  .0 .6 .7 .2 .3 .4
0 5 . 0 3 4  . 5 1 0 3 4  .0 .6 .6 .2 .3 .4
0 5 1 0 3 4  . 5 1 0 3 4  .0 .6 .6 .2 .3 .4
```

**Figura 4.14:** Atividade 3.2.1: Padrão de resposta dos alunos na execução do programa. [18]

Na figura 4.14, *frj* se refere a franja da árvore com raiz no vértice 0. Os registros de vetores em *pai* contém a árvore em questão, sua franja e a distância do vértice 0



a cada vértice da árvore, além dos pesos de cada vértice externo na iteração dada. Finalmente, *dist* contém as distâncias a partir do vértice 0 até cada vértice da árvore, sendo que a última linha fornece a árvore de caminho mínimo. A figura 4.15 mostra a árvore de custo mínimo da atividade 3.2.1.



**Figura 4.15:** Atividade 3.2.1: árvore de caminho mínimo obtida pelo algoritmo de Dijkstra. [18]

A última atividade, sobre o algoritmo de Kruskal, necessitou de uma consulta mais apurada na página do professor Feofiloff [18]. Alguns termos utilizados no enunciado não estavam evidenciados ao longo das ferramentas que estavam na folha impressa. Pelo fato dos alunos não terem concluído a tempo, foi proposto aos alunos a realização da atividade para posterior entrega.

#### 4.3.4 Aula extra: socialização das impressões sobre a sequência didática

Duas semanas após a terceira aula, todos os alunos participantes foram convidados para um encontro extra, em que seriam compartilhadas as impressões sobre a experiência vivenciada. As respostas foram transcritas pelo pesquisador à medida que as opiniões foram sendo manifestadas. Os itens da pauta do debate foram:

- a relevância do assunto grafos no currículo;
- a experiência com algoritmos para resolução de problemas que envolvem grafos;
- as dificuldades advindas das resoluções das atividades propostas;
- as interações com outros assuntos da Matemática;
- linguagem de programação e algoritmos para grafos.

Sobre o primeiro ponto, a maioria dos alunos se mostrou surpresa com a existência de um assunto novo e que pode ajudar na compreensão de outros tópicos da Matemática no segundo ano do Ensino Médio. O aluno 4 se mostrou muito entusiasmado com a possibilidade de uma nova abordagem em Análise Combinatória e Probabilidades, assuntos que ele apresentou, segundo autoexame, dificuldades ao longo dos meses iniciais do ano em que ocorreu a pesquisa. O aluno 13 questionou a organização curricular e apontou que grafos teria um significado maior do que Binômio de Newton, por exemplo. O aluno 9 apresentou grafos como uma possibilidade de aplicar matrizes em outra parte da Matemática. Até então, segundo seu argumento, não havia percebido importância no assunto de matrizes, mesmo sendo medalhista de Olimpíadas de Matemática.

A respeito dos algoritmos, o aluno 16 mostrou surpresa com a lógica e raciocínio articulados em cada um dos problemas apresentados na aula 2.

As dificuldades nas atividades foram apontadas pelos alunos 2, 4, 6 e 13. Eles alegaram que, a princípio, a novidade do assunto e a aplicação de algoritmos foram complexas. Uma demanda de tempo maior foi detectada pelos demais alunos no início da aula 2. No entanto, todos chegaram ao consenso que as atividades se encerraram nesse dia com considerável fluidez, dentro da perspectiva rítmica de cada um.

O aluno 13 comentou sobre a ligação que ele percebeu entre planificação de poliedros e grafos planares, sendo grande facilitador para a compreensão do relação de Euler. Segundo seu argumento, nem sempre é fácil visualizar a quantidade de faces, arestas e vértices do dodecaedro e comparar com o icosaedro. O aluno 1 concordou e completou que a árvore de decisão ajuda a resolver problemas de Probabilidades.

Questionados sobre a ausência em relação à linguagem de programação, os alunos 6 e 7 se manifestaram com argumentos similares. A especificidade da atividade (pelo fato de ser da área de computação) foi a razão. Um gesto de concordância foi quase geral entre os presentes. Os alunos que participaram da OBI (10 e 11) afirmaram que, mesmo se copiassem e colassem as ferramentas indicadas, a compilação e a execução requerem certo conhecimento da linguagem. O aluno 10 completou que, para participar da OBI, existe um prévio treinamento com o objetivo de potencializar o desempenho dos participantes.

Dos dezesseis alunos participantes, apenas um discorda da inclusão do assunto grafos no currículo, alegando possível inchaço. Os demais concordaram entre si sobre a importância que grafos teria no aprendizado de Matemática durante o Ensino Médio.

## Considerações finais

---

A contemporaneidade das aplicações de grafos e a possibilidade de interação com outras áreas da Matemática foram os argumentos para realização da pesquisa.

A sequência didática, formato escolhido para desenvolvimento das aulas, mostrou-se um meio eficiente para reunir toda a complexidade da prática e da intervenção reflexiva (planejamento, aplicação e avaliação).

As três partes que compõem a sequência didática apresentaram uma coerência que foi percebida pela execução das atividades, participação dos alunos e as respostas apresentadas e brevemente analisadas. Apesar da terceira aula ter a participação de apenas três alunos, percebeu-se que as duas primeiras aulas foram voltadas para o Ensino Médio, especialmente segundo ano. A atividade de probabilidade mostrou que os alunos que apresentam dificuldades numa abordagem mais tradicional, na abordagem diferenciada, com o uso de grafos, apresentaram resultados significativos na aprendizagem.

A primeira aula apresentou uma introdução à teoria de grafos e foi essencial para o desenvolvimento das partes seguintes. As atividades e o desenvolvimento teórico proporcionaram reflexões sobre o uso informal de grafos em diversos problemas ao longo da vida escolar. Há uma clara indicação da possibilidade do uso de grafos com suporte a outros temas como Análise Combinatória (princípio fundamental da contagem), Probabilidades (árvore de decisão), Poliedros (planificação e grafos planares), matrizes (como ferramenta de representação) entre outros. Portanto, existem benefícios que levam a considerar a possibilidade de incluir grafos no currículo do Ensino Médio.

A segunda aula, sobre algoritmos, mostrou que há possibilidade de trabalho com algoritmos sem a necessidade de se manter na reprodução tecnicista de procedimentos pré-estabelecidos. As pequenas discussões entre um problema e outro, além das trocas de ideias sobre as estratégias utilizadas, reforçam a oportunidade que existe no estudo dos algoritmos. Entender e aplicar corretamente um algoritmo proporciona a identificação de um amplo leque de situações problemas a serem modeladas.

Contudo, a terceira aula mostrou-se voltada para um público mais específico, com desejável conhecimento prévio em noções de linguagem de programação. A aplicação da Olimpíada Brasileira de Informática (OBI), mesmo que em algumas

escolas do ensino básico, atestam a possibilidade do uso de linguagem de programação na implementação de algoritmos que resolvam problemas (no caso dessa pesquisa, grafos). A escolha da linguagem C e o uso de Code::Blocks se revelaram eficientes no contexto da escola observada, uma vez que ambos já eram usados na OBI.

A implementação dos algoritmos para grafos em C (ou outra linguagem de programação conveniente) pode ser uma realidade para alunos do Ensino Médio. Seja como uma introdução à linguagem de programação ou aprofundamento. O desenvolvimento do raciocínio lógico é um dos benefícios verificados por meio dos alunos que participam da OBI.

A breve abordagem histórica apresentada no capítulo 2 mostra que a teoria de grafos pode ser utilizada na Química Orgânica para enumerar todos os isômeros dos hidrocarbonetos alifáticos. Na Física, circuitos elétricos podem ser investigados via grafos. Percebem-se algumas significativas oportunidades para abordagens interdisciplinares envolvendo grafos na Matemática com outras áreas de conhecimento.

Algumas questões foram apontadas durante as observações e análise de dados. Mesmo não fazendo parte do objeto central da pesquisa, mostram-se como potenciais recortes para futuras eventuais análises:

- Qual a contribuição da teoria dos grafos para a Matemática Discreta ao longo da história?
- A teoria dos grafos pode ser inserida em que parte do currículo do Ensino Fundamental? (Sem a necessidade de abordar algum tipo de linguagem de programação).
- Como utilizar a linguagem de provas da teoria dos grafos para ampliar os fundamentos utilizados nas provas de teoremas?

Os dados coletados e a socialização validaram a estrutura das atividades que foram utilizadas na observação de campo. Uma evidência é a demonstração associada ao algoritmo de Kruskal, que foi compreendida, sem maiores problemas, devido à linguagem mais acessível ao estudante do Ensino Médio.

Enfim, a possibilidade de inclusão da teoria dos grafos traz uma nova abordagem de certos assuntos correlatos. O uso de linguagem de programação não é um fator determinante, mas é potencializador da aprendizagem. A diversidade verificada no cenário educacional nacional é a oportunidade de estabelecer novas interações, e, nesse sentido, grafos se destacam.

## Apêndice - Sequência didática

---

A sequência didática das páginas seguintes está dividida em três aulas.

- Aula 1: breve histórico, definições e conceitos relacionados a grafos. Atividades exploram as características e propriedades mais relevantes.
- Aula 2: explora quatro algoritmos sobre grafos, relacionados a caminho mínimo e árvore geradora mínima. Apresenta atividades complementares sobre árvore de decisão e o teorema das quatro cores.
- Aula 3: atividades relacionadas aos algoritmos de Dijkstra e Kruskal utilizando linguagem de programação C.

Caso seja necessário o acesso ao arquivo numa extensão de fácil edição, solicite por email: [darcio.nogueirajr@gmail.com](mailto:darcio.nogueirajr@gmail.com)



**Introdução ao estudo de Grafos com auxílio  
de algoritmos de programação.**  
**Público alvo: Ensino Médio – alunos do segundo ano.**  
Professor (mestrando): Dárcio Costa Nogueira Júnior  
Professor (orientador): Luís Felipe Gonçalves Fonseca  
**Aula 1: GRAFOS.**



### 1.1. As sete pontes de Königsberg.

Os primeiros registros sobre a teoria de Grafos remontam ao ano de 1736, quando o notável matemático e geômetra Leonhard Euler visitou a cidade de Königsberg. A cidade era conhecida pela elite intelectual e pelo intenso comércio. Ao chegar na cidade, Euler tomou conhecimento de um problema que estava sendo discutido por essa elite:

*“No Pregel, rio que corta a cidade, havia duas ilhas que, na época, eram ligadas entre si por uma ponte. As duas ilhas se ligavam ainda às margens por mais seis pontes ao todo. O problema consistia em encontrar o percurso para um passeio que partisse de uma das margens e, atravessando uma única vez cada uma das sete pontes, retornasse à margem de partida”. [1]*

A ilustração a seguir mostra a situação que estava sendo proposta e um esquema que é uma representação gráfica do que hoje em dia chamamos de grafo.



Disponível em: <https://goo.gl/ZZLrn9> (acesso em 5/10/16).

**ATIVIDADE 1.1.1:** Existe algum passeio que atravesse cada ponte uma única vez e volte ao ponto de partida?

Resposta esperada:

Perceber que o problema não tem solução e que seria necessário acrescentar pelo menos uma ponte.

[1] BOAVENTURA NETTO, Paulo Oswaldo & JURKIEWICZ, Samuel. *Grafos: introdução e prática*. São Paulo: Blucher, 2009.

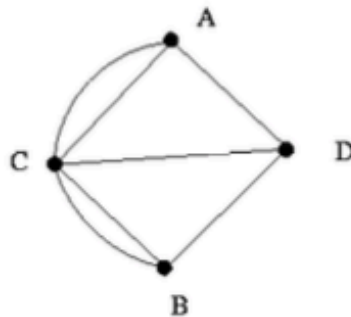
Euler provou que para o passeio proposto no problema de Königsberg ser possível, era necessário que cada massa de terra fosse ligada à outra por um número par de pontes.

A partir do esquema usado por Euler, é possível definir **grafo** como um objeto matemático formado por dois conjuntos:

1ª)  $V$  é o conjunto dos vértices, que na representação gráfica, são os pontos.

2ª)  $E$  é o conjunto de arestas ou conjunto de relações entre os vértices, que na representação gráfica, são as ligações entre dois pontos.

Vamos denominar grafos como  $G = (V, E)$  e se uma aresta une dois vértices  $v$  e  $w$ , representaremos essa aresta por  $(v,w)$  ou  $vw$ .



**ATIVIDADE 1.2.1:** Dê o que se pede.

- Conjunto  $V$

Resposta esperada:  $V = \{A, B, C, D\}$

- Conjunto  $E$

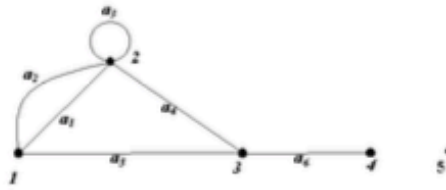
Resposta esperada:  $E = \{AC, CA, BC, CB, AD, BD, CD\}$

## 1.2. Noções de grafos.

Como representar um grafo? Usamos um esquema gráfico onde os vértices (quem em geral são rotulados) podem ser ligados a outros vértices. Quando um vértice é ligado a outro, dizemos que eles são adjacentes. Se um vértice é ligado a ele mesmo, temos um laço.

O grau de um vértice é dado pela quantidade de arestas que sai dele (ou chega a ele). Em outras palavras, o grau de um vértice é o número  $d(v)$  de vizinhos que ele possui. Se todos os vértices de um grafo não orientado (aquele em que a ordem dos vértices importa para determinar o sentido da aresta) tiverem o mesmo grau  $k$ , ele é dito  $k$ -regular

**ATIVIDADE 1.2.1:** Observe o grafo a seguir.



Disponível em: <https://goo.gl/gzv7EK>. Acesso em 05/10/16.

a) Determine o grau de cada vértice.

Resposta esperada:

$$D(V1) = 3$$

$$D(V2) = 5$$

$$D(V3) = 3$$

$$D(V4) = 1$$

$$D(V5) = 0$$

b) Existe laço no esquema? Em caso afirmativo, identifique.

Resposta esperada: Sim. Aresta  $a_3$  é um laço.

c) Faça uma lista com os vértices adjacentes de cada vértice.

Resposta esperada:

Vértice:            Vértices adjacentes:

1                    2,3

2                    1, 2, 3

3                    1, 2, 4

4                    3

5                    não há

d) Podemos tomar os rótulos dos vértices e associá-los às linhas e colunas de uma matriz quadrada. Quando um vértice é adjacente a outro, a posição  $a_{ij}$  deve ser igual a 1, indicando que os vértices  $i$  e  $j$  estão associados por meio de uma aresta. Caso contrário, deve ser igual a 0. Essa matriz se denomina **matriz de adjacência**. Determine a matriz de adjacência do grafo acima.

Resposta esperada:

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



### 1.3. Alguns conceitos importantes.

No estudo de grafos, alguns conceitos são essenciais para possibilitar a modelagem, que é uma descrição do problema a ser resolvido.

- \* **Ordem de um grafo:** número de vértices que ele possui.
- \* **Tamanho de um grafo:** número de arestas que ele possui.
- \* **Grafo complementar:** Se  $G$  é um grafo (orientado ou não orientado), o complementar de  $G$  é o grafo que possui exatamente as ligações que não estão presentes em  $G$ . Notação:  $G^c$  ou  $\bar{G}$ .

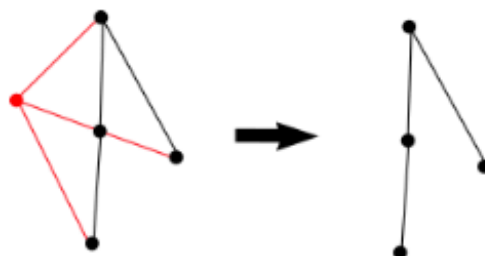
Exemplo:



Disponível em: <https://goo.gl/ye7xuT>. Acesso em 05/10/16.

- \* **Subgrafo:** Se  $H$  é um subgrafo do grafo  $G$ , dizemos que o conjunto de vértices de  $H$  está contido ou é igual ao de  $G$  e o conjunto de arestas de  $H$  está contido ou é igual ao de  $G$ .

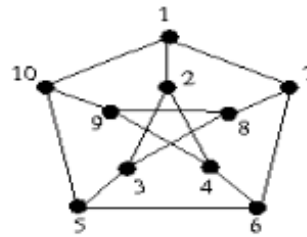
Exemplo:



Disponível em: <https://goo.gl/tV3NY6>. Acesso em 05/10/16.

- \* **Percurso em um grafo:** uma coleção de vértices (ou de arestas) sequencialmente adjacentes. Também é conhecido como cadeia.
- \* **Caminho:** percurso onde em cada vértice, a partir do inicial, há uma aresta para o próximo vértice da sequência, com exceção do vértice final. Seu comprimento é dado pelo número de arestas do percurso.
- \* **Percurso simples:** aquele que não repete ligações, ou seja, não contém arestas múltiplas.
- \* **Percurso elementar:** aquele que não repete vértices.
- \* **Ciclo:** percurso elementar fechado.
- \* **Circuito:** caminho elementar e fechado (ciclo orientado).

**ATIVIDADE 1.3.1:** Observe o grafo a seguir e dê um exemplo, se for possível, para o que se pede.



a) um percurso.

Uma resposta esperada: 1-2-4-9-10-5-6-7-8-1-10

b) caminho.

Uma resposta esperada: 1-2-4-9-10-5-6-7-8

c) percurso simples.

Uma resposta esperada: 5-6-7-1-2-3-8-7

d) percurso elementar.

Uma resposta esperada: 10-9-4-6-7-1

e) ciclo.

Uma resposta esperada: 1-7-6-5-10-1

f) circuito.

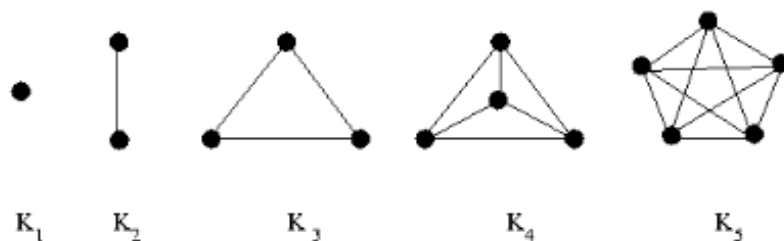
Uma resposta esperada: 1-2-3-5-6-7-1

#### 1.4. Alguns grafos especiais.

Existem grafos com características especiais, entre os quais se destacam:

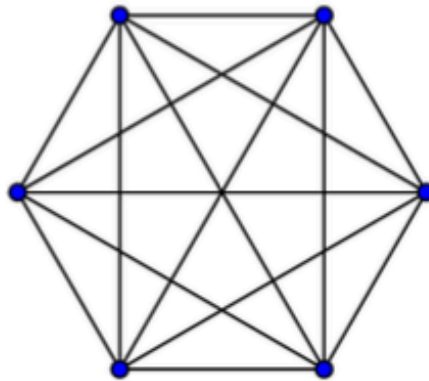
\* **Grafo completo:** possui ao menos uma ligação entre cada par de vértices.

Notação:  $K_n$ .



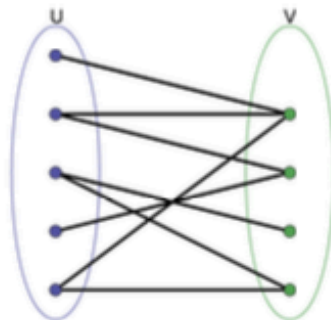
**ATIVIDADE 1.4.1:** Esboce um grafo completo  $K_6$ .

Resposta esperada:



\* **Grafo bipartido:** grafo em que seu conjunto de vértices pode ser particionado em dois subconjuntos de modo que não há ligações internas entre os vértices do mesmo subconjunto.

Exemplo:



Disponível em: <https://goo.gl/CU5UCZ>. Acesso em 05/10/16.

\* **Grafo orientado:** sua família de arestas é formada por pares ordenados, sendo que ao desenhar o grafo, a aresta parte do primeiro em direção ao segundo elemento do par.

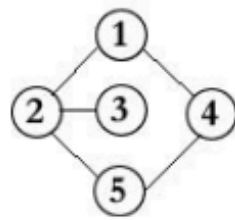
Exemplo:

|   |  |
|---|--|
| $a \longrightarrow b \longrightarrow c$ | $a \longrightarrow b \longleftarrow c$ |
| $VG_1 = \{ a,b,c \}.$                   | $VG_2 = \{ a,b,c \}.$                  |
| $AG_1 = \{ (a,b) , (b,c) \}$            | $AG_2 = \{ (a,b) , (c,b) \}$           |

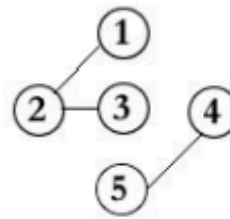
Os grafos  $G_1$  e  $G_2$  são diferentes. Se não fossem orientados seriam iguais.

\* **Grafo conexo:** sempre se pode ir de um a outro vértice, atravessando arestas.

*Exemplo:*



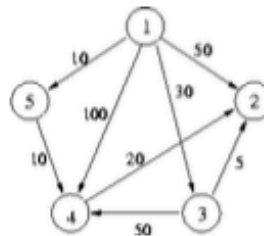
Grafo conexo



Grafo não conexo

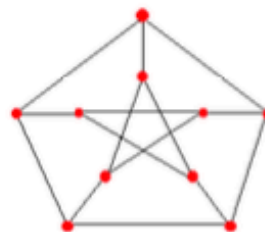
\* **Grafo valorado:** consiste de um conjunto finito não vazio de vértices  $V$ , ligados por um conjunto  $E$  de arestas com pesos.

*Exemplo:*



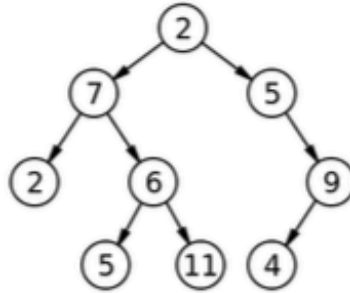
\* **Grafo  $k$ -regular:** todos os vértices possuem o mesmo grau  $k$ .

*Exemplo:* grafo de Petersen (3-regular).



\* **Árvore:** grafo conexo e sem ciclos.

*Exemplo:*



\* **Grafo planar:** aquele que pode ser imerso no plano de tal forma que suas arestas não cruzem em pontos que não são vértices.

*Exemplo:*



Fim da aula 1.



**Introdução ao estudo de Grafos com auxílio de algoritmos de programação.**

**Público alvo: Ensino Médio – alunos do segundo ano.**

Professor (mestrando): Dárcio Costa Nogueira Júnior

Professor (orientador): Luís Felipe Gonçalves Fonseca

**Aula 2: GRAFOS: problemas e algoritmos.**



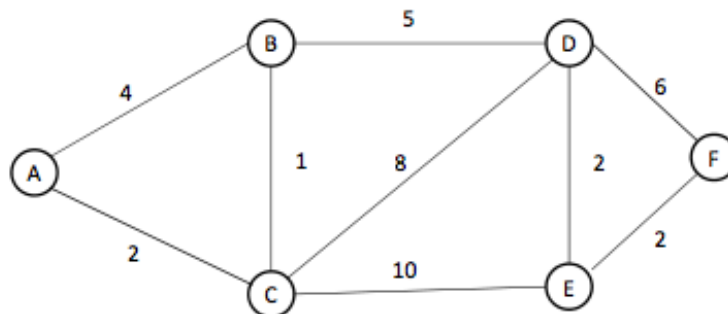
**2.1. Problemas de caminho mínimo.**

Em problemas de caminhos mínimos, o principal objetivo é determinar o caminho de menor custo, de acordo com um critério dado. Usando a linguagem de grafos, o caminho de menor custo entre dois vértices quaisquer de um grafo  $G = (V, E)$ , seja ele orientado ou não. O critério dado pode ser entre dois vértices determinados, de um vértice dado a cada um dos demais ou a união de cada par de vértices do grafo.

Um dos algoritmos usados para resolver esse tipo de problema é conhecido como **Algoritmo de Dijkstra** (concebido pelo cientista da computação holandês Edsger Dijkstra), descrito de forma sucinta como:

- 1º) Procura-se o vértice mais próximo do vértice dado;
- 2º) Depois, sucessivamente, procura-se entre os vértices não visitados aquele que tem a menor distância desde o vértice dado, diretamente ou passando por algum vértice já visitado, anotando sempre o percurso escolhido.

**ATIVIDADE 2.1.1:** Alguém necessita se deslocar de uma cidade A para a cidade F. Para isso, dispõe de várias estradas, que passam por algumas cidades. Com auxílio do grafo valorado a seguir, qual delas oferece uma trajetória de menor caminho?



Disponível em: <https://goo.gl/kl7ckD>. Acesso em 05/10/16.

**Respostas esperadas na tabela:**

| Vértice | Passo 1 | Passo 2 | Passo 3 | Passo 4 | Passo 5 | Passo 6 |
|---------|---------|---------|---------|---------|---------|---------|
| A       | 0, A    | *       | *       | *       | *       | *       |
| B       | 4, A    | 3, C    | 3, C    | *       | *       | *       |
| C       | 2, A    | 2, A    | *       | *       | *       | *       |
| D       | ∞       | 10, C   | 8, B    | 8, B    | *       | *       |
| E       | ∞       | 12, C   | 12, C   | 10, D   | 10, D   | *       |
| F       | ∞       | ∞       | ∞       | 14, D   | 12, E   | 12, E   |

### Etapas detalhadas do Algoritmo de Dijkstra:

1) Sejam  $G(V,E)$  um grafo orientado e A um vértice de G. Atribua o valor zero à estimativa do custo mínimo do vértice A (a raiz da busca) e infinito às demais estimativas;

2) Atribua um valor qualquer aos precedentes (o precedente de um vértice C é aquele vértice que precede C no caminho de custo mínimo de A para C);

3) Enquanto houver vértice aberto, adote os procedimentos a seguir:

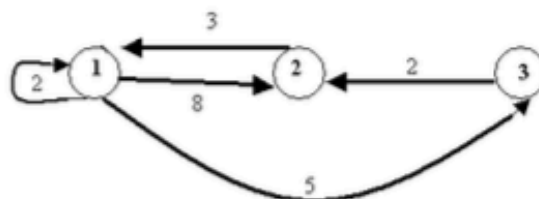
\* Seja K um vértice ainda aberto cuja estimativa seja menor dentre todos os vértices abertos;

\* Feche o vértice K;

\* Para todo vértice J ainda aberto que seja sucessor de K faça a soma da estimativa do vértice K com o custo do arco que une K a J. Caso esta soma seja melhor que a estimativa anterior para o vértice J, substitua-a e anote K como precedente de J.

Outro algoritmo usado para o problema do caminho mínimo é o **Algoritmo de Floyd**. Ele consiste em achar o caminho mais curto de cada vértice a todos os outros. Nesse caso é mais apropriado o uso da matriz de valores das ligações. Quando um vértice é adjacente a outro, a posição  $a_{ij}$  deve ser igual ao peso da aresta, indicando que os vértices  $i$  e  $j$  estão associados por meio de uma aresta. Se não há aresta ligando os vértices, deve ser igual a 0. Nessa matriz de valores, toma-se como referência um vértice para se obter a matriz de roteamento, que determina por onde passam os caminhos. Nessa matriz, todo elemento de uma coluna tem o índice da coluna, menos os que correspondem aos infinitos da outra matriz. Esses infinitos indicam que não há aresta ligando os vértices. Nesse caso, estes elementos receberão valor zero. A cada iteração, ele utiliza um vértice diferente, denominado vértice intermediário, tentando inseri-lo em itinerários unindo outros vértices.

**ATIVIDADE 2.1.2:** Determine todas as distâncias dos menores caminhos entre todos os pares de vértices do grafo abaixo.



a) Matriz de valores de ligação:

Resposta esperada:

$$D^{(0)} = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & \infty \\ \infty & 2 & 0 \end{bmatrix}$$

b) Matriz de roteamento (vértice intermediário: 1)

Resposta esperada:

$$D^{(1)} = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ \infty & 2 & 0 \end{bmatrix}$$

c) Matriz de roteamento (vértice intermediário: 2)

Resposta esperada:

$$D^{(2)} = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}$$

e) Matriz de roteamento (vértice intermediário: 3)

Resposta esperada:

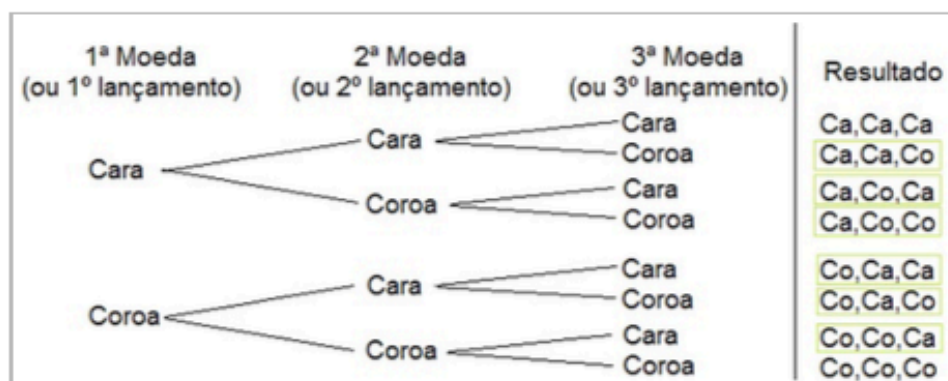
$$D^{(3)} = \begin{bmatrix} 0 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}$$

## 2.2. Problemas de interligação.

As árvores são grafos conexos e sem ciclos. Elas podem ser usadas para descrever hierarquia em uma empresa por meio de organograma ou o esquema eliminatório de um campeonato. No entanto, a **árvore de decisão** (árvore que permite listar todas as possibilidades a fim de se fazer uma escolha) é um importante instrumento para a tomada de decisões.

**ATIVIDADE 2.2.1:** Lançam-se 3 moedas não viciadas ao ar (ou lança-se uma moeda 3 vezes ao ar). Qual a probabilidade de saírem exatamente 2 vezes o mesmo lado (seja cara ou coroa)? Use diagrama de árvore.

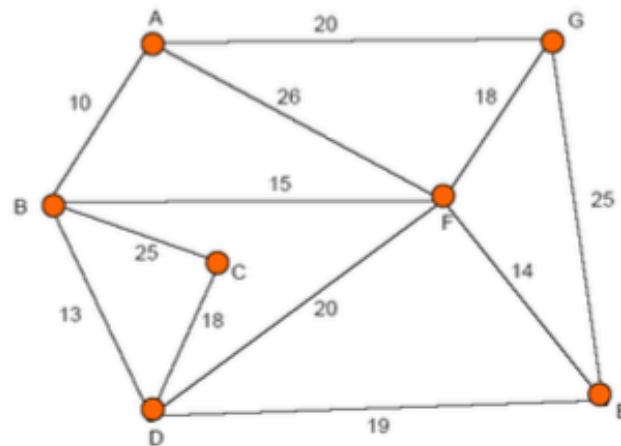
Resposta esperada:  $6/8 = 3/4$



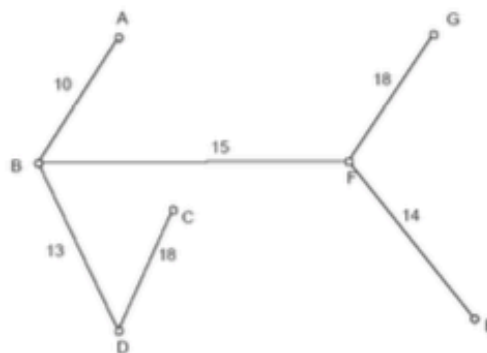


Nos problemas de caminho mínimo, procura-se saber o menor custo para se ir de um vértice ao outro. Em problemas de árvore e interligação, procura-se chegar ao destino de modo mais econômico possível. Um dos problemas é o da árvore parcial de custo mínimo, com aplicação do **Algoritmo de Kruskal**. O procedimento consiste em incluir a aresta de menor valor, sem formar ciclo, até que todos os vértices estejam interligados.

**ATIVIDADE 2.2.2:** Observe o grafo a seguir e em seguida determine a árvore parcial de custo mínimo através do Algoritmo de Kruskal.



Resposta esperada: 88 km



Esse algoritmo detalhado consiste em:

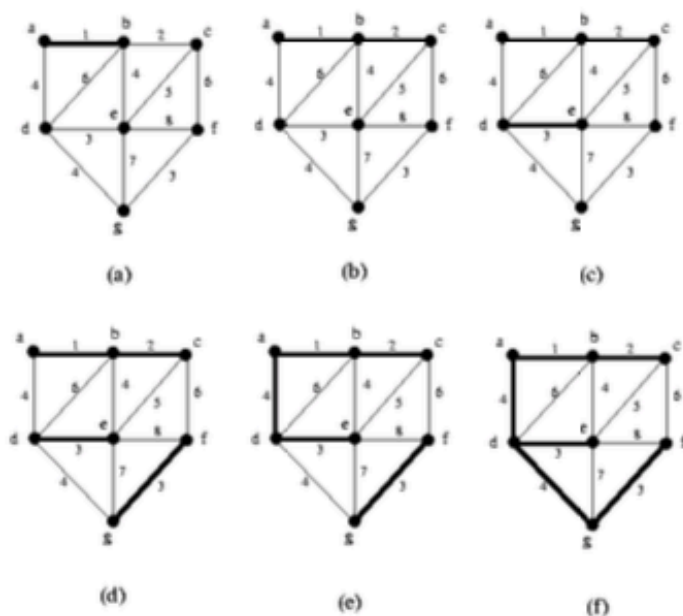
- 1) Escolher a aresta com menor peso;
- 2) Das restantes, escolher a aresta com menor peso;
- 3) Continuar sucessivamente, mas sem nunca escolher uma aresta que feche um ciclo.

O Algoritmo de Kruskal é do tipo **algoritmo guloso** devido às escolhas pela melhor solução imediata, sem pensar nas futuras consequências. No entanto são raros os casos em que funcionam bem. E felizmente, o de Kruskal funciona bem e isso pode ser provado matematicamente.

**TEOREMA 2.2.3:** O algoritmo de Kruskal fornece uma solução ótima (solução possível que otimiza a função objetivo) para o problema da interligação a custo mínimo.

**Demonstração:** Suponha que  $T$  não tenha menor peso. Nesse caso existe uma árvore  $T'$  tal que seu peso é menor que o peso de  $T$ . Seja  $e$  a primeira aresta para  $T$  de modo que ela não pertença a  $T'$ . Adicionando  $e$  a  $T'$  obtemos um ciclo que contém uma aresta  $f$  que não está em  $T$ . Retirando  $f$  temos uma árvore  $T''$  com peso menor que  $T$ . Nessa situação a aresta  $f$  teria sido escolhida no lugar de  $e$ . Logo, o algoritmo constrói efetivamente uma árvore de menor peso.

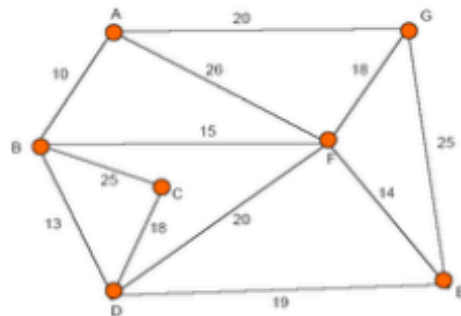
**ATIVIDADE 2.2.4:** Dê um exemplo que ilustre a demonstração do teorema 2.2.3.



Outro algoritmo para resolver problemas de interligação é o **Algoritmo de Prim**. De modo resumido, ele consiste em formar uma árvore e a cada passo incluir a aresta de menor peso que ligue um vértice fora da árvore a ela. Observe que esse método por vezes pode levar a um beco sem saída.

- 1) Começar num vértice qualquer;
- 2) Ir para o vizinho mais "próximo";
- 3) Daí para o vizinho "mais próximo" e assim sucessivamente até chegar ao último (sem fechar ciclo).

**ATIVIDADE 2.2.5:** Determine a árvore parcial de custo mínimo do grafo a seguir pelo algoritmo de Prim.



Por vezes este método pode levar a um beco sem saída, que é o que acontece se começarmos em A: AB, BD, DC. Não podemos continuar senão fechamos o ciclo.

Uma solução poderá ser:

AB, BD, DC, BF, FE, EG.

### 2.3. Problemas de coloração.

O número cromático de um grafo é o número mínimo necessário para colorir os seus vértices de forma a que dois vértices adjacentes não possuem a mesma cor. O número cromático das arestas de um grafo é o número mínimo de cores necessárias para colorir as arestas de  $G$ , de forma que as arestas incidentes num mesmo vértice não possam ter a mesma cor.

**ATIVIDADE 2.3.1:** Uma companhia industrial deseja armazenar sete diferentes produtos farmacêuticos  $C_1, C_2, \dots, C_7$ , mas alguns não podem ser armazenados juntos por motivos de segurança. A tabela a seguir mostra os produtos que não podem estar no mesmo local. Encontre o número mínimo de localizações necessárias para colocar estes produtos e escreva uma combinação possível dos medicamentos compatíveis.

|    | C1 | C2 | C3 | C4 | C5 | C6 | C7 |
|----|----|----|----|----|----|----|----|
| C1 |    | X  |    |    |    | X  | X  |
| C2 | X  |    | X  | X  |    |    |    |
| C3 |    | X  |    | X  | X  |    |    |
| C4 |    | X  | X  |    | X  | X  |    |
| C5 |    |    | X  | X  |    | X  | X  |
| C6 | X  |    |    | X  | X  |    | X  |
| C7 | X  |    |    |    | X  | X  |    |

Disponível em: <https://goo.gl/5scbLO>. Acesso em 05/10/16.

Resposta esperada:

O número cromático deste grafo é 4. Precisamos de quatro gavetas para colocar os medicamentos. Uma combinação possível dos medicamentos compatíveis é: C2C5, C3C6, C4C7, C1. Esta combinação não é única, pois a disposição dos produtos compatíveis pode variar consoante a coloração do grafo.

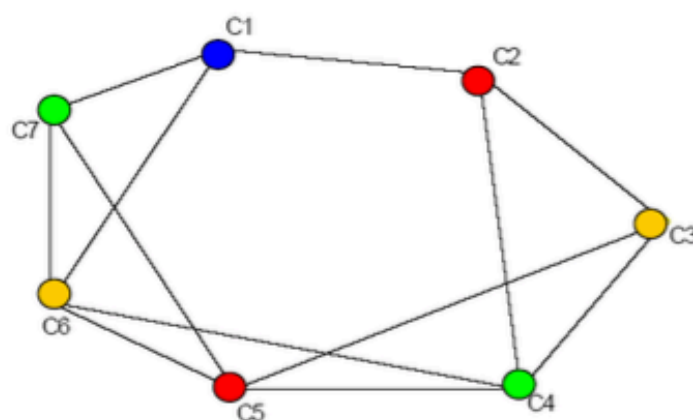
**TEOREMA 2.3.2 (TEOREMA DAS QUATRO CORES):** Qualquer mapa de regiões, desenhado num plano pode ser colorido com quatro cores de modo a que regiões adjacentes (arestas comuns ou parcialmente comuns) fiquem com cores diferentes.

*Exemplo:* mapa da Europa.



Disponível em: <https://goo.gl/5scbLO>. Acesso em 05/10/16.

**ATIVIDADE 2.3.3:** Refaça a atividade 2.3.1 utilizando o teorema das quatro cores.



Fim da aula 2



## Introdução ao estudo de Grafos com auxílio de algoritmos de programação.

Público alvo: Ensino Médio – alunos do segundo ano.

Professor (mestrando): Dárcio Costa Nogueira Júnior

Professor (orientador): Luís Felipe Gonçalves Fonseca

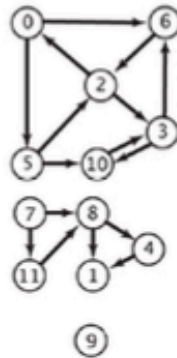
Aula 3: GRAFOS: algoritmos de programação.



### 3.1. Ideias básicas.

Um digrafo é um par de conjuntos: de arestas e de vértices. Cada aresta dirigida (ou arco) é um par ordenado de arestas.

**ATIVIDADE 3.1.1:** Especifique o digrafo a seguir por meio de seu conjunto de arcos (arestas dirigidas).



Algumas ferramentas básicas para a construção e manipulação de digrafos.

```
/* REPRESENTAÇÃO POR MATRIZ DE ADJACÊNCIAS: A função DIGRAPHinit()
constrói um digrafo com vértices 0 1 .. V-1 e nenhum arco. */
Digraph DIGRAPHinit( int V) {
    Digraph G = malloc( sizeof *G);
    G->V = V;
    G->A = 0;
    G->adj = MATRIXint( V, V, 0);
    return G;
}
```

```

/* REPRESENTAÇÃO POR MATRIZ DE ADJACÊNCIAS: A função MATRIXint() aloca
uma matriz com linhas 0..r-1 e colunas 0..c-1. Cada elemento da matriz
recebe valor val. */
static int **MATRIXint( int r, int c, int val) {
    Vertex i, j;
    int **m = malloc( r * sizeof (int *));
    for (i = 0; i < r; i++)
        m[i] = malloc( c * sizeof (int));
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            m[i][j] = val;
    return m;
}

/* REPRESENTAÇÃO POR MATRIZ DE ADJACÊNCIAS: A função DIGRAPHinsertA()
insere um arco v-w no digrafo G. A função supõe que v e w são
distintos, positivos e menores que G->V. Se o digrafo já tem um arco
v-w, a função não faz nada. */
void DIGRAPHinsertA( Digraph G, Vertex v, Vertex w) {
    if (G->adj[v][w] == 0) {
        G->adj[v][w] = 1;
        G->A++;
    }
}

/* REPRESENTAÇÃO POR MATRIZ DE ADJACÊNCIAS: A função DIGRAPHremoveA()
remove do digrafo G o arco v-w. A função supõe que v e w são
distintos, positivos e menores que G->V. Se não existe arco v-w, a
função não faz nada. */
void DIGRAPHremoveA( Digraph G, Vertex v, Vertex w) {
    if (G->adj[v][w] == 1) {
        G->adj[v][w] = 0;
        G->A--;
    }
}

/* REPRESENTAÇÃO POR MATRIZ DE ADJACÊNCIAS: A função DIGRAPHshow()
imprime, para cada vértice v do digrafo G, em uma linha, todos os
vértices adjacentes a v. */
void DIGRAPHshow( Digraph G) {
    Vertex v, w;
    for (v = 0; v < G->V; v++) {
        printf( "%2d:", v);
        for (w = 0; w < G->V; w++)
            if (G->adj[v][w] == 1)
                printf( " %2d", w);
        printf( "\n");
    }
}

```

### 3.2. Grafos, algoritmos e programação.

Dado um vértice  $s$  de um digrafo com custos positivos nos arcos, encontrar um caminho mínimo com raiz  $s$  no digrafo. (*Algoritmo de Dijkstra*).

```
/* Recebe digrafo G com custos positivos nos arcos e um vértice s.
Calcula uma SPT com raiz s e armazena a SPT no vetor parent. As
distâncias a partir de s são armazenadas no vetor dist. */
/* O digrafo é representado por suas listas de adjacência. A função
supõe que a soma dos custos de todos os arcos é estritamente menor que
a constante INFINITO. Supõe também que o digrafo tem no máximo maxV
vértices. (O código abaixo é uma versão modificada do Programa 20.3
de Sedgewick.) */
```

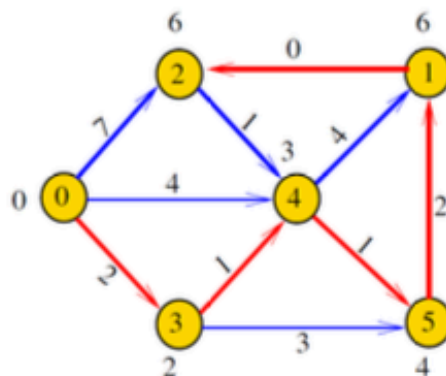
```
void DIGRAPHsptD1( Digraph G, Vertex s, Vertex parent[],
                  double dist[])
{
    Vertex v0, w, frj[maxV];
    link a; double c;
    initialize( G, s, parent, dist, frj);

    while (1) {
        double mindist = INFINITO;
        for (w = 0; w < G->V; w++)
            if (parent[w] == -1 && mindist > dist[w])
                mindist = dist[v0=w];
        if (mindist == INFINITO) break;
        parent[v0] = frj[v0];
        for (a = G->adj[v0]; a != NULL; a = a->next) {
            w = a->w, c = a->cost;
            if (parent[w] == -1 && dist[w] > dist[v0] + c) {
                dist[w] = dist[v0] + c;
                frj[w] = v0;
            }
        }
    }
}
```

```
/* Inicializa os vetores parent, dist e frj de modo que a primeira
iteração de DIGRAPHsptD1 comece com uma árvore radicada de um só
vértice, s, e a correspondente franja. */
```

```
void initialize( Digraph G, Vertex s, Vertex parent[],
                double dist[], Vertex frj[])
{
    Vertex w; link a; double c;
    for (w = 0; w < G->V; w++) {
        parent[w] = -1;
        dist[w] = INFINITO;
    }
    parent[s] = s;
    dist[s] = 0.0;
    for (a = G->adj[s]; a != NULL; a = a->next) {
        w = a->w, c = a->cost;
        dist[w] = c;
        frj[w] = s;
    }
}
```

**ATIVIDADE 3.2.1:** Considere o digrafo com custos nos arcos definido a seguir. Use o algoritmo de Dijkstra para encontrar uma árvore de custo mínimo com raiz 0.



Disponível em: <https://goo.gl/X4EzXQ>. Acesso em 05/10/16.

**Resultado impresso:**



Encontrar uma árvore geradora de custo mínimo de um grafo com custos nas arestas.

```
typedef struct {
    Vertex v, w;
    double cost;
} Edge;

#define Graph Digraph

/* A função GRAPHmstK recebe um grafo conexo G com custos arbitrários
nas arestas e calcula uma MST de G. O grafo é dado por suas listas de
adjacência. A função armazena as arestas da MST no vetor mst[0..k-1] e
devolve k. (Como G é conexo, k será igual a G->V-1.) Essa função é
uma implementação do algoritmo de Kruskal.*/
/* A função supõe que o grafo tem no máximo maxE arestas. Supõe
também que INFINITO é estritamente maior que o custo de qualquer
aresta. O código foi copiado, com ligeiras modificações, do programa
20.5 de Sedgewick. */

int GRAPHmstK( Graph G, Edge mst[])
{
    Vertex v, w, x, y;
    int i, k, E = G->A/2;
    Edge e[maxE];
    GRAPHedges( G, e);
    sort( e, 0, E-1);

    Ufinit( G->V);
    for (i = k = 0; i < E && k < G->V-1; i++) {
        x = Uffind( e[i].v); y = Uffind( e[i].w);
        if (x != y) {
            Ufunion( x, y);
            mst[k++] = e[i];
        }
    }
    return k;
}

/* Esta função armazena as arestas do grafo G no vetor vetor e[0..E-
1]. */

void GRAPHedges( Graph G, Edge e[])
{
    int i = 0; link a;
    for (v = 0; v < G->V; ++v)
        for (a = G->adj[v]; a != NULL; a = a->next)
            if (v < a->w)
                e[i++] = EDGE( v, a->w, a->cost);
}

/* A função EDGE constrói uma aresta com pontas v e w e custo cost. */

Edge EDGE( Vertex v, Vertex w, double cost) {
    Edge e;
    e.v = v, e.w = w;
    e.cost = cost;
    return e;
}
```

```

/* A função Uffind devolve o chefe de v (ou seja, o chefe da árvore
que contém v na floresta geradora mst[0..k-1]). A função UFunction
recebe dois chefes distintos x e y e faz a união das correspondentes
árvores. */
static Vertex ch[maxV];
static int sz[maxV];

void Ufinit( int V) {
    Vertex v;
    for (v = 0; v < V; v++) {
        ch[v] = v;
        sz[v] = 1;
    }
}

Vertex Uffind( Vertex v) {
    Vertex x = v;
    while (x != ch[x])
        x = ch[x];
    return x;
}

void UFunction( Vertex x, Vertex y) {
    if (sz[x] < sz[y]) {
        ch[x] = y;
        sz[y] += sz[x];
    }
    else {
        ch[y] = x;
        sz[x] += sz[y];
    }
}

```

**ATIVIDADE 3.2.2:** Encontrar a árvore geradora de custo mínimo no grafo de custos definido a seguir:

|     |     |     |     |     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0-2 | 0-3 | 1-2 | 1-4 | 1-6 | 2-3 | 2-4 | 3-4 | 3-5 | 4-5 | 4-6 | 5-6 |
| .5  | .8  | 1.6 | 3.0 | 2.6 | 1.0 | .3  | .2  | 1.8 | 1.2 | 1.4 | .4  |

**Resultado impresso:**

**Atividade 3.2.4:** É possível que os cavalos da figura 1 fiquem na posição da figura

2? (Extraído de <https://goo.gl/j4THmd>. Acesso em 05/10/16).

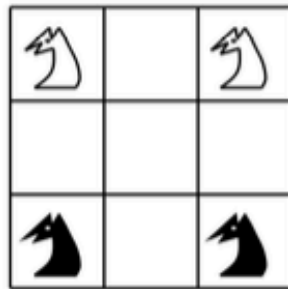


Figura 1

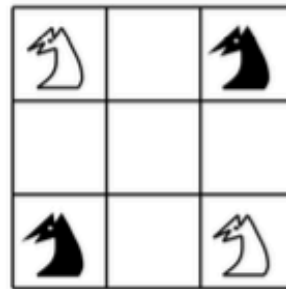


Figura 2

**Impressão:**

Fim da aula 3

Atividades adaptadas e códigos transcritos.

Disponível em [http://www.ime.usp.br/~pf/algoritmos\\_para\\_grafos/](http://www.ime.usp.br/~pf/algoritmos_para_grafos/) Acesso em 05/10/16.

# Bibliografia

---

- [1] *Code::Blocks - Main Page*. <http://wiki.codeblocks.org>. Acessado em 28/12/2016.
- [2] *Code::Blocks - Screenshots*. <http://www.codeblocks.org/screenshots>. Acessado em 28/12/2016.
- [3] *Every connected graph has a spanning tree*. <https://www.coursehero.com/file/p6tfdb4/Proposition-Every-connected-graph-has-a-spanning-tree-Proof-Let-G-be-any>. Acessado em 08/01/2017.
- [4] *Grafos Hamiltonianos*. <http://www.inf.ufsc.br/grafos/temas/hamiltoniano/cavalo.htm>. Acessado em 15/11/2016.
- [5] *Isomorfismo de Grafos*. [http://www.igm.mat.br/aplicativos/index.php?option=com\\_content&view=article&id=811:isomorfismos-grafos&catid=80:grafos](http://www.igm.mat.br/aplicativos/index.php?option=com_content&view=article&id=811:isomorfismos-grafos&catid=80:grafos). Acessado em 20/10/2016.
- [6] *Parâmetros Curriculares Nacionais - Ensino Médio*. <http://portal.mec.gov.br/seb/arquivos/pdf/ciencian.pdf>. Acessado em 27/12/2016.
- [7] *PCN+ Ensino Médio*. <http://portal.mec.gov.br/seb/arquivos/pdf/CienciasNatureza.pdf>. Acessado em 27/12/2016.
- [8] *Teoria dos Grafos - aula 9*. [http://www.land.ufrj.br/~classes/grafos/slides/aula\\_9.pdf](http://www.land.ufrj.br/~classes/grafos/slides/aula_9.pdf). Acessado em 12/01/2017.
- [9] *XVIII Olimpíada Brasileira de Informática*. <http://olimpiada.ic.unicamp.br/info/geral>. Acessado em 27/12/2016.
- [10] Appel, Kenneth I e Wolfgang Haken: *Every planar map is four colorable*, volume 98. American mathematical society Providence, RI, 1989.
- [11] Boaventura Netto, Paulo Oswald e Samuel Jurkiewicz: *Grafos: introdução e prática*. 2009.
- [12] Brassard, Gilles e Paul Bratley: *Algorithmics: theory & practice*. Prentice-Hall, Inc., 1988.
- [13] Campos, Vânia Barcelos G.: *Algoritmos para resolução de problemas em rede*. <http://aquarius.ime.eb.br/~webde2/prof/vania/apostilas/Apostila-Redes.pdf>. Acessado em 20/12/2016.
- [14] Carvalho, Marco Antonio M.: *Teoria dos Grafos*. [http://www.decom.ufop.br/marco/site\\_media/uploads/bcc204/16\\_aula\\_16.pdf](http://www.decom.ufop.br/marco/site_media/uploads/bcc204/16_aula_16.pdf). Acessado em 31/12/2016.
- [15] Carvalho, Marco Antonio M.: *Teoria dos Grafos*. [http://www.decom.ufop.br/marco/site\\_media/uploads/bcc204/12\\_aula\\_12.pdf](http://www.decom.ufop.br/marco/site_media/uploads/bcc204/12_aula_12.pdf). Acessado em 25/10/2016.
- [16] Castro, Marcos: *Caminhos mínimos - Algoritmo de Dijkstra*. <http://pt.slideshare.net/mcastrosouza/caminhos-mnimos-algoritmo-de-dijkstra>. Acessado em 12/12/2016.
- [17] Euler, Leonhard: *Solutio problematis ad geometriam situs pertinentis*, volume 8. 1741.
- [18] Feofiloff, Paulo: *Algoritmos para Grafos em C via Sedgewick*. [https://www.ime.usp.br/~pf/algoritmos\\_para\\_grafos](https://www.ime.usp.br/~pf/algoritmos_para_grafos). Acessado em 05/10/2016.

- [19] Furtado, Gustavo: *O que é um algoritmo?* <http://www.dicasdeprogramacao.com.br/o-que-e-algoritmo/>. Acessado em 20/12/2016.
- [20] Gagnon, Michel: *Algoritmos e teoria dos grafos*. [http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/CaminhoMin/caminho\\_min.html](http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/CaminhoMin/caminho_min.html). Acessado em 20/12/2016.
- [21] Gagnon, Michel: *Grafos Eulerianos e Hamiltonianos*. [http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/EulerHam/euler\\_ham.html](http://www.professeurs.polymtl.ca/michel.gagnon/Disciplinas/Bac/Grafos/EulerHam/euler_ham.html). Acessado em 20/10/2016.
- [22] GARNICA, Antonio Vicente Marafioti: *História oral e educação matemática*. Pesquisa qualitativa em educação matemática. Belo Horizonte: Autêntica, páginas 77–98, 2004.
- [23] Heawood, Percy John: *Map-colour theorem*. Proceedings of the London Mathematical Society, 2(1):161–175, 1949.
- [24] Jurkiewicz, Samuel: *Grafos, uma introdução*. <http://www.obmep.org.br/docs/apostila5.pdf>. Acessado em 22/11/2016.
- [25] Lima, Elon Lages, Paulo Cezar Pinto Carvalho, Eduardo Wagner e Augusto César Morgado: *A matemática do ensino médio - volume 2*. SBM, 1998.
- [26] Loureiro, Antônio Alfredo Ferreira: *Grafo planar*. [http://homepages.dcc.ufmg.br/~loureiro/md/md\\_9Grafos\\_MaterialExtra.pdf](http://homepages.dcc.ufmg.br/~loureiro/md/md_9Grafos_MaterialExtra.pdf). Acessado em 10/12/2016.
- [27] Loureiro, Antonio Alfredo Ferreira: *Grafos*. [http://homepages.dcc.ufmg.br/~loureiro/md/md\\_9Grafos.pdf](http://homepages.dcc.ufmg.br/~loureiro/md/md_9Grafos.pdf). Acessado em 05/10/2016.
- [28] M. Toffolo, Túlio Ângelo: *Algoritmos e programação avançada*. [http://www.decom.ufop.br/toffolo/site\\_media/uploads/2011-1/bcc402/slides/03.\\_grafos.pdf](http://www.decom.ufop.br/toffolo/site_media/uploads/2011-1/bcc402/slides/03._grafos.pdf). Acessado em 09/01/2017.
- [29] Madeira, Tiago: *Representando grafos na programação*. <https://tiagomadeira.com/2006/01/representando-grafos-na-programacao/>. Acessado em 20/12/2016.
- [30] Medeiros, Esdras: *Teoria dos Grafos*. [http://www.mat.ufc.br/~esdras/matdisc/aula\\_grafos.pdf](http://www.mat.ufc.br/~esdras/matdisc/aula_grafos.pdf). Acessado em 20/10/2016.
- [31] Nacionais-PCN, Parâmetros Curriculares: *Matemática*. Brasília: Secretaria de Educação Fundamental, 1998.
- [32] Nunes, Flávio Humberto Cabral: *Grafos Planares*. <http://homepages.dcc.ufmg.br/~rainerpc/cursos/grafos/aulas/a12.pdf>. Acessado em 22/11/2016.
- [33] Nívio Ziviani, Charles Ornelas Almeida e: *Algoritmos em grafos*. <http://www2.dcc.ufmg.br/livros/algoritmos-edicao2/cap7/transp/completo4/cap7.pdf>. Acessado em 20/12/2016.
- [34] Poggi, Marcus: *Grafos e algoritmos via indução*. <https://www-di.inf.puc-rio.br/~poggi//chap4.pdf>. Acessado em 20/12/2016.
- [35] Prestes, Edson: *Grafos A2 - UFRGS*. <http://www.inf.ufrgs.br/~prestes/Courses/Graph%20Theory/GrafosA2.pdf>. Acessado em 15/11/2016.
- [36] Ribeiro, Cristina: *Árvores de expansão mínimas - Algoritmo de Prim*. <https://web.fe.up.pt/~aed2/acetatos/arvexpan.pdf>. Acessado em 26/12/2016.
- [37] Ribeiro, Pedro: *Árvores de Suporte de Custo Mínimo*. [http://www.dcc.fc.up.pt/~pribeiro/aulas/daa1415/slides/7\\_mst\\_30112014.pdf](http://www.dcc.fc.up.pt/~pribeiro/aulas/daa1415/slides/7_mst_30112014.pdf). Acessado em 13/01/2017.
- [38] Rodrigues, Helder: *As pontes de Königsberg*. <http://www.helderrodrigues.eu/2008/03/28/o-problema-das-pontes-de-konigsberg/>. Acessado em 11/01/2017.
- [39] Rossetti, R.: *Algoritmos em Grafos: Circuitos de Euler e Problema do Carteiro Chinês*. [http://paginas.fe.up.pt/~rossetti/rrwiki/lib/exe/fetch.php?media=teaching:1011:cal:08\\_2.09\\_1.grafos6.pdf](http://paginas.fe.up.pt/~rossetti/rrwiki/lib/exe/fetch.php?media=teaching:1011:cal:08_2.09_1.grafos6.pdf). Acessado em 10/12/2016.

- [40] Sampaio, João Carlos V.: *Quatro Cores e Matemática*. [http://www.dm.ufscar.br/profs/sampaio/Quatrocores\\_2aBienalsbmretocado.pdf](http://www.dm.ufscar.br/profs/sampaio/Quatrocores_2aBienalsbmretocado.pdf). Acessado em 10/12/2016.
- [41] Sousa, Diego: *Por que só existem 5 sólidos platônicos?* <http://gigamatematica.blogspot.com.br/2013/07/por-que-so-existem-5-solidos-platonicos.html>. Acessado em 08/01/2017.
- [42] Sousa, Rafael Castro de: *Uma abordagem paralela do algoritmo de Floyd para solução do problema do caminho mínimo*. <http://cdsid.org.br/sbpo2015/wp-content/uploads/2015/08/142081.pdf>. Acessado em 12/01/2017.
- [43] Tofflo, Túlio: *Estrutura de Dados I - Análise de Algoritmos*. [http://www.decom.ufop.br/reinaldo/site\\_media/uploads/2013-02-bcc202/aula\\_06\\_-\\_analise\\_de\\_algoritmos\\_\(parte\\_3\)\\_v4.pdf](http://www.decom.ufop.br/reinaldo/site_media/uploads/2013-02-bcc202/aula_06_-_analise_de_algoritmos_(parte_3)_v4.pdf). Acessado em 08/01/2017.
- [44] Vieira, Newton José: *Introdução à Teoria dos Grafos baseado em Townsend*. <http://homepages.dcc.ufmg.br/~nvieira/cursos/md/slides/cap6.tex>. Acessado em 05/10/2016.
- [45] Wakabayashi, Yoshiko: *Euler e as origens da Teoria de Grafos*. <http://www.ime.usp.br/~yw/2016/grafinhos/aulas/Euler-yw-usp-2007.pdf>. Acessado em 05/10/2016.
- [46] Zabala, Antoni: *A prática educativa: como ensinar*. Artmed, 1998.