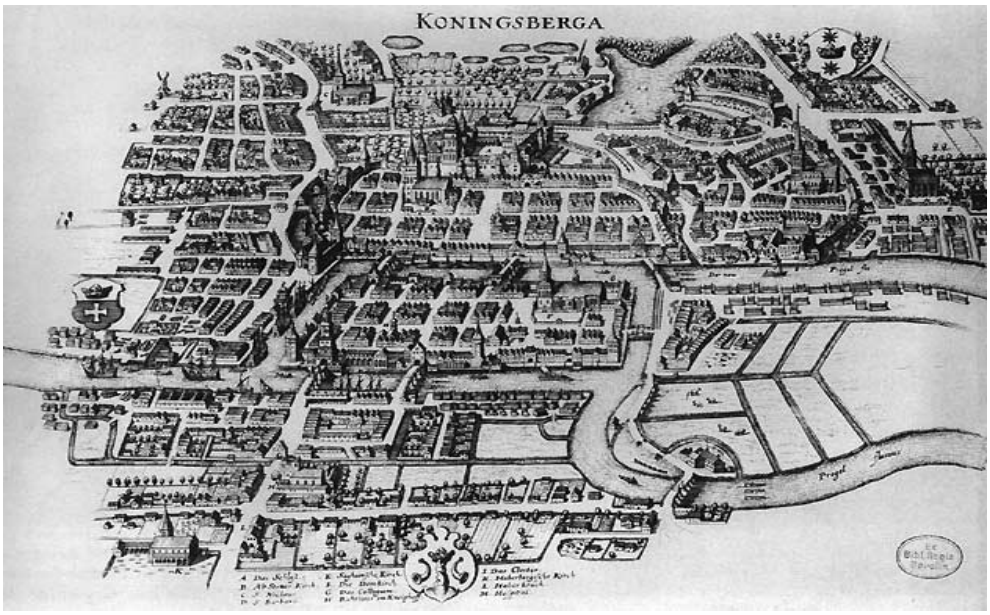
Algoritmos e Teoria dos Grafos

Jair Donadelli[1]

ultima revisao, 5 de julho de 2008

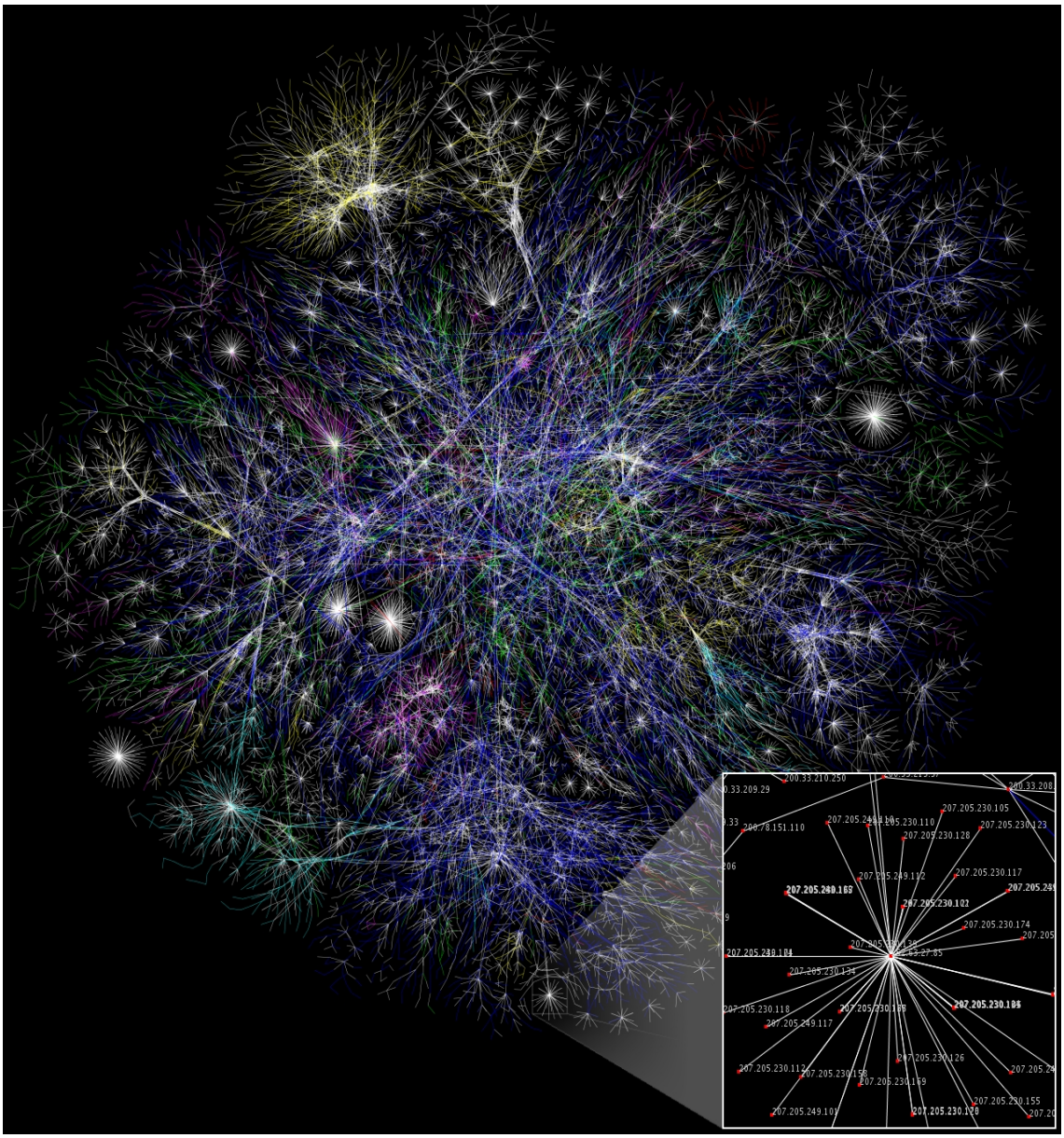
[1] A versão eletrônica desse texto contém hyperlinks que ilustram a discussão de alguns tópicos da disciplina. Eu tomei o cuidado de fazer ligações com páginas WEB que tinham, no momento que fiz o acesso, informações corretas, entretanto essas páginas estão fora do meu controle e podem sofrer alterações, portanto leia com cautela.



Mapa de Königsberg em 1652. Fonte: http://www.preussen-chronik.de/_/bild_jsp/key=bild_kathe2.html (domínio público)

"*In 1736, Euler solved the problem known as the Seven Bridges of Königsberg. The city of Königsberg, Prussia was set on the Pregel River, and included two large islands which were connected to each other and the mainland by seven bridges. The problem is to decide whether it is possible to follow a path that crosses each bridge exactly once and returns to the starting point. It is not: there is no Eulerian circuit. This solution is considered to be the first theorem of graph theory*"

Wikipedia (en.wikipedia.org/wiki/Leonhard_Euler)



Grafo da Internet em 2005.

"*Each line is drawn between two nodes, representing two IP addresses. The length of the lines are indicative of the delay between those two nodes. This graph represents less than 30% of the Class C networks reachable by the data collection program in early 2005. Lines are color-coded according to their corresponding RFC 1918 allocation as follows:*

Dark blue: net, ca, us
Green: com, org
Red: mil, gov, edu
Yellow: jp, cn, tw, au, de
Magenta: uk, it, pl, fr
Gold: br, kr, nl
White: unknown"

Sumário

Símbolos

- $\mathbb{N}$ Denota o conjunto dos números naturais;
- $\mathbb{Z}$ denota o conjunto dos números inteiros;
- $\mathbb{Q}$ denota o conjunto dos números racionais;
- $\mathbb{R}$ denota o conjunto dos números reais;
- $\mathbb{R}^+$ denota o conjunto dos números reais positivos;
- se a e b são números inteiros (resp., reais) então $[a, b]$ é o intervalo do números inteiros (resp., reais) entre a e b inclusive (intervalo fechado);
- $|X|$ denota a cardinalidade do conjunto X;
- para $X \subseteq V$ denotamos por $\overline{X}$ o complemento de X em V, isto é, o conjunto $V \setminus X$;
- $\binom{V}{2}$ denota o conjunto $\{\{x, y\} \subseteq V : x \neq y\}$ dos subconjuntos de V de cardinalidade 2;
- $\lfloor x \rfloor = \max\{n \in \mathbb{Z} : n \leq x\}$ para todo x real;
- $\lg(x)$ é a função logaritmo na base 2;
- para qualquer família de conjuntos $A_1, A_2, \ldots, A_m$

$$\bigcup_{i=1}^{m} A_i = A_1 \cup A_2 \cup \cdots \cup A_m;$$

- denotamos por $A \triangle B$ a diferença simétrica dos conjuntos A e B, isto é, $A \triangle B = (A \cup B) \setminus (A \cap B)$;
- *Notação assintótica*: Sejam f_n e g_n seqüências de números reais, onde $f_n > 0$ para todo n suficientemente grande. Usaremos as seguintes notações para o comportamento assintótico dessas seqüências:
  - $g_n = O(f_n)$, quando $n \to \infty$, se existem constantes positivas $c \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $|g_n| \leq cf_n$, para todo $n \geq n_0$;
  - $g_n = \Omega(f_n)$, quando $n \to \infty$, se existem constantes positivas $C \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $g_n \geq Cf_n$, para todo $n \geq n_0$;
  - $g_n = \Theta(f_n)$, quando $n \to \infty$, se existem constantes positivas c, $C \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $Cf_n \leq g_n \leq cf_n$, para todo $n \geq n_0$;

CONCEITOS BÁSICOS

Neste capítulo apresentamos os conceitos mais básicos da Teoria dos Grafos dos pontos de vista combinatório e algorítmico, também aqui convenciona-se as notações que serão usadas por todo o texto. Nas primeiras seções apresentamos a definição de grafo e alguns parâmetros e aspectos estruturais. Os conceitos algorítmicos aparecerão a partir da sétima seção com representações de grafos apropriadas para tratar problemas computacionais. O capítulo termina com um algoritmo para percorrer grafos e que serve como base para muitos outros algoritmos que veremos nos capítulos posteriores.

1.1 Grafos—definições iniciais

Um **grafo** é um par ordenado de conjuntos finitos (V, E) tal que $E \subseteq \binom{V}{2}$. Cada elemento de V é chamado de **vértice** do grafo e cada elemento de E é chamado de **aresta** do grafo.

Todo grafo pode ser *representado* geometricamente por um **diagrama**. No plano, desenhamos um ponto para cada vértice e um segmento de curva ligando cada par de vértices que determinam uma aresta (figura 1.1). Claramente, a representação geométrica de um grafo não é única, mas para cada representação existe um único grafo.

Exemplo 1. Os conjuntos

$$\begin{aligned} V &= \{1,2,3,4,5,6,7,8\} \\ E &= \big\{\{1,2\},\{1,5\},\{2,5\},\{3,4\},\{6,7\}\big\} \end{aligned}$$

definem um grafo e um diagrama desse grafo é apresentado na figura 1.1 a seguir.

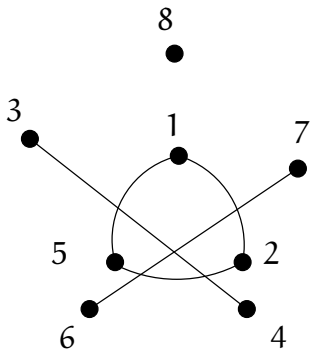


Figura 1.1: Representação geométrica (ou *diagrama*) do grafo.

Se G denota o grafo que é definido pelo par (V, E) então escrevemos $\mathbf{G} = (\mathbf{V}, \mathbf{E})$.

Sejam u e v vértices de um grafo G; se $\{u, v\}$ é uma aresta de G, então dizemos que u e v são vértices **adjacentes**, também dizemos que u e v são **extremos** de uma aresta; dizemos que duas arestas são **arestas adjacentes** se elas têm um vértice em comum.

Quando nos referimos a um grafo conhecido G sem especificarmos o conjunto dos vértices e o conjunto das arestas que definem G esses passam a ser referidos como $\mathbf{V}(\mathbf{G})$ e $\mathbf{E}(\mathbf{G})$, respectivamente.

Para um grafo G qualquer, chamamos $|V(G)|$ de **ordem** de G e chamamos $|V(G)| + |E(G)|$ de **tamanho** de G, por exemplo, o grafo do exemplo 1 tem ordem 8 e tamanho 13.

Um expoente em G, quando G é um grafo, denota a ordem de G, assim quando queremos ressaltar que G é um grafo de ordem n, para algum $n \in \mathbb{N}$, escrevemos $\mathbf{G^n}$.

Chamamos $G^0 = (\varnothing, \varnothing)$ de **grafo vazio** e *todo* grafo de ordem 1 de **grafo trivial**.

1.1.1 Grau

Para um vértice v qualquer num grafo G, definimos os conjuntos

$$E_G(v) = \{\{x, y\} \in E(G) : x = v \text{ ou } y = v\}, \tag{1.1}$$

e

$$N_G(v) = \{w \in V(G) : \{v, w\} \in E(G)\}, \tag{1.2}$$

esse último chamado de **vizinhança** de v. Os elementos de $N_G(v)$ são chamados de **vizinhos** de v.

Para todo $v \in V(G)$, o número de vizinhos de v é chamado de **grau** do vértice v no grafo G. Os graus dos vértices de um grafo é um dos seus parâmetros importantes e por isso recebem uma notação especial. Para um grafo $G = (V, E)$ não-vazio

$$\begin{aligned}
\textbf{grau } \text{de } u \text{ em } G\text{:} \quad d_G(u) &= |N_G(u)| && (1.3)\\
\textbf{grau mínimo } \text{em } G\text{:} \quad \delta(G) &= \min\{d_G(u) : u \in V\} && (1.4)\\
\textbf{grau máximo } \text{em } G\text{:} \quad \Delta(G) &= \max\{d_G(u) : u \in V\} && (1.5)\\
\textbf{grau médio } \text{em } G\text{:} \quad d(G) &= \frac{1}{|V|}\sum_{u \in V} d_G(u) && (1.6)
\end{aligned}$$

Observação 1. As vezes suprimimos os índices $\_G$ ou os argumentos $\_(G)$ para simplificar a notação; escrevemos, por exemplo, $E(v)$ e $N(v)$ para os conjuntos definidos acima, e escrevemos Δ para o grau máximo e $d(v)$ para o grau de v.

Exemplo 2. No exemplo 1 encontramos $d(7) = |E(7)| = d(6) = |E(6)| = 1$ e $d(5) = |E(5)| = 2$. Também, o grau máximo é $\Delta(G) = 2$, o grau mínimo é $\delta(G) = 0$ e o grau médio é $d(G) = 5/4 = 1,25$.

Teorema 1. *Para todo grafo* G *vale que*

$$\sum_{u \in V(G)} d_G(u) = 2|E(G)|. \tag{1.7}$$

Demonstração. Seja (V, E) em grafo e defina o conjunto $X = \{(u, e) \in V \times E : u \in e\}$ e vamos contar seu número de elementos de duas maneiras distintas.

Primeiro, cada vértice u participa de $d(u)$ elementos de X, portanto

$$|X| = \sum_{u \in V} d(u). \tag{1.8}$$

Depois, cada aresta e está presente em dois elementos de X, logo

$$|X| = 2|E|. \tag{1.9}$$

De (1.8) e (1.9) temos (1.7). □

Corolário 2. *Em todo grafo o número de vértices com grau ímpar é par.*

Demonstração. Seja G um grafo. Denote por I o subconjunto formado pelos vértices em $V(G)$ de grau ímpar e denote por P o subconjunto dos vértices de grau par. Usando que $I \cap P = \varnothing$ e que $I \cup P = V(G)$ podemos escrever $\sum_{u \in V} d_G(u) = \sum_{u \in I} d_G(u) + \sum_{v \in P} d_G(v)$, e do Teorema 1, temos

$$\underbrace{2|E(G)|}_{\text{par}} = \sum_{u \in I} d_G(u) + \underbrace{\sum_{v \in P} d_G(v)}_{\text{par}}, \tag{1.10}$$

portanto devemos ter $\sum_{u \in I} d(u)$ par, o que semente é possível quando $|I|$ é par. □

Exercícios

Exercício 1. Um químico deseja embarcar os produtos A, B, C, D, E, F, X usando o menor número de caixas. Alguns produtos não podem ser colocados numa mesma caixa porque reagem. Os produtos A, B, C, X reagem dois-a-dois; A reage com F e com D e vice-versa; E também reage com F e com D e vice-versa. Descreva o grafo que modela essa situação, mostre um diagrama desse grafo e use-o para descobrir o menor número de caixas necessárias para embarcar os produtos com segurança.

Exercício 2. Adaltina esperava as amigas Brandelina, Clodina, Dejaina e Edina para um lanche em sua casa. Enquanto esperava preparou os lanches: Bauru, Misto quente, Misto frio e X-salada. Brandelina gosta de Misto frio e de X-salada; Clodina de Bauru e X-salada; Dejaina gosta de Misto quente e Misto frio; Edina gosta de de Bauru e Misto quente. Descreva o grafo que modela essa situação, mostre um diagrama desse grafo e use-o para descobrir se é possível que cada amiga de Adaltina tenha o lanche que gosta. Se é possível, determine o número de soluções.

Exercício 3. Para todo $n \in \mathbb{N}$, qual é o número máximo de arestas que pode ter um grafo com n vértices?

Exercício 4. O **complemento** de um grafo G, denotado por $\overline{G}$, é o grafo que tem o mesmo conjunto de vértices de G e dois vértices formam uma aresta em $\overline{G}$ se e somente se **não** formam uma aresta de G:

$$\begin{aligned} V(\overline{G}) &= V(G), \\ E(\overline{G}) &= \binom{V(G)}{2} \setminus E(G) = \big\{\{u,v\} \subset V(G) \colon \{u,v\} \notin E(G)\big\}. \end{aligned}$$

Dê o complemento dos seguintes grafos

(i) G dado por

$$\begin{aligned} V(G) &= \{1,2,3,4,5\}, \\ E(G) &= \{\{1,2\},\{1,3\},\{1,4\},\{1,5\},\{2,3\},\{2,4\},\{2,5\},\{3,4\},\{3,5\},\{4,5\}\}. \end{aligned}$$

(ii) H dado por

$$\begin{aligned} V(H) &= \{1,2,3,4,5,6,7\}, \\ E(H) &= \{\{1,2\},\{1,3\},\{2,4\},\{2,5\},\{3,6\},\{3,7\}\}. \end{aligned}$$

(iii) B dado por

$$\begin{aligned} V(B) &= \{a,b,c,1,2,3\}, \\ E(B) &= \{\{a,1\},\{a,2\},\{a,3\},\{b,1\},\{b,2\},\{b,3\},\{c,1\},\{c,2\},\{c,3\}\}. \end{aligned}$$

Exercício 5. Defina a **união** dos grafos G e H por

$$G \cup H = \Big(V(G) \cup V(H), E(G) \cup E(H)\Big).$$

A união $G \cup H$ é um grafo?

Exercício 6. Defina a **intersecção** dos grafos G e H por $G \cap H = \Big(V(G) \cap V(H), E(G) \cap E(H)\Big)$. A intersecção $G \cap H$ é um grafo?

Exercício 7. Um grafo é chamado de **completo** sobre V se todo par de vértices de V é uma aresta do grafo, ou seja $E = \binom{V}{2}$. Um grafo completo com n vértices é denotado por $\mathbf{K^n}(V)$. Prove que

$$G \cup \overline{G} = K^n$$

onde $K^n = K^n(V(G))$.

Exercício 8. Considere o caso geral do exercício 1: Um químico deseja embarcar os produtos $p_1, p_2, \ldots, p_n$ usando o menor número de caixas. Alguns produtos não podem ser colocados numa mesma caixa porque reagem. Seja G o grafo que modela esse problema, onde vértices são produtos e arestas os pares que reagem, e denote por $\chi(G)$ o número de mínimo de caixas de modo que seja possível encaixotar os produtos com segurança. Prove que

$$\chi(G) \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$$

onde m é o número de pares de produtos que reagem. (Dica: Em uma distribuição mínima de caixas, a cada duas caixas, precisa existir pelo menos um produto em uma caixa reagindo com um produto da outra caixa. Assim podemos garantir um número mínimo de arestas para o grafo, m.)

Exercício 9. Chico e sua esposa foram a uma festa com três outros casais. No encontro deles houveram vários apertos de mão. Ninguém apertou a própria mão ou a mão da(o) esposa(o), e ninguém apertou a mão da mesma pessoa mais que uma vez.

Após os cumprimentos Chico perguntou para todos, inclusive para a esposa, quantas mãos cada um apertou e recebeu de cada pessoa uma resposta diferente. Quantas mãos Chico apertou?

Exercício 10. Prove que $\delta(G) \leq d(G) \leq \Delta(G)$ para todo grafo G.

Exercício 11. Decida se pode existir um grafo G com vértices que têm graus $2, 3, 3, 4, 4, 5$, respectivamente. E graus $2, 3, 4, 4, 5$? Se sim, descreva-os.

Exercício 12. Seja G um grafo com 14 vértices e 25 arestas. Se todo vértice de G tem grau 3 ou 5, quantos vértices de grau 3 o grafo G possui?

Exercício 13. Prove que em todo grafo de ordem pelo menos dois existem pelo menos dois vértices com o mesmo grau. (Dica: comece por um caso pequeno, por exemplo ordem 3, antes de tentar resolver o caso geral.)

Exercício 14. Para um número natural r, um grafo é r-**regular** se todos os vértices têm grau r. Para um grafo r-regular com n vértices e m arestas, expresse m em função de n e r.

Exercício 15. Dê exemplo de um grafo 3-regular que não é completo.

Exercício 16. Dado G, o **grafo linha** de G, denotado por LG, é o grafo cujos vértices são as arestas de G e um par de vértices define uma aresta em LG se, e somente se, esses vértices são arestas adjacentes em G. Dado G determine $|V(LG)|$ e $|E(LG)|$.

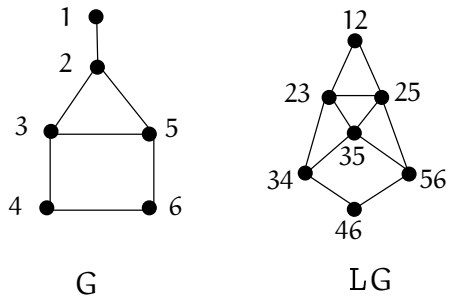


Figura 1.2: Exemplo de um grafo e o respectivo grafo linha.

Exercício 17. Prove que num grafo G com $\delta(G) > 0$ e $|E(G)| < |V(G)|$ existem pelo menos dois vértices de grau 1.

1.2 Subgrafos

Dizemos que o grafo H é um **subgrafo** do grafo G se, e somente se, $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$ e nesse caso escrevemos $H \subseteq G$ para indicar que H é subgrafo de G.

Exemplo 3. Considerando o grafo G do exemplo 1 temos que

$$\begin{aligned} G' &= (\{1,2,5\},\{\{1,5\},\{5,2\},\{1,2\}\}) \quad \text{e} \\ G'' &= (\{3,5,6\},\varnothing) \end{aligned}$$

são subgrafos de G, enquanto que

$$\begin{aligned} H &= (\{1,2,3\},\{\{1,2\},\{3,4\}\}), \\ I &= (\{1,2,3,4,9\},\{\{1,2\},\{3,4\}\}) \quad \text{e} \\ J &= (\{1,2,3,4,8\},\{\{1,2\},\{3,4\},\{1,8\}\}) \end{aligned}$$

não são subgrafos de G pois: H não é grafo, em I não vale $V(I) \subseteq V(G)$ e em J não vale $E(J) \subseteq E(G)$.

Dados um grafo G e um subconjunto de vértices $U \subseteq V(G)$, escrevemos $G[U]$ para o **subgrafo induzido por U** que é o subgrafo

$$G[U] = \left(U, E(G) \cap \binom{U}{2}\right).$$

Analogamente, definimos subgrafo induzido por um subconjunto de arestas. Se $M = \{e_1, e_2, \ldots, e_m\} \subseteq E(G)$, então o **subgrafo induzido por M**, denotado , tem como conjunto de vértices $e_1 \cup e_2 \cup \cdots \cup e_m$ e como conjunto de arestas o próprio M

$$G[M] = \Big(\bigcup_{i=1}^{m} e_i, M\Big).$$

Exemplo 4. Dos grafos G, H e I cujos diagramas são dados na figura 1.3, podemos dizer que H é um subgrafo induzido de G enquanto que I é um subgrafo mas não é induzido.

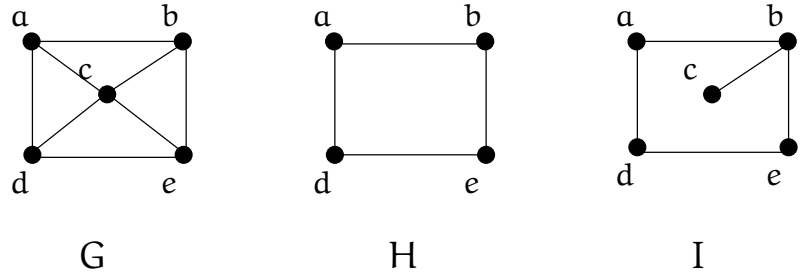


Figura 1.3: Diagrama dos grafos G, H e I.

Um subgrafo $H \subseteq G$ onde $V(H) = V(G)$ é chamado de **subgrafo gerador**. No exemplo acima I é subgrafo gerador de G, enquanto que H não é subgrafo gerador de G.

1.2.1 Clique e conjunto independente

Se o subconjunto $U \subseteq V(G)$ induz um subgrafo completo em G então chamamos U de **clique** em G. Mais especificamente, se $G[U]$ é um grafo completo com k vértices então dizemos que U é um **k-clique** em G.

O caso particular de um 3-clique num grafo G é chamado de *triângulo* de G.

Por outro lado, se $U \subseteq V(G)$ é tal que $G[U] = (U, \varnothing)$ é chamado de **conjunto independente** de G, ou **k-conjunto-independente** no caso $|U| = k$.

Exemplo 5. O subgrafo G' do exemplo 3 é um 3-clique e G'' do exemplo 3 é um 3-conjunto-independente.

No grafo G do exemplo 1 os conjuntos $\{3,5,6\}$ e $\{1,4,6,8\}$ são independentes; no caso de $\{1,4,6,8\}$ temos um conjunto independente de cardinalidade máxima pois não há naquele grafo conjunto independente com 5 ou mais vértices. Nesse mesmo grafo, $\{8\}$, $\{6,7\}$ e $\{1,2,5\}$ são cliques, o último de cardinalidade máxima.

Observação 2. O tamanho do maior clique e o tamanho do maior conjunto independente num grafo G são difíceis de serem calculados computacionalmente. Eles pertencem a classe dos problemas NP-difíceis (veja [8], página 53). Uma conseqüência desse fato é que não é sabido se existem algoritmos cujo tempo de execução no pior caso é um polinômio em $|V(G)|+|E(G)|$ para resolver esse problemas. A descoberta de um algoritmo com tempo de pior caso polinomial no tamanho de G, ou a prova de que ele não existe, é um dos problemas não-resolvidos mais importantes da atualidade, o problema P $\times$ NP. Trata-se de um dos sete problemas do milênio [9], dos quais restam seis não resolvidos, cada um com uma recompensa de US$1.000.000,00 para uma solução, paga pelo *Clay Mathematics Institute*.

1.2.2 Grafo bipartido e corte

Chamamos um grafo G de **grafo bipartido** se existem dois conjuntos independentes A e B em G que *particionam* V(G), isto é, A e B são tais que $A \cap B = \varnothing$ e $A \cup B = V(G)$. Por exemplo, o seguinte grafo é bipartido

$$\begin{aligned} V(G) &= \{1,2,3,4,5,6,7,8\} \\ E(G) &= \big\{\{1,6\},\{6,2\},\{3,7\},\{3,8\},\{7,4\},\{7,5\}\big\}, \end{aligned}$$

pois $V(G) = \{1,2,3,4,5\} \cup \{6,7,8\}$ e tanto $\{1,2,3,4,5\}$ quanto $\{6,7,8\}$ são conjuntos independentes.

Notemos que a bipartição pode não ser única, no caso do exemplo acima podemos escrever $V(G) = \{6,3,4,5\} \cup \{1,2,7,8\}$ e tanto $\{6,3,4,5\}$ quanto $\{1,2,7,8\}$ são conjuntos independentes. Para evitar ambigüidades escrevemos um grafo bipartido G com bipartição $\{A,B\}$ como $\mathbf{G} = (\mathbf{A} \cup \mathbf{B}, \mathbf{E})$.

Sejam G um grafo, A e $B \subset V(G)$ dois subconjuntos disjuntos em V(G). Definimos o subconjunto de arestas

$$E(A,B) = \Big\{\{u,v\} \in E(G) \colon u \in A \text{ e } v \in B\Big\}; \tag{1.11}$$

e o **subgrafo bipartido induzido** por A e B é o grafo bipartido

$$\Big(A \cup B, E(A,B)\Big).$$

Observação 3. Convencionamos que os grafos triviais e vazio são grafos bipartidos.

Exemplo 6. A figura abaixo mostra as arestas de $E(A,B)$ para $A = \{0,1,8\}$ e $B = \{3,4,5,6\}$.

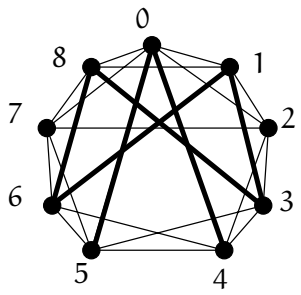


Figura 1.4: $E\big(\{0,1,8\},\{3,4,5,6\}\big)$ é formado pelas arestas $\big\{\{0,4\},\{0,5\},\{1,3\},\{1,6\},\{8,3\},\{6,8\}\big\}$.

O conjunto de arestas $E(A,\overline{A})$ é chamado de **corte definido por** A . Da definição de corte podemos escrever

$$E(G) = E(G[A]) \cup E(G[\overline{A}]) \cup E(A,\overline{A}). \tag{1.12}$$

Exemplo 7. A figura abaixo mostra as arestas de $E(A,B)$ para $A = \{0,1,8\}$ e $B = \{3,4,5,6\}$.

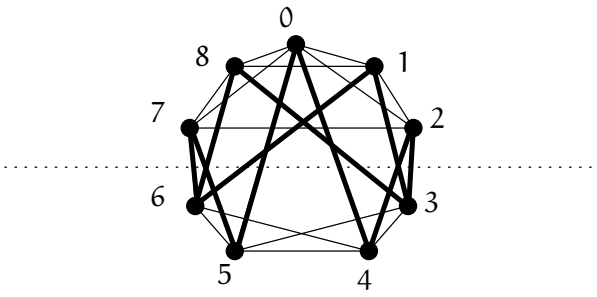


Figura 1.5: O corte definido pelo conjunto $\{0,1,2,7,8\}$ é formado pelas arestas $\big\{\{0,4\},\{0,5\},\{1,3\},\{1,6\},\{8,3\},\{6,8\},\{5,7\},\{6,7\},\{2,3\},\{2,4\}\big\}$.

1.2.3 Teorema de Mantel

Suponha que $G = (V, E)$ é um grafo que não contenha triângulo. Vamos determinar o número máximo de arestas que pode haver em G.

Seja A um conjunto independente em G de cardinalidade máxima. Como G não contém triângulos a vizinhança de qualquer vértice é um conjunto independente, portanto temos

$$d(v) \leq |A|, \quad \text{para todo } v \in V. \tag{1.13}$$

Como A é um conjunto independente em G podemos classificar as arestas de $E(G)$ em dois tipos: E_1 são as arestas de G que têm exatamente um dos extremos fora de A e E_2 são as arestas de G que tem ambos extremos fora de A. Dessa forma, o número de arestas em E é $|E_1| + |E_2|$ e

$$\sum_{u \in \overline{A}} d(u) = |E_1| + 2|E_2| \geq |E|.$$

Usando (1.13) chegamos a

$$\sum_{u \in \overline{A}} d(u) \leq \sum_{u \in \overline{A}} |A| = |A||\overline{A}|,$$

portanto $|E| \leq |A||\overline{A}|$. Usando a desigualdade entre as médias aritmética e geométrica[1]

$$|A||\overline{A}| = |A|(|V| \setminus |A|) \leq \frac{|V|^2}{4}. \tag{1.14}$$

Assim, provamos o seguinte resultado que foi mostrado pela primeira vez por Mantel em 1906.

Teorema (Mantel, 1906)**.** *Se G é um grafo sem triângulos então* $|E(G)| \leq |V(G)|^2/4$. □

Esse teorema é um caso particular do famoso Teorema de Turán, que foi o princípio de um ramo da teoria dos grafos chamada de Teoria Extremal de Grafos (veja mais sobre esse assunto em [2]).

Exercícios

Exercício 18. Quantos subgrafos tem o grafo $\big(\{1,2,3,4,5,6\},\{\{1,2\}\}\big)$?

Exercício 19. Quantos subgrafos completos tem o grafo completo de ordem n?

Exercício 20. Sejam G um grafo e $M \subseteq E(G)$. Tome o subconjunto $U = \bigcup_{e \in M} e$ de vértices de G. Prove ou dê um contra-exemplo para $G[U] = G[M]$.

[1] Desigualdade: $(a_1 a_2 \cdots a_n)^{\frac{1}{n}} \leq \frac{a_1 + a_2 + \cdots + a_n}{n}$. O caso $n = 2$ é simples e pode ser derivado do fato de que $(a-b)^2 \geq 0$.

Exercício 21. Descubra um subgrafo induzido de

$$\begin{aligned} V(G) &= \{1,2,3,4,5,6,7,8\} \text{ e} \\ E(G) &= \big\{\{1,2\},\{1,3\},\{2,3\},\{2,5\},\{3,6\},\{8,5\},\{8,6\},\{5,6\},\{3,4\},\{5,7\}\big\} \end{aligned}$$

1-regular e com o maior número possível de arestas. (Qual a relação com a resolução do exercício 2?)

Exercício 22. Mostre que em *qualquer* grafo G com pelo menos 6 vértices vale: ou G tem um 3-clique e $\overline{G}$ tem um 3-conjunto-independente, ou G tem um 3-conjunto-independente e $\overline{G}$ tem um 3-clique. (Dica: exercício 7 e princípio da casa dos pombos sobre $E_{K^6}(v)$, para algum vértice v.)

Exercício 23. Dado um grafo G, denotamos por $\alpha(\mathbf{G})$ a cardinalidade do maior conjunto independente em G,

$$\alpha(G) = \max\big\{|A|\colon A \subset V(G) \text{ é um conjunto independente}\big\}.$$

Prove que se $d(G) > \alpha(G)$ então G contém triângulo, para todo G.

Exercício 24. Para todo grafo G, denotamos por $\omega(\mathbf{G})$ a cardinalidade do maior clique em G

$$\omega(G) = \max\big\{|A|\colon A \subset V(G) \text{ é um clique}\big\}.$$

Prove que $\omega(G) = \alpha(\overline{G})$.

Exercício 25. Demonstre que as desigualdades abaixo valem para todo grafo G

(i) $\alpha(G) \geq |V(G)|/(\Delta(G)+1)$;

(ii) $\alpha(G) \leq |E(G)|/\delta(G)$, se $\delta(G) \neq 0$;

(iii) $\omega(G) \leq \Delta(G)+1$.

Exercício 26. Suponha $H \subseteq G$. Prove ou refute as desigualdades:

(i) $\alpha(H) \leq \alpha(G)$;

(ii) $\alpha(G) \leq \alpha(H)$;

(iii) $\omega(G) \leq \omega(H)$;

(iv) $\omega(H) \leq \omega(G)$.

Exercício 27. Seja G um grafo bipartido. Prove que todo subgrafo de G é bipartido.

Exercício 28. Seja $G = (A \cup B, E)$ um grafo bipartido qualquer e suponha que $|A| < |B|$. É verdade que $\alpha(G) = |B|$? Determine $\omega(G)$.

Exercício 29. Um grafo bipartido G com *partes* A e B é dito **completo** se

$$E(G) = \big\{\{a,b\} \subseteq V(G)\colon a \in A \text{ e } b \in B\big\}.$$

Um grafo bipartido completo sobre $\{A,B\}$ com partes de cardinalidade $|A| = n$ e $|B| = m$ é denotado por $\mathbf{K^{n,m}(A,B)}$. Determine $|E(K^{n,m}(A,B))|$.

Exercício 30. Prove que todo grafo G tem um subgrafo bipartido H com $|E(H)| \geq |E(G)|/2$.

Exercício 31. Prove que todo grafo G tem um subgrafo gerador bipartido H tal que $d_H(v) \geq d_G(v)/2$ para todo $v \in V(G)$.

Exercício 32. Prove a afirmação da equação (1.12).

Exercício 33. Dado um grafo G, defina para todo $U \subseteq V(G)$ a **vizinhança de** U, denotada $N_G(U)$, por

$$N_G(U) = \bigcup_{u \in U} N_G(u).$$

É verdade que $|E(U,\overline{U})| = |N_G(U)|$? Justifique.

Exercício 34. Prove que para quaisquer $U, W \subseteq V(G)$, onde G é um grafo, vale que $E(U \triangle W, \overline{U \triangle W}) = E(U, \overline{U}) \triangle E(W, \overline{W})$.

Exercício 35. Dado $F \subset E(G)$ mostre que se $|E(C) \cap F|$ é par para todo circuito C em G então F é um corte.

Exercício 36. Prove que para todo G e todo $U \subset V(G)$

$$\sum_{u \in U} d_G(u) = 2|E(G[U])| + |E(U, \overline{U})|.$$

Exercício 37. Um grafo G é dito **k-partido**, para $k \in \mathbb{N}$, se existem k conjuntos independentes A_1, A_2, ..., A_k que particionam $V(G)$, ou seja, $V(G) = A_1 \cup A_2 \cup \cdots \cup A_k$, o conjunto A_i é um conjunto independente em G para todo $i \in \{1, 2, \ldots, k\}$ e $A_i \cap A_j = \varnothing$ para quaisquer i e j distintos. Prove que dentre os grafos k-partidos ($k \geq 2$) completos com n vértices o número máximo de arestas é atingido quando $|A_i| - |A_j| \leq 1$ para todos $i, j \in \{1, 2, \ldots, n\}$ distintos.

Exercício 38. Mostre que, se $n = kq + r$ com $0 \leq r < k$, então o número de arestas do grafo do exercício anterior é

$$\frac{1}{2}\left(\frac{k-1}{k}\right)(n^2 - r^2) + \binom{r}{2}$$

e que esse número é limitado por

$$\leq \frac{k-1}{k}\binom{n}{2}.$$

Exercício 39 (Teorema de Turán, 1941). Para todo $k \geq 2$, se G tem n vértices e não contém um k-clique então

$$|E(G)| \leq \frac{k-2}{k-1}\frac{n^2}{2}.$$

Exercício 40. Redefina para todo grafo G o parâmetro $\chi(G)$ dado no exercício 8 em função dos conjuntos independentes de G. Esse parâmetro de um grafo é conhecido na literatura como número cromático[2] do grafo (veja [5], capítulo 5).

Exercício 41. Prove que G é bipartido se e somente se $\chi(G) < 3$.

Exercício 42. Prove que as duas desigualdades dadas a seguir valem para todo grafo G com pelo menos um vértice

$$\omega(G) \leq \chi(G) \tag{1.15}$$

$$\chi(G) \geq \frac{|V(G)|}{\alpha(G)}. \tag{1.16}$$

Exercício 43. Prove que todo grafo G satisfaz

$$\chi(G) \leq 1 + \max_{H \subseteq G} \delta(H).$$

1.3 Isomorfismo

Dizemos que os grafos G e H são **isomorfos** e, nesse caso escrevemos $G \simeq H$, se existe uma função bijetora

$$f\colon V(G) \to V(H) \tag{1.17}$$

tal que

$$\{u, v\} \in E(G) \iff \{f(u), f(v)\} \in E(H) \tag{1.18}$$

para todos $u, v \in V(G)$. Uma função f como acima é chamada de **isomorfismo**.

Exemplo 8 (Grafo de Petersen). Os grafos representados na figura 1.6 são isomorfos pelo isomorfismo $f(1) = a$, $f(2) = b$, $f(3) = c$, $f(4) = d$, $f(5) = e$, $f(6) = f$, $f(7) = g$, $f(8) = h$, $f(9) = i$, $f(10) = j$. Esse grafo é chamado de **grafo de Petersen**, é um dos grafos mais conhecidos na Teoria dos Grafos.

[2] Computar o número cromático é um problema NP-difícil [8].

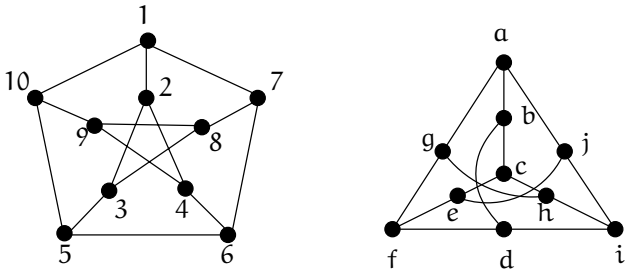


Figura 1.6: Grafos isomorfos (grafo de Petersen).

Notamos que quaisquer dois grafos completos G e H de mesma ordem são isomorfos. Mais que isso, qualquer bijeção entre V(G) e V(H) define um isomorfismo entre eles. Nesse caso, dizemos que o grafo é *único a menos de isomorfismos* e por isso usamos a mesma notação para todos eles, a saber K^n, quando o conjunto dos vértices não é relevante.

Exemplo 9. Há oito grafos distintos com três vértices, eles estão descritos nas representações da figura 1.7 abaixo.

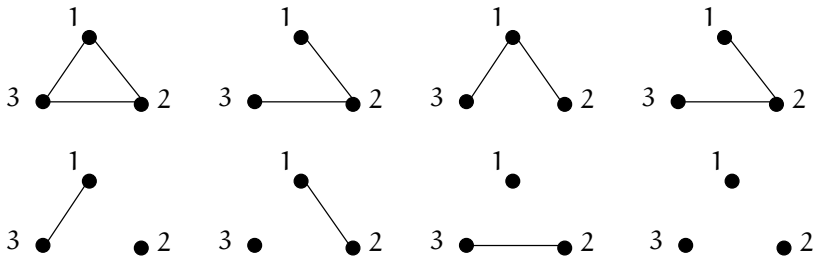


Figura 1.7: Grafos distintos de ordem 3.

No entanto, há apenas 4 grafos não-isomorfos com três vértices, representados pelos diagramas da figura 1.8

Figura 1.8: Grafos não-isomorfos de ordem 3.

Não existe uma caracterização simples de grafos isomorfos. Isso significa que não há algoritmo eficiente que recebe dois grafos e decide se eles são isomorfos.

Exemplo 10. Nenhum dos grafos G, H e K representados na figura 1.9 são isomorfos.

Temos que G não é isomorfo a H porque G não tem um vértice de grau quatro enquanto que o vértice 5 em H tem grau quatro, portanto não há como haver uma bijeção entre os vértices desse grafo que preserve as adjacências. Pelo mesmo motivo H não é isomorfo a K. Agora, G não é isomorfo a K porque caso existisse um isomorfismo $f\colon V(G) \to V(K)$ então a imagem por f do conjunto $\{2,3,5\} \subset V(G)$ é, obrigatoriamente, o conjunto $\{2,3,5\} \subset V(K)$, mas qualquer bijeção f não preserva adjacência entre esses vértices pois $\{2,3,5\}$ em G induz um triângulo e em K não (veja o exercício 46 abaixo).

Nesse exemplo foram dados argumentos diferentes para concluir o mesmo fato, o *não-isomorfismo* entre pares de grafos. Ainda, existem exemplos de grafos não isomorfos para os quais esses argumentos não funcionam (da mesma forma que a existência de um vértice de grau quatro funciona para mostrar que G não é isomorfo a H mas não serve para mostrar que G não é isomorfo a K pois $1,2,2,3,3,3$ são os graus dos vértices de ambos os grafos).

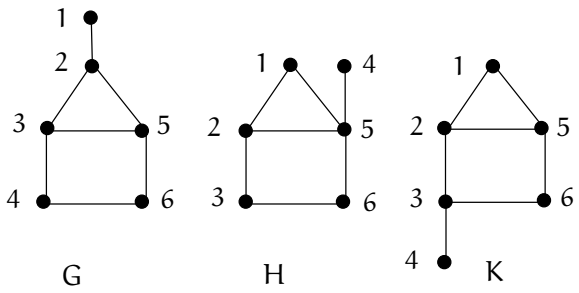


Figura 1.9: Grafos não-isomorfos.

Observação 4. É difícil caracterizar de modo eficiente o não-isomorfismo entre grafos:

- *O problema do não-isomorfismo de grafos*: Dados os grafos $G = (V, E)$ e $H = (V, E')$ decidir se eles são não-isomorfos.

Não se conhece algoritmo de tempo polinomial no tamanho dos grafos para decidir se dois grafos não são isomorfos. Mais do que isso, não se conhece um algoritmo de tempo polinomial que receba como entrada uma terna (G, H, P) onde P é uma prova de que G e H não são isomorfos e que devolva *sim* se G_1 não é isomorfo a G_2 e devolva *não* caso contrário. Em linguagem técnica dissemos que *não se sabe se o problema do não-isomorfismo de grafos está na classe* NP *de complexidade computacional.*

Observação 5. Por outro lado, podemos considerar o problema do isomorfismo de grafos:

- *O problema do isomorfismo de grafos*: Dados os grafos $G = (V, E)$ e $H = (V, E')$ decidir se eles são isomorfos.

Atualmente não se conhece algoritmo polinomial no tamanho do grafo que resolva o problema. Entretanto, não é difícil projetar um algoritmo de tempo polinomial que recebe a terna (G, H, f) onde $f\colon V(G) \to V(H)$ e devolve *sim* caso G e H são isomorfos e f é o isomorfismo, caso contrário devolve *não*. Em linguagem técnica dizemos que *o problema do isomorfismo de grafos está na classe* NP *de complexidade de problemas computacionais.* Entretanto, não é sabido se esse problema é NP-completo.

Exercícios

Exercício 44. Determine quais pares dentre os grafos abaixo são isomorfos.

(i) G_1 dado por $V(G_1) = \{v_1, u_1, w_1, x_1, y_1, z_1\}$ e
$E(G_1) = \{\{u_1, v_1\}, \{u_1, w_1\}, \{v_1, w_1\}, \{v_1, x_1\}, \{w_1, y_1\}, \{x_1, y_1\}, \{x_1, z_1\}\}$;

(ii) G_2 dado por $V(G_2) = \{v_2, u_2, w_2, x_2, y_2, z_2\}$ e
$E(G_2) = \{\{u_2, v_2\}, \{u_2, w_2\}, \{v_2, w_2\}, \{v_2, x_2\}, \{w_2, y_2\}, \{x_2, y_2\}, \{y_2, z_2\}\}$;

(iii) G_3 dado por $V(G_3) = \{v_3, u_3, w_3, x_3, y_3, z_3\}$ e
$E(G_3) = \{\{u_3, v_3\}, \{u_3, w_3\}, \{v_3, w_3\}, \{v_3, x_3\}, \{w_3, y_3\}, \{x_3, y_3\}, \{u_3, z_3\}\}$.

Exercício 45. Mostre que existem 11 grafos não-isomorfos com 4 vértices.

Exercício 46. Sejam G e H grafos isomorfos e $f\colon V(G) \to V(H)$ um isomorfismo. É verdade que $G[U]$ é isomorfo a $H[f(U)]$ para todo $U \subseteq V(G)$? Justifique.

Exercício 47. Mostre que o grafo de Petersen é isomorfo ao complemento do grafo linha do K^5.

Exercício 48. Um **automorfismo** de um grafo é um isomorfismo do grafo sobre ele mesmo. Quantos automorfismos tem um grafo completo?

Exercício 49. Mostre que o conjunto de automorfismos de um grafo com a operação de composição de funções definem um grupo.

Exercício 50. Qual o número de grafos distintos sobre um conjunto de vértices V de tamanho n?

Exercício 51. Prove que há pelo menos $2^{\binom{n}{2}}(n!)^{-1}$ grafos não isomorfos sobre um conjunto de vértices de ordem n.

Exercício 52. Um grafo $G = (V, E)$ é **vértice-transitivo** se para quaisquer $u, v \in V$ existe um automorfismo f de G com $f(v) = u$. Analogamente, G é **aresta-transitivo** se para quaisquer arestas $\{x, y\}, \{z, w\} \in E$ existe um automorfismo f de G tal que $\{f(x), f(y)\} = \{z, w\}$.

Dê um exemplo de grafo vértice-transitivo. Dê um exemplo de grafo aresta-transitivo. Dê um exemplo de grafo aresta-transitivo mas não vértice-transitivo.

1.4 Outras noções de grafos

Em algumas situações podemos ter um modelo para um problema a ser resolvido e esse modelo seria um grafo se desconsiderássemos algumas peculiaridades da situação. Por exemplo, um mapa rodoviário pode ser modelado definindo-se um vértice para cada cidade e duas cidades formam uma aresta no grafo (modelo) se existe rodovia ligando essas cidades correspondentes aos vértices. Normalmente, distância é um parâmetro importante nesses mapas e assim as arestas devem ter um comprimento associado a elas. Entretanto, "comprimento de aresta" não faz parte da definição de um grafo. Num outro exemplo, se estamos interessados em rotas de tráfego dentro de uma cidade podemos definir um vértice por esquina e duas esquinas consecutivas numa mesma rua formam uma aresta. Nesse caso, as ruas têm sentido (mão e contra-mão) e as arestas também deveriam ter mas, novamente, essa característica não faz parte da definição de grafos.

Esses problemas e muitos outros podem ser modelados com "outros tipos" de grafos. Alguns desses outros tipos são

Grafo com pesos nas arestas é um tripla (V, E, ρ) de modo que (V, E) é um grafo e $\rho\colon E \to R$ é uma função que assume valores em $R \subseteq \mathbb{R}$.

Grafo orientado têm orientação nas arestas (são pares ordenados) de modo que para vértices u e v, se (u, v) é uma aresta então (v, u) não é aresta.

Grafos dirigido ou digrafo é dado por um par (V, E) onde $E \subseteq V \times V \setminus \{(v, v)\colon v \in V\}$.

Multigrafo é dado por um conjunto V de vértices, um conjunto E de arestas e uma função de E em $\binom{V}{2}$ que diz quem são os extremos de cada aresta; nesse caso podemos ter mais de uma aresta com os mesmos extremos.

Em cada um dos casos acima, o grafo (V, E) naturalmente definido por essas estruturas é chamado **grafo subjacente** .

Outros tipos podem ser derivados combinando esses tipos de grafos, por exemplo, grafo dirigido com pesos nas arestas.

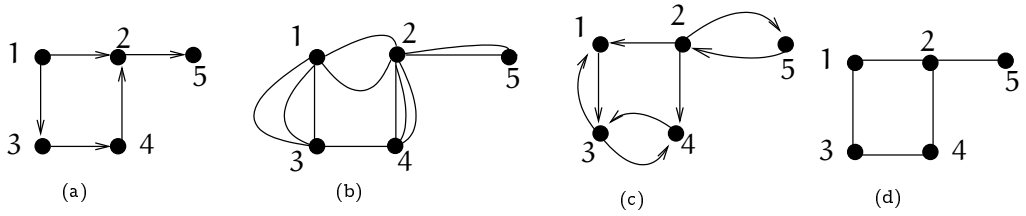


Figura 1.10: (a) Grafo orientado. (b) Multigrafo. (c) Grafo dirigido. (d) Grafo subjacente aos grafos dados em (a), (b) e (c).

Exercícios

Exercício 53. Defina isomorfismo para grafos orientados.

Exercício 54. Formule uma versão do teorema 1 para grafos orientados.

1.5 Representação computacional

O primeiro passo para representar computacionalmente um grafo é mapear o conjunto de vértices do grafo G no subconjunto dos número naturais $\{1, 2, \ldots, |V(G)|\}$ para facilitar o acesso as informações usando estruturas indexadas, ou seja, dado G o problema é construir um grafo isomorfo a G com vértices no conjuntos dos números naturais. Assim, se G é uma entrada para um problema computacional, então G deve ser "traduzido" para sua cópia isomorfa, o problema é tratado e a solução do problema sobre a cópia deve ser traduzida para uma solução para G.

Isso pode ser feito usando as técnicas de busca conhecidas para recuperar os inteiros associados aos vértices como, por exemplo, tabela de espalhamento (*hashing*) ou árvore binária de busca. Feito isso, qualquer problema computacional sobre um grafo G é tratado computacionalmente sobre um grafo isomorfo G' com vértices $V(G') = \{1, 2, \ldots, |V(G)|\}$ e a estrutura de busca escolhida no passo anterior é usada para representar o isomorfismo.

No que segue, sempre que tratamos problemas computacionais sobre grafos estamos assumindo os grafos são definidos sobre o conjunto de vértices $\{1, 2, \ldots, n\}$, para algum $n \in \mathbb{N}$. Veremos duas das estruturas de dados mais conhecidas para a representação de grafos.

1.5.1 Listas de adjacências de G

É um vetor N[ ] de listas ligadas. A lista N[i] contém os vizinhos do vértice i e cada nó dessa lista é composto por uma variável que armazena um vértice e um ponteiro que aponta para o próximo nó da lista.

Exemplo 11. Para o G grafo representado na figura 1.9, uma lista de adjacências é como segue

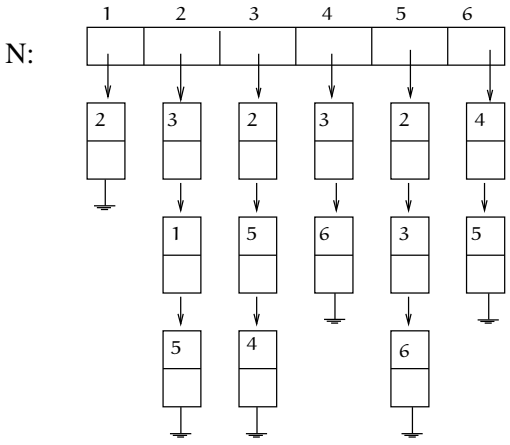


Figura 1.11: Lista de adjacências do grafo G na figura 1.9.

Observação 6. A representação de uma grafo por listas de adjacências não é única, pois qualquer permutação dos nós em N[i] define uma lista válida.

1.5.2 Matriz de adjacências de G

É a matriz $|V| \times |V|$ denotada por $A(G)$, ou A simplesmente, definida por

$$A(i,j) = \begin{cases} 1, & \text{se } \{i,j\} \in E(G) \\ 0, & \text{caso contrário.} \end{cases}$$

Exemplo 12. Para o G grafo representado na figura 1.9, a matriz de adjacências é

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

1.5.3 Complexidade de algoritmos em grafos

Neste texto entendemos por complexidade do algoritmo a

complexidade de tempo no pior caso

expressa em função do tamanho da representação do grafo.

Exemplo 13. Sejam A e B dois algoritmos distintos que executam a mesma tarefa sobre um grafo $G = (V, E)$. O algoritmo A gasta, no pior caso, $|V|^2$ passos para executar a tarefa e o algoritmo B gasta $(|V| + |E|) \log |E|$ passos no pior caso. Para grafos com $|V|^2/4$ arestas o primeiro algoritmo é sempre melhor enquanto que para grafos com $1.000|V|$ arestas o segundo algoritmo é assintoticamente melhor (melhor para $|V| \geq 16.645$, enquanto que o primeiro é melhor para $|V| \leq 16.644$). Para entradas pequenas (até 25 mil vértices) a ordem das funções são comparadas nos gráficos da figura 1.12.

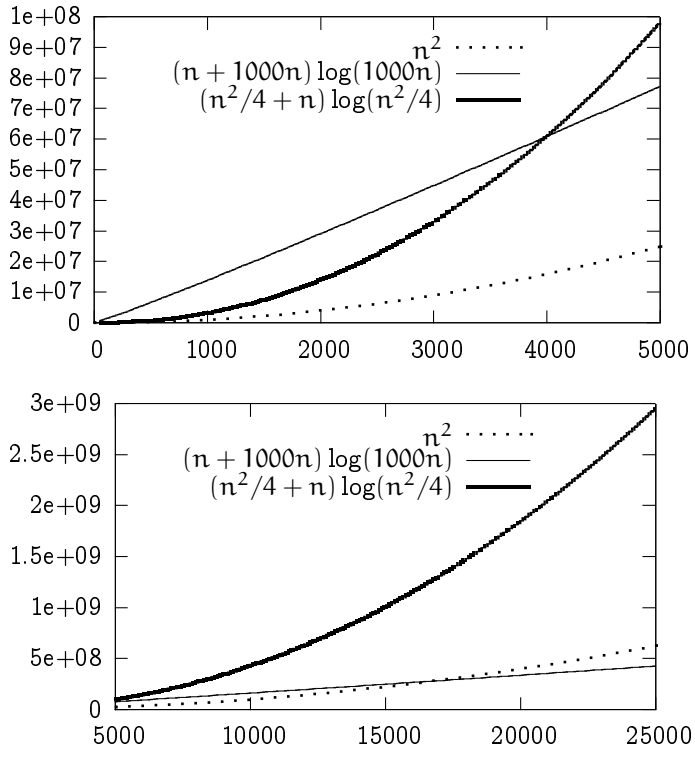


Figura 1.12: Gráfico das funções do exemplo 13.

Usualmente, a expressão que estabelece a complexidade de um algoritmo é escrita em função do tamanho da representação usada e em notação assintótica: sejam f, g duas funções definidas nos naturais, então

$$f(n) = O(g(n)), \quad \text{para } n \text{ auficientemente grande}$$

significa que existem constantes positivas c e n_0 tais que **para todo** $\mathbf{n \geq n_0}$ vale

$$f(n) \leq c \cdot g(n).$$

No nosso caso n é o *tamanho da representação* do grafo.

Vejamos a complexidade de alguns algoritmos que executam tarefas simples comparando cada uma das representações dadas acima para um grafo fixo $G = (V, E)$

Tarefa	Matriz de adjacências	Lista de adjacências
$\{i, j\} \in E$?	$O(1)$	$O(\|V\|)$
Determine $d(i)$	$O(\|V\|)$	$O(\|V\|)$
Devolva E	$O(\|V\|^2)$	$O(\|E\|)$

Observação 7. Um algoritmo em grafo é de complexidade polinomial se a complexidade de tempo no pior caso é expressa por uma função polinomial no tamanho da representação. Complexidade polinomial é independente da representação do grafo por matriz ou lista de adjacências, ou seja, essas representações são *polinomialmente relacionadas*.

Exercícios

Exercício 55. Mostre que se $f(n) = O(g(n))$ então $O(f(n)) + O(g(n)) = O(g(n))$.

Exercício 56. É verdade que para todo G^n vale que $|E(G)| = O(n)$ para n suficientemente grande?

Exercício 57. É verdade que para todo G^n vale que $\log |E(G)| = O(\log n)$ para n suficientemente grande?

Exercício 58. Sejam G um grafo sobre o conjunto de vértices $\{1, 2, \ldots, n\}$, A a sua matriz de adjacências e $b(i, j)$ as entradas da matriz A^2. Que parâmetro de G está associado ao número $b(i, i)$, para cada $i \in V(G)$? Qual a relação entre $b(i, j)$ e, $N_G(i)$ e $N_G(j)$ quando $i \neq j$?

Exercício 59. Seja G um grafo qualquer sobre o conjunto de vértices $\{1, 2, \ldots, n\}$ e A sua matriz de adjacências. Como é possível determinar o número de triângulos de um grafo G qualquer a partir de A^3?

Exercício 60. Em 1990, Coppersmith e Winograd [3] mostraram um algoritmo que determina o produto de matrizes duas $n \times n$ usando $O(n^{2,376})$ operações aritméticas. Como esse algoritmo pode ser usado para determinar se um grafo tem triângulo?

Exercício 61. Sejam G um grafo sobre o conjunto de vértices $\{1, 2, \ldots, n\}$ e A sua matriz de adjacências. Prove que se os autovalores de A são distintos então o grupo dos automorfismos é abeliano.

Exercício 62. O **traço** de uma matriz A, denotado por $tr(A)$, é a soma dos elementos na diagonal da matriz. Quanto vale o traço de uma matriz de adjacências? Quanto vale $tr(A^2)$ em função de $E(G)$?

Exercício 63. Por A ser uma matriz simétrica, sabemos da Álgebra Linear que todos os seus autovalores são números reais. Determine os autovalores da matriz de adjacências do grafo completo K^n sobre o conjunto de vértices $\{1, 2, \ldots, n\}$. Mostre que se G é um grafo r-regular sobre o conjunto de vértices $\{1, 2, \ldots, n\}$, então r é uma autovalor de A.

Exercício 64. Dada uma representação por listas de adjacências de um grafo dirigido D descreva um algoritmo com tempo de execução $O(|V(D)| + |E(D)|)$ para computar a representação por listas de adjacências do grafo subjacente.

Exercício 65. Neste exercício apresentamos duas estruturas de dados para representar grafos dirigidos. Dizemos que a aresta $(u, v) \in E$ num grafo dirigido (V, E) *sai de* u e *chega em* v.

Os **vetores de incidências** de um grafo dirigido $D = (V, E)$ são os vetores S e C indexados por E tais que para $e = (u, v) \in E$ temos $S[e] = u$ e $C[e] = v$, e nesse caso dizemos que e *sai* de u e *chega* em v.

Uma **matriz de incidências** de um grafo dirigido $D = (V, E)$ é uma matriz B de dimensão $|V| \times |E|$ (as linhas são indexadas por V e as colunas por E) tal que

$$B(i, j) = \begin{cases} -1 & \text{se a aresta } j \text{ sai do vértice } i, \\ 1 & \text{se a aresta } j \text{ chega no vértice } i, \\ 0 & \text{caso contrário.} \end{cases}$$

Para cada uma das representações, determine o tempo para:

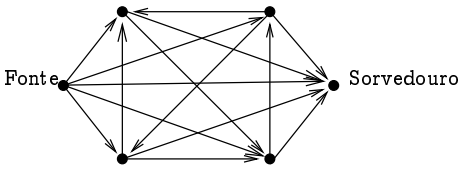
(i) dados $e \in E$ e $v \in V$, determinar se e sai ou chega em v;

(ii) dado $e \in E$, determinar suas extremidades;

(iii) dado $v \in V$ determinar as arestas que saem de v e determinar as arestas que chegam em v.

Determine o que as entradas da matriz $|V| \times |V|$ dada por $B \cdot B^T$ representam, onde B^T é a matriz transposta de B.

Exercício 66 ([4]). Mostre que determinar se um grafo orientado contém um *sorvedouro* — isto é, um vértice com grau de entrada $|V| - 1$ e grau de saída 0 — pode ser feito em tempo $O(|V|)$ quando uma representação por matriz é utilizada.



1.6 Percursos em grafos

Nessa seção veremos algoritmos que formam a base para muitos algoritmos em grafos (para quase todos deste texto). O objetivo de um percurso num grafo é o seguinte: dado a representação de um grafo $G = (V, E)$, sistematicamente visitar todos os vértices e todas as arestas desse grafo.

Dado um vértice w em V, dizemos que o vértice u do grafo é *alcançável* por w em G se existe uma seqüência de vértices $v_1, v_2, \ldots, v_m$ de G tal que

- $w = v_1$,
- $u = v_m$,
- v_i e v_{i+1} são adjacentes, para todo $i \in \{1, 2, \ldots, m-1\}$.

Definimos que w é alcançável por w em G, para todo $w \in V$.

Uma seqüência de vértices como acima é chamada de **passeio** em G.

1.6.1 Percurso genérico

Um percurso num grafo G começa com as seguintes considerações: cada vértice está em um de dois estados possíveis em qualquer instante da execução: *não-visitado* ou *visitado*; cada aresta em $E(u)$ está em um de dois estados: *não-visitada a partir de* u ou *visitada a partir de* u, para todo $u \in V(G)$. De início todos os vértices de G estão *não-visitado* e todas as arestas *não-visitada* a partir de ambos os extremos.

O seguinte algoritmo, que chamaremos de Visite(G, v), recebe um grafo G e um vértice inicial $v \in V(G)$ e visita uma única vez todos os vértices alcançáveis a partir de v e visita duas vezes todas as arestas induzidas por esses vértices, uma visita a partir de cada extremo. Para gerenciar essa visita o algoritmo Visite(G, v) usa uma lista L na qual a busca, inserção e remoção de elementos é feita de maneira *arbitrária* e com custo constante, também o teste "$L = \varnothing$?" é feito em tempo constante.

Algoritmo 1: Visite(G, v).

Dado : um grafo G e um vértice v.
Devolve: visita aos vértices e arestas não-visitados e alcançáveis a partir de v.

```
1  insira v em L;
2  marque v visitado;
3  enquanto L ≠ ∅ faça
4      escolha u ∈ L;
5      se em E_G(u) há aresta não-visitada a partir de u então
6          escolha {u, w} em E_G(u) não-visitada a partir de u;
7          marque {u, w} visitada a partir de u;
8          se w é não-visitado então
9              insira w em L;
10             marque w visitado;
11     senão remova u de L.
```

Análise e correção do algoritmo *Visite*(G, v). No que segue faremos uma análise detalhada do algoritmo Visite(G, v). Fixe um grafo G representado por uma lista de adjacências e um vértice v de G, suponha que todos os vértices e arestas são não-visitados, *considere uma execução de* Visite(G, v) e defina

$$A = \{u \in V(G) : u \text{ é alcançável a partir de } v\}.$$

Proposição 3. *Cada vértice de* G *entra em* L *no máximo uma vez.*

Demonstração. Um vértice w é inserido em L na linha 1 ou 9. A condição para essas linhas serem executadas é que w seja *não-visitado* e após a inserção o vértice w é marcado *visitado*, nas linhas 2, 10. Logo a condição para inserção de w em L passa a ser falsa. A proposição segue do fato de que um vértice *visitado* nunca se tornará *não-visitado* durante uma execução de Visite(G, v). □

Proposição 4. *A linha 4 é executada no máximo* $d_G(w) + 1$ *vezes, para cada* $w \in V(G)$.

Demonstração. Fixe um vértice w e assuma que $w \in L$. Para cada execução da linha 4 com $u = w$, ou vale a condição da linha 5 ou é executada a linha 11.

Se vale a condição da linha 5 então o número de arestas *não-visitadas a partir de* w em $E_G(w)$ diminui de um, se não vale então w é removido de L. Como conseqüência não há aresta *não-visitadas a partir de* w após $d(w)$ execuções da linha 4 com $u = w$, e após $d(w) + 1$ execuções w é removido de L. O resultado agora segue da proposição anterior e do fato de nunca uma aresta ser marcada não-visitada. □

Corolário 5. *O algoritmo* Visite(G, v) *termina.*

Demonstração. Por definição de grafo o conjunto $|V(G)|$ é finito. Pela proposição 3 cada vértice entra em L no máximo uma vez. Pela proposição 4 após no máximo

$$\sum_{u \in V(G)} d(u) + 1 = 2|E| + |V|$$

iterações do **enquanto - faça** na linha 3 temos $L = \varnothing$. □

Lema 6. *Numa execução de* Visite(G, v) *todo vértice alcançável a partir de* v *é visitado.*

Demonstração. Seja $w \in V(G)$ um vértice alcançável a partir de v. Por definição existe uma seqüência

$$v_1, v_2, \ldots, v_{m-1}, v_m \tag{1.19}$$

de vértices de G tal que

$$v = v_1,\ w = v_m \text{ e } v_i \text{ é adjacente a } v_{i+1}\ \ (\forall i \in \{1, 2, \ldots, m-1\}).$$

Vamos provar por indução em m que se w é alcançável a partir de v por uma seqüência de m arestas então w é visitado.

Para a base da indução, suponha $m = 1$. Como v é alcançável a partir de v e é visitado (linha 2), a base está provada.

Suponha (hipótese da indução) que todo vértice alcançável a partir de v por uma seqüência de $m-1$ vértices é visitado, $m > 1$. Seja w um vértice alcançável a partir de v por uma seqüência de m vértices, como em (1.19).

Escreva $e_m = \{v_{m-1}, v_m\}$. Como v_{m-1} é alcançável por uma seqüência de $m-1$ arestas, por hipótese v_{m-1} foi visitado, portanto, entrou em L. Pelo corolário5 v_{m-1} é removido de L em algum momento da execução e antes disso a aresta e_m é visitada a partir de v_{m-1} na linha 7; nesse ponto ou v_m já foi visitado ou vale a condição da linha 8 e v_m é visitado na linha 10. Assim $w = v_m$ é visitado. Pelo Princípio da Indução Finita todo vértice alcançável por m arestas é visitado, para todo inteiro $m \geq 1$. □

Corolário 7. *Cada aresta de* G[A] *é visitada duas vezes.*

Demonstração. Seja $\{x, y\}$ uma aresta de G[A]. Como $x \in A$ temos $x \in L$ em algum momento da execução de Visite(G, v) e antes de x ser removido de L a aresta $\{x, y\}$ é visitada a partir de x. Analogamente, de $y \in A$ temos que a aresta $\{x, y\}$ é visitada a partir de y, logo a aresta é visitada duas vezes. Como uma aresta visitada nunca será rotulada como não-visitada durante uma execução, a aresta $\{x, y\}$ será visitada exatamente duas vezes. □

Com isso temos o seguinte resultado, que estabelece que Visite(G, v) faz o que foi prometido. Esse resultado é a correção do algoritmo, isto é, a prova de que ele funciona corretamente.

Teorema 8. *Após* Visite(G, v) *cada vértice de* A *foi visitado uma vez e cada aresta de* G[A] *foi visitada duas vezes.* □

O próximo passo é determinar a complexidade do algoritmo. Como é usual, $O(1)$ denota tempo constante. As duas primeiras linhas tem tempo $O(1)$. Vimos que o laço é executado no máximo $2|E| + |V|$ vezes, logo uma estimativa é o número máximo de rodadas do laço vezes o custo de cada rodada; se cada linha tiver custo constante, então o custo total do laço é $O(|E| + |V|)$. Para determinar o custo de cada linha precisamos saber detalhadamente como cada operação de cada linha pode ser realizada.

A linha 1 é feita em tempo constante por hipótese, assim como as linhas 3, 4, 9 e 11. A linha 2 pode ser implementada em tempo constante, basta manter um vetor com $|V|$ posições, a posição i do vetor contém 0 ou 1 indicando se o vértice i é não-visitado ou visitado. Com isso, as linha 8 e 10 também tem custo $O(1)$. Resta determinar os custos das linhas 5, 6 e 7; pra esses casos basta manter um ponteiro em cada lista de adjacências N[i] indicando o primeiro vértice não visitado dela, os detalhes são deixamos a cargo do leitor.

Lema 9. *A complexidade de* Visite(G, v) *é* $O(|A| + |E(G[A])|)$.

Demonstração. O algoritmo Visite(G, v) pode ser implementado de forma que cada linha tenha custo constante, logo a complexidade é proporcional ao número de rodadas do laço na linha 3, ou seja, o tempo gasto é proporcional ao número de visitas. Pelo teorema 8 a complexidade é $O(|A| + |E(G[A])|)$. □

Agora, é simples mostrar (exercício 67) que o seguinte algoritmo percorre o grafo (V, E), visitando cada vértice uma única vez e visitando cada aresta duas vezes e com complexidade de tempo $O(|V| + |E|)$.

Algoritmo 2: Percurso(G).

Dado : um grafo G.
Devolve: visita aos vértices e arestas de G.

```
1  marque todos elementos de V(G) e de E(G) não-visitado;
2  enquanto há vértice não-visitado faça
3      escolha v não-visitado;
4      insira v em L;
5      marque v visitado;
6      enquanto L ≠ ∅ faça
7          escolha u ∈ L;
8          se em E_G(u) há aresta não-visitada a partir de u então
9              escolha {u, w} em E_G(u) não-visitada a partir de u;
10             marque {u, w} visitada a partir de u;
11             se w é não-visitado então
12                 insira w em L;
13                 marque w visitado;
14         senão remova u de L.
```

Exercícios

Exercício 67. Prove que Percurso(G) visita cada vértice de G uma vez e cada aresta de G duas vezes e tem complexidade $O(|V(G)| + |E(G)|)$.

Exercício 68. Para facilitar a análise do seguinte algoritmo, você pode considerar instâncias G tais que todo vértice de G é alcançável a partir de qualquer vértice dado.

Algoritmo 3: Visite_vértices(G).

Dado : um grafo G tal que todo vértice é alcançável a partir de qualquer outro vértice.
Devolve: visita aos vértices de G.

```
1  marque todos elementos de V(G) não-visitado;
2  escolha um vértice v não-visitado;
3  insira v em L;
4  marque v visitado;
5  enquanto E(L, L̄) ≠ ∅ faça
6      escolha uma aresta {u, w} ∈ E(L, L̄);
7      insira {u, w} em M;
8      v ← {u, w} ∩ L̄;
9      insira v em L;
10     marque v visitado.
```

(a) Prove ou refute: o seguinte algoritmo visita todos os vértices de G (Pode-se assumir que $E(L, \overline{L}) \neq \varnothing$ se e somente se $L = V(G)$; esse fato será provado no capítulo 3).

(b) Prove que, no final de execução do algoritmo, M contém $|V(G)| - 1$ arestas.

(c) Determine a complexidade do algoritmo do exercício anterior. Justifique tudo com cuidado, principalmente no que se refira as linhas 5, 6 e 8. Pode-se assumir que as operação em L e em M são feitas em tempo constante.

1.7 Algoritmos de busca

Algoritmo de busca é como são chamados, usualmente, alguns dos algoritmos de percurso em grafos. Nessa seção veremos dois casos particulares do algoritmo 2 visto acima, cada caso é obtido quando definimos uma política de gerenciamento para a lista L.

Se consideramos apenas as visitas aos vértice, então uma estratégia geral de busca pode ser descrita da seguinte maneira: enquanto houver vértice u não-visitado, visite u e seus vizinhos não-visitados. Para visitar os vizinhos de u destacamos as duas estratégias a seguir, as quais refletem duas políticas de gerenciamento da lista L.

1.7.1 Busca em Largura

Numa busca em largura, primeiro visitamos cada vizinho não-visitado de v, percorremos $N(v)$ e guardamos-o numa fila; quando não houver mais vizinhos removemos v, pegamos o primeiro vértice da fila e repetimos o processo. Em outras palavras, a lista L é administrada como *fila*.

No algoritmo 2, inserção é feita no fim da fila L, remoção é feita no começo da fila L e a linha `escolha` $u \in L$ devolve o primeiro elemento da fila.

1.7.2 Busca em Profundidade

Numa busca em profundidade, visitamos cada vizinho não visitado de v, percorremos $N(v)$ e guardamos-o numa pilha; quando não houver mais vizinhos removemos v de L, pegamos o topo da pilha e repetimos o processo. Em outras palavras, a lista L acima é administrada como *pilha*.

No algoritmo 2, inserção é feita no topo da pilha L, remoção é feita no topo da pilha L e a linha `escolha` $u \in L$ devolve o topo da pilha L.

Busca em profundidade rotulada. A seguinte versão da busca em profundidade, que chamamos de *busca em profundidade rotulada*, será muito útil adiante; a idéia é manter um contador de operações na pilha e rotular cada vértice com dois valores inteiros: o valor do contador quando o vértice entrou na pilha e o valor do contador quando o vértice saiu da pilha. Pra isso o algoritmo usa dois vetores, $chega[\,]$ e $sai[\,]$, indexados por V. Em $chega[v]$ temos o momento no qual v foi empilhado e em $sai[v]$ o momento em que v foi removido da pilha.

Ainda, o algoritmo usa um vetor $pai[\,]$ indexado pelos vértices. No final da execução $pai[v]$ tem o vértice que foi empilhado antes de v durante a execução, em outras palavras, foi o vértice que descobriu v.

A seguir damos uma versão recursiva para a busca em profundidade rotulada (a pilha fica implementada, implicitamente, pela pilha da recursão) a qual recebe um grafo G dado por uma lista de adjacências e um vértice $w \in V(G)$; no início temos $cont = sai[v] = chega[v] = 0$ e $pai[v] = nil$, para todo vértice v de G.

Algoritmo 4: $BP(G, w)$.

Dado : um grafo G e um vértice w.
Devolve: busca em profundidade em G a partir de w com rótulos $chega$, sai e pai em cada vértice.

```
1 cont ← cont + 1;
2 chega[w] ← cont;
3 para cada u ∈ N(w) faça
4     se chega[u] = 0 então
5         pai[u] ← w;
6         BP(G, u);
7 cont ← cont + 1;
8 sai[w] ← cont.
```

Proposição 10. *Após uma execução de* $\mathrm{BP}(G, w)$*, para quaisquer* $u, v \in V(G)$ *alcançáveis a partir de* w*, um e somente um dos itens abaixo vale para esse par de vértices*

(a) $\mathrm{chega}[u] < \mathrm{sai}[u] < \mathrm{chega}[v] < \mathrm{sai}[v]$;
(a') $\mathrm{chega}[v] < \mathrm{sai}[v] < \mathrm{chega}[u] < \mathrm{sai}[u]$;
(b) $\mathrm{chega}[u] < \mathrm{chega}[v] < \mathrm{sai}[v] < \mathrm{sai}[u]$;
(c) $\mathrm{chega}[v] < \mathrm{chega}[u] < \mathrm{sai}[u] < \mathrm{sai}[v]$.

Demonstração. A prova é deixada como exercício. □

Definimos, para todo $t \in \mathbb{N}$, a t-ésima iteração do vetor pai, denotada por $\mathbf{pai^{(t)}}$, da seguinte maneira

$$\mathrm{pai}^{(t)}[v] = \begin{cases} v & \text{se } t = 0 \\ \mathrm{pai}^{(t-1)}\big[\mathrm{pai}[v]\big] & \text{se } t > 0. \end{cases} \tag{1.20}$$

Exercícios

Exercício 69. Escreva uma versão não recursiva da busca em profundidade rotulada.

Exercício 70. Prove a proposição 10.

Exercício 71. Considere uma execução da busca rotulada $\mathrm{BP}(G, w)$. Para simplificar, considere que todo vértice de G é alcançável a partir de w. Prove as seguintes equivalências. Para quaisquer $u, v \in V(G)$

1. $\big[\mathrm{chega}[v], \mathrm{sai}[v]\big] \cap \big[\mathrm{chega}[u], \mathrm{sai}[u]\big] = \varnothing$ se e somente se não existe $t \in \mathbb{N}$ tal que $\mathrm{pai}^{(t)}[u] = v$ e não existe $t' \in \mathbb{N}$ tal que $\mathrm{pai}^{(t')}[v] = u$;
2. $\big[\mathrm{chega}[v], \mathrm{sai}[v]\big] \subset \big[\mathrm{chega}[u], \mathrm{sai}[u]\big]$ se e somente se existe $t \in \mathbb{N}$ tal que $\mathrm{pai}^{(t)}[v] = u$;
3. $\big[\mathrm{chega}[v], \mathrm{sai}[v]\big] \supset \big[\mathrm{chega}[u], \mathrm{sai}[u]\big]$ se e somente se existe $t \in \mathbb{N}$ tal que $\mathrm{pai}^{(t)}[u] = v$.

Exercício 72. Adapte o algoritmo 3 do exercício 68 para que o percurso seja uma busca em largura. Faça o mesmo para busca em profundidade.

CAMINHOS, CIRCUITOS E CAMINHOS MÍNIMOS

Neste capítulo tratamos de duas classes especiais de grafos e de um problema algorítmico clássico. Na primeira seção apresentamos a classe dos caminhos e noções que permeiam essa classe como comprimento de caminhos e alguns parâmetros associados a caminhos em grafos como distância entre vértices e diâmetro do grafo, em seguida apresentamos a classe dos circuitos. A segunda seção trata do problema algorítmico de determinar distâncias entre pares de vértices num grafo com pesos nas arestas. Mais especificamente, nessa seção são apresentados dois algoritmos tradicionais para o problema: o algoritmo de Dijkstra de 1959 e o algoritmo e o algoritmo de Floyd e Warshall de 1962. Esses algoritmos usam uma técnica para projeto de algoritmos conhecida como Programação Dinâmica.

2.1 Caminhos e circuitos

Um **caminho** é qualquer grafo ou subgrafo isomorfo a (V, E) dado por

$$\begin{aligned} V &= \{0, 1, \ldots, k\} \\ E &= \big\{\{i, i+1\} \colon 0 \leq i \leq k-1\big\}, \end{aligned} \tag{2.1}$$

para algum $k \in \mathbb{N}$.

Exemplo 14. Um caminho com quatro vértices é representado pelo diagrama da figura 2.1.

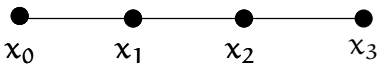


Figura 2.1: Representação geométrica do caminho $\big(\{x_0, x_1, x_2, x_3\}, \big\{\{x_0, x_1\}, \{x_1, x_2\}, \{x_3, x_3\}\big\}\big)$.

O caminho P definido em (2.1) também é denotado pela seqüência

$$P = 0, 1, \ldots, k$$

de seus vértices, de modo que vértices consecutivos na seqüência são adjacentes. Seguindo notação estabelecida na seção 1.1, denotamos por P^{k+1} um caminho com $k+1$ vértices.

Exemplo 15. Considere o grafo G representado no diagrama da figura 2.2 abaixo. O caminho $P^4 = a, b, f, i$ é o subgrafo $G[\{a, b, f, i\}]$, também o subgrafo $P^5 = \big(\{a, d, c, h, i\}, \big\{\{a, d\}, \{d, c\}, \{c, h\}, \{h, i\}\big\}\big)$ é um caminho. O subgrafo induzido pelo conjunto de vértices $\{m\}$ é um P^1 e o subgrafo induzido pela aresta $\{j, l\}$ é um P^2.

Num caminho P, dizemos que os vértices de grau 1 em P são os **extremos** de P e dizemos que os vértices de grau 2 em P são os **vértices internos** de P.

O número de arestas num caminho é o **comprimento** desse caminho.

Em um grafo $G = (V, E)$, a **distância** entre dois vértices quaisquer $u, v \in V$, denotada por $\mathrm{dist}_G(u, v)$, é definida como o comprimento do menor caminho com extremos u e v, isto é

$$\mathrm{dist}_G(u, v) = \min\big\{k \in \mathbb{N} \colon u, v \text{ são extremos de um caminho } P^{k+1} \subseteq G\big\}, \tag{2.2}$$

quando existe algum caminho. Se u e v não são extremos de algum caminho em G, então convencionamos $\mathrm{dist}_G(u, v) = \infty$, de modo que ∞ satisfaz

$$n + \infty = \infty \quad \text{e} \tag{2.3}$$

$$n < \infty, \tag{2.4}$$

para todo $n \in \mathbb{N}$.

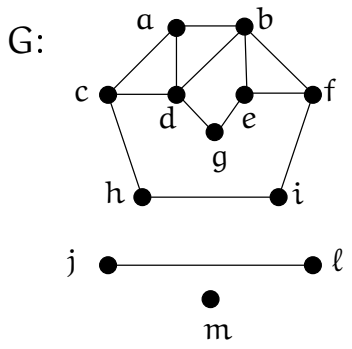


Figura 2.2: Exemplos de subgrafos que são caminhos.

Exemplo 16. No grafo do exemplo 15 temos $\mathrm{dist}_G(a,j) = \infty$, $\mathrm{dist}_G(a,b) = 1$, $\mathrm{dist}_G(a,i) = 3$, $\mathrm{dist}_G(m,j) = \infty$. Observe que o caminho de comprimento mínimo pode não ser único.

Observação 8 (dist é uma *métrica*). Para todo G temos definida a função $\mathrm{dist}_G\colon V(G)\times V(G) \to \mathbb{N}$ que satisfaz

1) $\mathrm{dist}_G(u,v) = \mathrm{dist}_G(v,u)$ para todos $u, v \in V(G)$;

2) $\mathrm{dist}_G(v,u) = 0$ se e somente se $u = v$;

3) a *desigualdade triangular*

$$\mathrm{dist}(u,w) \leq \mathrm{dist}(u,v) + \mathrm{dist}(v,w) \tag{2.5}$$

para quaisquer $u, v, w \in V(G)$.

Portanto, a distância define uma métrica em $V(G)$.

Definimos o **diâmetro** do grafo G como a maior distância entre dois vértices quaisquer de G, ou seja

$$\mathrm{diam}(G) = \max_{u,v\in V(G)} \mathrm{dist}_G(u,v). \tag{2.6}$$

Naturalmente, $\mathrm{diam}(G) = \infty$ se existirem dois vértices no grafo G que não extremos de um caminho.

Um **circuito** é qualquer grafo ou subgrafo isomorfo a (V,E) dado por

$$\begin{aligned} V &= \{1,\ldots,k\} \\ E &= \big\{\{i,i+1\}\colon 1 \leq i \leq k-1\} \cup \big\{\{k,1\}\big\}, \end{aligned} \tag{2.7}$$

para algum $k \geq 3$.

Como no caso do caminho, para simplificar, o circuito definido em (2.7) também é denotado pela seqüência de seus vértices

$$C = 1,2\ldots,k,1$$

de modo que vértices consecutivos são adjacentes. O número de arestas num circuito é o **comprimento** desse circuito e denotamos um circuito de comprimento k por C^k.

Exemplo 17. Consideremos o grafo G do exemplo 15 temos que $C^4 = G\big[\{b,d,e,g\}\big]$ é um circuito, bem como o subgrafo $C^5 = (V,E)$ dado por $V = \{a,b,d,e,g\}$ e $E = \{\{a,b\},\{b,e\},\{e,g\},\{g,d\},\{d,a\}\}$.

Observação 9. Quando denotamos um caminho por uma seqüência $v_1, v_2, \ldots, v_k$ os elementos da seqüência são 2-a-2 distintos, ou seja, não há vértices repetidos na seqüência. Quando a seqüência denota um circuito a *única* repetição permitida é a do primeiro vértice que deve ser igual ao último.

Proposição 11. *Todo grafo* G *contém um caminho de comprimento pelo menos* $\delta(G)$ *e, se* $\delta(G) \geq 2$*, um circuito de comprimento pelo menos* $\delta(G)+1$.

Demonstração. Dado um grafo G, vamos exibir um caminho com pelo menos $\delta(G)$ arestas e um circuito com pelo menos $\delta(G)+1$ arestas.

Seja $P = x_0, x_1, \ldots, x_k$ um caminho de comprimento máximo em G. Temos que

$$N_G(x_0) \subset V(P), \tag{2.8}$$

caso contrário, haveria caminho mais longo que P em G. Logo, $|N_G(x_0)| \leq |V(P)| - 1$ e, como $|N_G(x_0)| \geq \delta(G)$ e $|E(P)| = |V(P)| - 1$, segue que $|E(P)| \geq \delta(G)$, isto é, P é um caminho de comprimento pelo menos $\delta(G)$.

Agora, tratando do caso de circuito, vamos supor que $\delta(G) \geq 2$. Se o grau mínimo é pelo menos dois, então o vértice x_0 tem um vizinho em P diferente de x_1 e assim está bem definido o número

$$m = \max\{j\colon 1 < j \leq k \text{ e } x_j \in N_G(x_0)\}, \tag{2.9}$$

que é o maior índice em $\{2, \ldots, k\}$ de um vértice vizinho de x_0 em P. Agora, basta notar que o circuito $x_0, x_1, \ldots, x_m, x_0$ tem comprimento $m+1$ e que $N_G(x_0) \subseteq \{x_1, x_2, \ldots, x_m\}$, logo $m \geq \delta(G)$. Portanto, C tem comprimento pelo menos $\delta(G)+1$. □

Definimos a **cintura** do grafo G como o comprimento do menor circuito contido em G, ou seja

$$\operatorname{cin}(G) = \min\{k \in \mathbb{N}\colon \text{ existe um circuito } C^k \subseteq G\}, \tag{2.10}$$

quando existe um, se não existe convencionamos $\operatorname{cin}(G) = \infty$.

Notemos que um circuito de comprimento ímpar, ou simplesmente *circuito ímpar* não é um grafo bipartido. Como subgrafo de grafo bipartido também é bipartido (exercício 27), concluímos que grafos bipartidos não contêm circuito ímpar. De fato, a classe dos grafos livres de circuitos de comprimento ímpar e a classe dos grafos bipartidos coincidem.

Teorema 12. *Um grafo* G *é bipartido se e somente se* G *não contém circuitos de comprimento ímpar.*

Demonstração. Seja G um grafo. Suponha que G contém um circuito ímpar e vamos mostrar que G não é bipartido.

A prova é por contradição. Seja $C = x_0, \ldots, x_{2k}, x_0$ um circuito ímpar em G e suponha que G é bipartido com partes A e B. Para $1 \leq j \leq k-1$ temos

$$N_C(x_{2j}) = \{x_{2j-1}, x_{2j+1}\}$$

portanto, x_{2j} não pode pertencer a mesma parte que os vértices $\{x_{2j-1}, x_{2j+1}\}$. Digamos, sem perda de generalidade, que

$$\bigcup_{j=1}^{k-1} \{x_{2j}\} \subseteq A$$

e, portanto,

$$\bigcup_{j=1}^{k-1} \{x_{2j-1}, x_{2j+1}\} \subseteq B.$$

Como $x_1 \in B$ devemos ter $x_0 \in A$. Como $x_{2k-1} \in B$ devemos ter $x_{2k} \in A$, portanto $\{x_0, x_{2k}\} \in E(C)$, contradizendo o fato de A ser um conjunto independente.

Agora, vamos mostrar que se G não contém circuito ímpar então G é bipartido. Fixamos w, um vértice de G, e seja H_w o subgrafo de G induzido pelos vértices $v \in V(G)$ com $\operatorname{dist}_G(w,v) < \infty$. Primeiro, vamos mostrar que H_w é bipartido. Defina os conjuntos disjuntos

$$A_w = \{v \in V(H_w)\colon \operatorname{dist}_G(w,v) = 2j \text{ para algum } j \in \mathbb{N}\} \tag{2.11}$$

$$B_w = \{v \in V(H_w)\colon \operatorname{dist}_G(w,v) = 2j+1 \text{ para algum } j \in \mathbb{N}\} \tag{2.12}$$

com $A_w \cup B_w = V(H_w)$. Vamos mostrar que esses conjuntos são independentes. Sejam $u, v \in A_w$ distintos e sejam $P = w = x_1, \ldots, x_{2a} = u$ e $Q = w = y_1, \ldots, y_{2b} = v$ os caminhos com extremos

w e u e w e v, respectivamente, e que realizam as distâncias. Se os vértices internos de P e Q são disjuntos então $\{u, v\} \in E(G)$ implicaria num circuito ímpar, logo u e v não são adjacentes nesse caso. Se os vértices internos não são disjuntos, sejam p e q os maiores índices para os quais $x_p = y_q$. Como os caminhos realizam as distâncias $p = q$ e, como anteriormente, se $\{u, v\} \in E(G)$ então essa aresta e os caminhos de x_p a u e de y_q ($=x_p$) a v formariam circuito ímpar.

Seja $W = \{w_1, w_2, \ldots, w_m\}$ um subconjunto de $V(G)$ tal que $\text{dist}_G(w_i, w_j) = \infty$ para todo $i \neq j$ de cardinalidade máxima. Os subgrafos $H_{w_1}, \ldots, H_{w_m}$ são bipartidos

$$V(H_{w_i}) \cap V(H_{w_j}) = \varnothing, \text{ para todos } i \neq j \tag{2.13}$$

e

$$G = \bigcup_{i=1}^{m} H_i \tag{2.14}$$

logo G é bipartido, pois união de grafos bipartidos disjuntos é bipartido (verifique). □

Exercícios

Exercício 73. É verdade que todo grafo com pelo menos três vértices, exatamente dois vértices de grau 1 e os demais vértices com grau 2 é um caminho?

Exercício 74. Escreva um algoritmo que recebe uma grafo G e decide se G é um caminho. Determine a complexidade do algoritmo.

Exercício 75. Prove que se $u_1, u_2, \ldots, u_k$ é um passeio em G então existem índices $i_1, i_2, \ldots, i_t$ tais que a subseqüência $u_1, u_{i_1}, u_{i_2}, \ldots, u_{i_t}, u_k$ é um caminho em G.

Exercício 76. Sejam $P = u_1, u_2, \ldots, u_k$ e $Q = v_1, v_2, \ldots, v_\ell$ duas seqüências de vértices de um grafo. Definimos a **concatenação** dessas seqüências como a seqüência dos vértices de P seguidos dos vértices de Q, denotada $P, Q = u_1, u_2, \ldots, u_k, v_1, v_2, \ldots, v_\ell$. Supondo que P e Q sejam caminhos no grafo, sob que condições a seqüência P, Q é caminho?

Exercício 77. Prove a desigualdade triangular (equação (2.5)).

Exercício 78. Prove as afirmações feitas nas equações (2.13) e (2.14).

Exercício 79. Escreva um algoritmo baseado na busca em largura para reconhecer grafos bipartidos. Prove que seu algoritmo funciona corretamente.

Exercício 80. Seja G um grafo onde a distância entre qualquer par de vértices é finita. Mostre que se existem $w \in V(G)$ e $\{u, v\} \in E(G)$ tais que $\text{dist}(w, u) = \text{dist}(w, v)$ então $\text{cin}(G) \leq \text{dist}(w, u) + \text{dist}(w, v) + 1$.

Exercício 81. Mostre que a relação $e \sim f$ em $E(G)$ dada por $e = f$ ou existe um circuito $C \subset G$ tal que e e f pertencem a $E(C)$ é uma relação de equivalência.

Exercício 82. Seja G um grafo nas condições do exercício 68 e seja M o conjunto construído por Visite_vértices(G). Prove que G[M] não contém circuito.

Exercício 83. Seja $C \subseteq G$ um circuito no grafo G. Prove que para qualquer $U \subseteq V(G)$ vale que $|E(C) \cap E(U, \overline{U})|$ não pode ser ímpar.

Exercício 84. Seja k um número natural. O **k-cubo** é o grafo cujo conjunto de vértices são as seqüencias binárias de k bits e dois vértices são adjacentes se e somente se as k-tuplas correspondentes diferem exatamente em uma posição. Determine o número de vértices, arestas, o diâmetro e a cintura do k-cubo. Determine os parâmetros α, ω e χ do k-cubo. Prove que o k-cubo é bipartido.

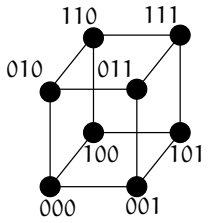


Figura 2.3: Exemplo: o 3-cubo.

Exercício 85. Prove que o seguinte algoritmo (que está baseado numa busca em largura) funciona corretamente. Determine a complexidade do algoritmo.

Algoritmo 5: Distância(G, w).
Dado : um grafo G e um vértice $w \in V(G)$.
Devolve: $\text{dist}_G(w, v)$ para todo $v \in V(G)$.

```
1  marque todos elementos de V(G) não-visitado;
2  marque w visitado;
3  enfileire w em L;
4  d[w] ← 0;
5  enquanto L ≠ ∅ faça
6      tome u o primeiro elemento da fila L;
7      para cada v ∈ N_G(u) faça
8          se v é não-visitado então
9              marque v visitado;
10             enfileire v em L;
11             d[v] ← d[u] + 1;
12     remova u de L;
13 devolva d[ ].
```

2.2 Caminhos mínimos em grafos com pesos nas arestas

Nesta seção consideramos grafos com pesos nas arestas, esses grafos são definidos por uma terna (V, E, ρ) onde V e E são os usuais conjuntos de vértices e arestas, respectivamente, e $\rho\colon E(G) \to \mathbb{R}^+$ é uma função que atribui um peso a cada aresta de E.

O **comprimento** de um caminho $P = x_0, x_1, \ldots, x_k$ em (V, E, ρ) é a soma dos pesos (comprimentos) das arestas do caminho

$$c(P) = \sum_{i=o}^{k-1} \rho(\{x_i, x_{i+1}\}),$$

e a **distância** entre dois vértices u e v é o comprimento do menor caminho com extremos u e v,

$$\text{dist}(u, v) = \min\left\{c(P)\colon P \text{ é um caminho com extremos } u \text{ e } v\right\}$$

quando existe algum caminho, caso contrário convencionamos que $\text{dist}(u, v) = \infty$ tal que valem (2.3) e (2.4), página 31. Um caminho com extremos u e v de comprimento $\text{dist}(u, v) < \infty$ é chamado de **caminho mínimo**.

Esses grafos podem ser representados por lista de adjacências com a lista de v contém os vértices $u \in V(G)$ tais que $\{v, u\} \in E(G)$ junto com o peso $\rho(\{u, v\})$.

Exemplo 18. O diagrama da figura 2.4 representa um grafo com pesos nas arestas e uma lista de adjacências para esse grafo é representada na figura 2.5.

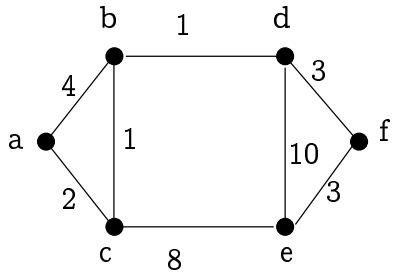


Figura 2.4: Diagrama de um grafo com peso nas arestas. A distância entre a e e é 10 e um caminho mínimo é a, c, b, d, f, e.

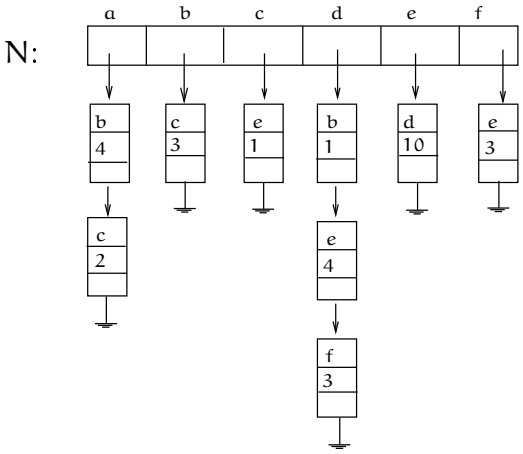


Figura 2.5: Lista de adjacências do grafo G no exemplo acima.

Exercícios

Exercício 86. Seja G um grafo com pesos nas arestas e s um vértice de G. Prove a seguinte variante da desigualdade triangular (2.5): para todo $\{v, u\} \in E(G)$

$$\text{dist}_G(s, u) \leq \text{dist}_G(s, v) + \rho(\{v, u\}). \tag{2.15}$$

Exercício 87. Suponha que $P = v_1, v_2, \ldots, v_k$ é um caminho mínimo num grafo G. Prove que $P_{i,j} = v_i, \ldots, v_j$ é um caminho mínimo com extremos v_i e v_j em G para quaisquer i, j com $1 \leq i < j \leq k$.

Exercício 88. Dados (V, E, ρ) e $s \in V$ considere o problema de determinar $\text{dist}(s, x)$ para todo $x \in V$ com a seguinte estratégia: declare s visitado e em cada passo visite o vértice não visitado mais próximo de um vértice já visitado. Construa um exemplo de grafo com pesos nas arestas que mostre que essa estratégia não funciona.

2.2.1 Algoritmo de Dijkstra para caminhos mínimos

Suponha que são dados $G = (V, E, \rho)$ e um vértice $s \in V$. Queremos determinar $\text{dist}_G(s, v)$, para todo $v \in V$.

As estruturas de dados que vamos usar são: um vetor $d[\,]$, uma estrutura para representar um subconjunto $S \subseteq V$ (pode ser um vetor de bits) e uma fila de prioridades Q.

S: em qualquer ponto da execução do algoritmo o conjunto S identifica os vértices $v \in V$ que já têm a distância $\text{dist}(s, v)$ computada;

$d[\,]$**:** em qualquer ponto da execução do algoritmo $d[v]$ armazena um valor que significa

(a) ou a distância $\mathrm{dist}(s,v)$ de s para v, caso $v \in S$;

(b) ou $d[v] = \infty$, caso v ainda não tenha sido visitado;

(c) ou ainda $d[v]$ é o comprimento de algum caminho com extremos s e v, caso $v \in \overline{S}$;

Q**:** fila de prioridades com elementos de $\overline{S}$ e chave $d[\,]$, o elemento de maior prioridade corresponde ao de menor chave. O procedimento `extrai mínimo de Q` remove e devolve o elemento de maior prioridade de Q.

O algoritmo que descrevemos a seguir recebe G e s como acima, determina $\mathrm{dist}_G(s,x)$, para todo $x \in V$ e funciona da seguinte forma. A inicialização das estruturas de dados são: $S = \varnothing$; $d[s] = 0$ pois s está a distância 0 dele mesmo; para qualquer vértice $v \neq s$ temos $d[v] = \infty$ e Q é uma fila de prioridades com todos os vértices de G e com prioridades dadas por $d[\,]$.

Algoritmo 6: Inicie(G,s).
Dado : um grafo G e um vértice $s \in V(G)$.
Devolve: estruturas de dados iniciadas.

```
1 para cada v ∈ V(G) faça  d[v] ← ∞;
2 d[s] ← 0;
3 construa uma fila de prioridades Q indexada por V(G) e com as prioridades dadas por d[ ].
```

Num determinado momento da execução a fila de prioridades Q contém os vértices de $\overline{S} \neq \varnothing$ e o conjunto S contém os vértices cujas distâncias já estão determinadas. Nesse ponto, retiramos de Q o vértice u de maior prioridade (o que tem menor valor de $d[\,]$). Pela construção da fila de prioridades u é o vértice de $\overline{S}$ (conjunto dos vértices com distância ainda não determinada pelo algoritmo) mais próximo do vértice de saída s, portanto, sua distância fica determinada pelo valor de $d[u]$ e podemos inseri-lo em S. Em seguida, visitamos cada vizinho t de u (por (2.15) $\mathrm{dist}(s,t) \leq d[u] + \rho(\{u,t\})$) e testamos se um caminho de menor comprimento foi encontrado, se for o caso então atualizamos $d[\,]$, essa etapa é chamada de Relaxação

Algoritmo 7: Relaxação(u,t).
Dado : um grafo G com pesos ρ nas arestas e uma aresta $\{u,t\} \in E(G)$.
Devolve: relaxação na aresta $\{u,v\}$.

```
1 se d[t] > d[u] + ρ({u,t}) então  d[t] ← d[u] + ρ({u,t}).
```

Note que a condição na linha 1 do algoritmo 7 é sempre falsa para $t \in S$. O algoritmo prossegue, até que $Q = \varnothing$. Esse algoritmo é conhecido como algoritmo de Dijkstra.

Algoritmo 8: Dijkstra(G,s).
Dado : um grafo G com pesos ρ nas arestas e um vértice $s \in V(G)$.
Devolve: $\mathrm{dist}_G(s,v)$ para todo $v \in V(G)$.

```
1 Inicie(G,s);
2 enquanto Q não-vazio faça
3     u ← extrai mínimo de Q;
4     insira u em S;
5     para cada t ∈ N(u) faça
6         Relaxação(u,t).
```

Uma demonstração formal de que o algoritmo funciona, isto é, no final $d[v]$ armazena as distâncias $\mathrm{dist}(s,v)$, para todo $v \in V$, e a complexidade do algoritmo são determinados a seguir.

Análise e correção do algoritmo de Dijkstra. Vamos mostrar que o algoritmo funciona corretamente.

Proposição 13. *Em qualquer momento da execução de* Dijkstra(G, s) *valem os seguintes fatos*

(a) $d[v] \geq \text{dist}(s, v)$ *para todo* $v \in V$ *e uma vez que* $d[v] = \text{dist}(s, v)$ *o valor de* $d[v]$ *nunca mais muda.*

(b) *Se* $\text{dist}(s, v) = \infty$ *então* $d[v] = \infty$ *durante toda a execução.*

Demonstração. Para provar (a) vamos supor o contrário e assumir que $u \in V$ é o primeiro vértice para o qual ocorreu que $d[u] < \text{dist}(s, u)$ durante uma execução e seja $v \in V$ o vértice que causou a mudança no valor de $d[u]$ numa relaxação, assim a relaxação faz

$$d[u] = d[v] + \rho(\{u, v\}) \tag{2.16}$$

e, nesse momento da execução $d[v] + \rho(\{u, v\}) \geq \text{dist}(s, v) + \rho(\{u, v\})$ pois u foi a primeira ocorrência do fato suposto, ou seja,

$$\text{dist}(s, u) > d[u] = d[v] + \rho(\{u, v\}) \geq \text{dist}(s, v) + \rho(\{u, v\})$$

contradizendo a desigualdade (2.15).

Para provar (b) notemos que, por (a), $d[v] \geq \text{dist}(s, v) = \infty$. □

Proposição 14. *Se* $P = s, x_1, \ldots, x_k, v$ *é um caminho mínimo e* $d[x_k] = \text{dist}(s, x_k)$ *então após* Relaxação(x_k, v) *teremos* $d[v] = \text{dist}(s, v)$.

Demonstração. Após Relaxação(x_k, v) temos $d[v] \leq d[x_k] + \rho(\{x_k, v\}) = \text{dist}(s, x_k) + \rho(\{x_k, v\})$. Pelo exercício 87 $\text{dist}(s, x_k) + \rho(\{x_k, v\}) = \text{dist}(s, v)$, portanto, $d[v] \leq \text{dist}(s, v)$. Pela proposição 13(a), $d[v] \geq \text{dist}(s, v)$, portanto $d[v] = \text{dist}(s, v)$. □

A correção do algoritmo segue do próximo resultado. Provaremos um *invariante do laço* enquanto. Esse invariante é uma propriedade que permanece válida a cada iteração laço, independentemente do número de vezes que é executado.

Teorema 15. *Para todo* $k \in \mathbb{N}$*, após* k *iterações* enquanto *na linha* 2 *vale que*

$$d[v] = \text{dist}(s, v) \text{ para todo } v \in S.$$

Demonstração. No início, antes da primeira iteração, temos $S = \varnothing$, portanto o teorema vale trivialmente.

A prova é por contradição. Suponha que

$$X = \{t \in \mathbb{N}: \text{ após a t-ésima rodada existe } u \in S \text{ tal que } \text{dist}(s, u) \neq d[u]\} \neq \varnothing.$$

Pelo Princípio da Boa-Ordenação, esse conjunto tem um mínimo. Seja $k = \min X$, $k \geq 1$. Seja u o primeiro vértice inserido em S e com $d[u] \neq \text{dist}(s, u)$, ou seja, o vértice inserido na k-ésima iteração do laço. Seja P um caminho mínimo com extremos s e u em G. Esse caminho existe por causa da afirmação feita na proposição 13(b).

Seja $\{x, y\} \in E(G)$ a primeira aresta de P no sentido de s para u com $x \in S$ e $y \in \overline{S}$ no momento da execução imediatamente anterior a u entrar em S. Notemos que, nesse momento da execução, $u \neq s$, logo $S \neq \varnothing$, é possível termos $x = s$ e $y = u$ mas $x \neq u$.

Agora, quando x entrou em S, antes de u, ocorreu Relaxação(x, y) o que fez com que $d[y] = \text{dist}(s, y)$ pela proposição 14 e por $d[x] = \text{dist}(s, x)$. Como os pesos não são negativos e P é mínimo $\text{dist}(s, y) \leq \text{dist}(s, u)$, ou seja

$$d[y] = \text{dist}(s, y) \leq \text{dist}(s, u) \leq d[u]$$

onde a última desigualdade vem da proposição 13(a). Mas, $d[u] \leq d[y]$ pois ambos estão em $\overline{S}$ e u foi escolhido, assim $d[u] = \text{dist}(s, u)$ e temos uma contradição. □

A correção do algoritmo de Dijkstra segue facilmente do teorema acima; cada iteração remove um vértice de Q, logo após $|V|$ iterações $Q = \varnothing$, $S = V(G)$ e a afirmação do teorema é que $d[v]$ está correto para todo $v \in V(G)$. A prova de que Dijkstra, com uma pequena modificação, encontra um caminho mínimo e deixada para o exercício 94.

Agora, vamos analisar a complexidade do algoritmo de Dijkstra, a qual depende da estrutura idealizada para a fila de prioridades. No caso de *heap* binária $O(|V|)$ é o tempo para construir uma fila de prioridade (veja [4]), portanto $\text{Inicie}(G, s)$ tem complexidade $O(|V|)$. A contribuição do laço da linha 2 para a complexidade do algoritmo é

$$\sum_{u \in V} T_3(u) + T_4(u) + T_{5,6}(u)$$

onde $T_i(u)$ é o tempo gasto no linha i para u dado na linha 3.

A remoção da linha 3 tem o custo $O(\log(|\overline{S}|))$ de refazer a *heap* [4], logo $T_3(u) = O(\log|V|)$ para todo $u \in V$. Inserção em S pode ser feita em tempo constante, ou seja, $T_4(u) = O(1)$. Agora, $T_{5,6}(u)$ pode ser escrita como

$$T_{5,6}(u) = \sum_{t \in N(u)} T_6(u).$$

Cada vez que a linha 6 de Dijkstra é executada pode ser necessária uma atualização da *heap*, da seguinte forma

$$Q[t] \leftarrow \min\{Q[t], d[u] + \rho(\{u, t\})\};$$
$$\text{refaz } \texttt{heap}.$$

Essa operação toma tempo $O(\log|V|)$. Logo

$$\sum_{u \in V} T_3(u) + T_4(u) + T_{5,6}(u) = \sum_{u \in V} \left(O(\log|V|) + O(1) + \sum_{t \in N(u)} O(\log|V|) \right) = O\left((|V| + |E|)\log|V|\right).$$

Teorema 16. *O algoritmo de Dijkstra sobre* $G = (V, E)$ *tem complexidade* $= O\left((|V| + |E|)\log|V|\right)$.

Exercícios

Exercício 89. Construa um grafo com pesos que podem ser negativos e no qual algoritmo de Dijkstra devolve resposta errada.

Exercício 90. Suponha que o teste na linha 7 do algoritmos de Dijkstra seja mudado para $|Q| > 1$, o que faria o laço ser executado $|V| - 1$ vezes ao invés de $|V|$. O funcionamento do algoritmo estaria correto?

Exercício 91. Na estimativa de $T_3(u)$ dissemos que $T_3(u) = O(\log(|\overline{S}|))$ e depois superestimamos tomando $T_3(u) = O(\log|V|)$ para todo u. Usando a primeira estimativa, obteríamos

$$\sum_{u \in V} T_3(u) = \sum_{i=1}^{|V|} O(\log i).$$

Prove que, assintoticamente, a alternativa usada não é superestimada, isto é, prove que

$$\sum_{i=1}^{n} O(\log i) = \Theta(n \log n).$$

Exercício 92. Dado um grafo D com pesos $\rho\colon E(D) \to [0, 1]$ que representam a probabilidade de uma aresta não falhar. Escreva um algoritmo para descobrir o caminho mais seguro entre dois vértices.

Exercício 93. Seja $D = (V, E, \rho)$ um grafo com peso $\rho\colon E \to \{0, 1, \ldots, m\}$ nas arestas, $m \in \mathbb{N}$ fixo. Modifique a fila de prioridades do Dijkstra para computar distância em tempo $O(m|V| + |E|)$.

Exercício 94. Considere a seguinte variante do algoritmo de Dijkstra

Algoritmo 9: Dijkstra'(G, s).

Dado : um grafo G com pesos ρ nas arestas e um vértice $s \in V(G)$.
Devolve: $\text{dist}_G(s, v)$ para todo $v \in V(G)$.

```
1 Inicie(G, s);
2 para cada v ∈ V(G) faça p[v] ← nil;
3 enquanto Q não-vazio faça
4     u ← extrai mínimo de Q;
5     para cada t ∈ N(u) faça
6         se d[t] > d[u] + ρ({u, t}) então
7             d[t] ← d[u] + ρ({u, t});
8             p[t] ← u.
```

O seguinte algoritmo recebe um grafo com pesos nas arestas e um par de vértices w, v desse grafo e devolve uma seqüência de vértices $v_1, v_2, \ldots, v_k$, com $v_1 = w$ e $v_k = v$.

Algoritmo 10: Imprime_caminho_mínimo(G, w, v).

Dado : um grafo G vértices $w, v \in V(G)$.
Devolve: caminho mínimo com extremos w e v, quando existe.

```
1 Dijkstra'(G, w);
2 enquanto p[v] ≠ nil faça
3     empilhe v em L;
4     v ← p[v];
5 imprima w;
6 enquanto L ≠ ∅ faça imprima (desempilhe(L)).
```

Prove que a resposta de Imprime_caminho_mínimo(G, w, v) é um caminho mínimo com extremos w e v. (Dica: prove que a seqüência é um passeio, que o passeio tem comprimento mínimo e conclua que o passeio é um caminho.)

2.2.2 Algoritmo de Floyd–Warshall para caminhos mínimos

Seja $G = (V, E, \rho)$ um grafo com pesos nas arestas. Nesta seção vamos apresentar um algoritmo para resolver o seguinte problema: dado $G = (V, E, \rho)$ computar $\text{dist}_G(i, j)$ para todos $i, j \in V$. Notemos que isso pode ser feito através de Dijkstra(G, v) para todo $v \in V$.

Suponha que G seja representado pela matriz

$$a_{i,j} = \begin{cases} 0 & \text{se } i = j \\ \rho(\{i, j\}) & \text{se } \{i, j\} \in E \\ \infty & \text{caso contrário.} \end{cases} \tag{2.17}$$

Antes de apresentarmos o algoritmo precisamos de algumas definições. Para todo $k \in \{0, 1, \ldots, |V|\}$ denotamos por $[k]$ o subconjunto $\{1, 2, \ldots, k\} \subseteq V$, com $[0] = \varnothing$. Dizemos que o caminho $P = i, v_0, \ldots, v_t, j$ é um $[k]$-caminho se o seus vértices internos $v_0, \ldots, v_t$ pertencem a $[k]$. Com isso podemos definir uma variante da distância que chamaremos de $[k]$-distância entre i e j como o comprimento do menor $[k]$-caminho com extremos i e j e é denotada por $\text{dist}_k(i, j)$, com $\text{dist}_0(i, j) = a_{i,j}$.

Exemplo 19. O seguinte diagrama representa um grafo com pesos nas arestas. O único $[0]$-caminho com extremos 3 e 4 é $3, 4$; os $[1]$-caminhos com extremos 3 e 4 são $3, 4$ e $3, 1, 4$; os $[2]$-caminhos com extremos 3 e 4 são $3, 4$ e $3, 1, 4$ e $3, 2, 4$ e $3, 1, 2, 4$ e $3, 2, 1, 4$; esses são também todos os $[3]$-caminhos e $[4]$-caminhos com extremos 3 e 4.

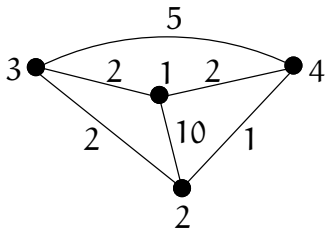


Figura 2.6: Exemplo dos conceitos definidos acima.

No grafo representado acima temos $d_0(3,4) = 5$, $d_1(3,4) = 4$ e $d_2(3,4) = d_3(3,4) = d_4(3,4) = 3$. Também, $d_0(1,2) = d_1(1,2) = d_2(1,2) = 10$, $d_3(1,2) = 4$ e $d_4(1,2) = 3$.

Notemos que $d_0(i,j)$ é dado por $a_{i,j}$ definido em (2.17).

Suponha que conhecemos os valores de $\text{dist}_k(i,j)$ para algum $k > 0$ e para todos os pares $(i,j) \in V \times V$. Podemos obter dist_{k+1} a partir de dist_k da seguinte maneira. Por definição, a $[k+1]$-distância é o comprimento do menor $[k+1]$-caminho e a $[k]$-distância é o comprimento do menor $[k]$-caminho; a diferença entre esses caminhos é que no primeiro, os $[k+1]$-caminhos, é possível que o vértice $k+1$ seja vértice interno e no segundo não, pois $[k+1] = [k] \cup \{k+1\}$. Dessa forma,

$$\text{dist}_{k+1}(i,j) = \min\left\{\text{dist}_k(i,j), \text{dist}_k(i,k+1) + \text{dist}_k(k+1,j)\right\}. \tag{2.18}$$

Resumindo, da matriz de adjacências temos a $[0]$-distância e, para todo $k \in [|V|-1]$, a partir da $[k]$-distância sabemos determinar a $[K+1]$-distância. Repetindo esse procedimento podemos determinar as $[|V|]$-distâncias, que nada mais é do que a distância usual.

Escrevendo de outra forma fica:

Algoritmo 11: Floyd–Warshall(G)

Dado : um grafo G com peso ρ nas arestas.
Devolve: $\text{dist}_G(i,j)$ para todos $i,j \in V(G)$.

```
1 para cada (i,j) ∈ V² faça
2     dist_0(i,j) ← a_{i,j}, onde a_{i,j} é como em (2.17);
3 para cada k de 1 até |V| faça
4     para cada (i,j) ∈ V² faça
5         dist_k(i,j) ← min{dist_{k−1}(i,j), dist_{k−1}(i,k) + dist_{k−1}(k,j)}.
```

Análise e correção do algoritmo de Floyd–Warshall. A correção segue de (2.18).

Teorema 17. *A complexidade do algoritmo de* Floyd–Warshall *com entrada* $G = (V,E)$ *é* $O(|V|^3)$.

Exercícios

Exercício 95. Determine $\text{dist}_i(j,k)$ para toda terna $(i,j,k) \in V^3$ no grafo da figura 2.6.

Exercício 96. A seguinte versão do algoritmo de Floyd–Warshall funciona corretamente? Justifique.

Algoritmo 12: Floyd–Warshall(G)

Dado : um grafo G com peso ρ nas arestas.
Devolve: $\mathrm{dist}_G(i,j)$ para todos $i,j \in V(G)$.

```
1 para cada (i,j) ∈ V² faça
2     dist(i,j) ← a_{i,j};
3 para cada k de 1 até |V| faça
4     para cada (i,j) ∈ V² faça
5         dist(i,j) ← min{dist(i,j), dist(i,k) + dist(k,j)}.
```

Exercício 97. Escreva um algoritmo $O(|V|^3)$ que determina a cintura de um grafo $G = (V, E)$.

CONEXIDADE

Neste capítulo, apresentamos a classe dos grafos conexos, grafos com quaisquer dois vértices ligados por um caminho. Também, apresentamos uma medida de conexidade de grafos, abordamos problemas algorítmicos quando for pertinente e por fim mostramos um exemplo de como construir grafos de alta conexidade, ou seja, que tolera um grande número de remoções sem tornar-se desconexo e com o número mínimo de arestas possível.

3.1 Grafos conexos

Um grafo G é **conexo** se é *não-vazio* e quaisquer dois vértices de G são extremos de um caminho em G.

Dizemos que H é um *subgrafo conexo maximal* de G se não existe um subgrafo conexo $F \subseteq G$ tal que $H \subset F$. Os subgrafos conexo maximais de um grafo qualquer são os **componentes conexos** do grafo. Por exemplo, no grafo do exemplo 1 temos quatro componentes conexos, a saber, induzidas pelos conjuntos de vértices $\{1,2,5\}$, $\{3,4\}$, $\{6,7\}$ e $\{8\}$.

Uma definição equivalente de componente conexo é a seguinte. A relação binária sobre $V(G)$ definida por $u \sim_G v$ se, e somente se, existe um caminho com extremos u e v em G é uma relação de equivalência. As classes de equivalência dessa relação são os vértices dos componentes componentes conexos de G.

É fácil de ver que se G não-vazio é *desconexo* então há pelo menos dois subconjuntos de vértices U e W disjuntos tais que $E(U,W) = \varnothing$. A recíproca dessa afirmação não vale no caso geral (justifique), mas é válida quando $W = \overline{U}$.

Proposição 18. *Um grafo é conexo se e somente se* $E(U,\overline{U}) \neq \varnothing$ *para todo subconjunto próprio e não-vazio de vértices* U.

Demonstração. Se G é conexo e U um subconjunto próprio e não-vazio de $V(G)$, para qualquer $u \in U$ e qualquer $w \in \overline{U}$ existe em G um caminho com extremos u e w e pelo menos uma aresta desse caminho deve estar em $E(U,\overline{U})$.

Por outro lado, suponha que $E(U,\overline{U}) \neq \varnothing$ para todo subconjunto próprio e não-vazio $U \subset V(G)$. Definimos $U(v)$ como o conjunto de todos os vértices $w \in V(G)$ para os quais existe um caminho com extremos v e w em G e vamos mostrar que $U(v) = V(G)$ o que, claramente, significa que G é conexo.

Suponha $U(v) \neq V(G)$. De $v \in U(v)$ temos que $U(v)$ é não-vazio. Como $E\big(U(v),\overline{U(v)}\big) \neq \varnothing$, existe $\{x,y\} \in E\big(U(v),\overline{U(v)}\big)$ e nesse caso $y \in U(v)$ por definição de $U(v)$, contradizendo $\{x,y\} \in E\big(U(v),\overline{U(v)}\big)$. Portanto, $U(v) = V(G)$. □

Se G é um grafo e e uma aresta de G, então definimos o grafo obtido pela remoção de e por

$$\mathbf{G - e} = \big(V, E \setminus \{e\}\big). \tag{3.1}$$

Proposição 19. *Se* $G = (V,E)$ *é conexo então* $G - e$ *é conexo para toda aresta* e *que pertence a algum circuito de* G.

Demonstração. Seja G um grafo conexo, $e = \{v_1, v_2\}$ uma aresta de G e $C = v_1, v_2, \ldots, v_k, v_1$ um circuito de G ao qual pertence a aresta e. Vamos mostrar que $G - e$ é conexo. Vamos denotar por $C - e$ o caminho $v_2, \ldots, v_k, v_1$ em $G - e$.

Sejam u e v dois vértices quaisquer de G e vamos denotar $u = u_1$ e $v = u_m$.

Seja $P = u_1, u_2, \ldots, u_m$ um caminho em G com extremos u e v. Se $e \notin E(P)$ então $P \subseteq G - e$, caso contrário $e = \{u_j, u_{j+1}\}$, para algum j, e $u_1, \ldots, u_{j-1}, C - e, u_{j+2}, \ldots, u_m$ é um passeio em $G - e$ com extremos u e v (veja exercício 75). Logo $G - e$ é conexo. □

Se G é um grafo e v um vértice de G, então definimos o grafo obtido pela remoção de v por

$$\mathbf{G - v} = \big(V \setminus \{v\}, E \setminus E(v)\big). \tag{3.2}$$

Um vértice $v \in V$ num grafo conexo $G = (V, E)$ é chamado de **articulação** (ou **vértice de corte**) se a remoção do vértice v e das arestas de $E(v)$ em G resulta num grafo desconexo, isto é, o grafo $G - v$ é desconexo.

Um grafo sem articulações é chamado de **biconexo**. Os subgrafos biconexos maximais de um grafo G qualquer são chamados de **componentes biconexos** de G.

Exemplo 20. No grafo do diagrama na figura 3.1 são articulações os vértices $1, 3, 7, 9$. Os com-

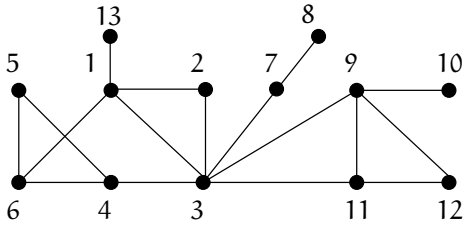


Figura 3.1: $1, 3, 7, 9$ são articulações.

ponentes biconexos são os subgrafos induzidos pelos conjuntos de vértices: $\{1, 13\}$, $\{1, 2, 3, 4, 5, 6\}$, $\{3, 7\}$, $\{7, 8\}$, $\{9, 10\}$ e $\{3, 9, 11, 12\}$.

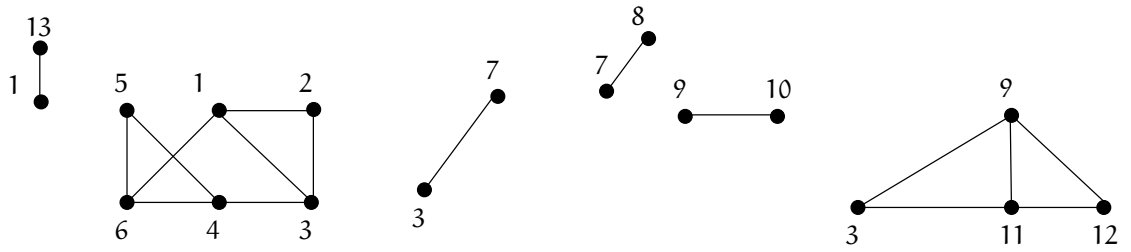


Figura 3.2: Os seis componentes biconexos do grafo exibido na figura acima.

3.1.1 Articulações

Na discussão que segue, o problema computacional que tratamos é: dado G conexo, determinar as articulações de G. Consideramos um grafo conexo G com pelo menos 3 vértices e usaremos a versão rotulada da busca em profundidade, o algoritmo 4, para determinar as articulações de um grafo. Notemos que numa execução de $BP(G, w)$, como o grafo é conexo, todos os vértices serão visitados.

Baseados em (1.20) derivamos a seguinte nomeclatura. Se existe algum inteiro $t \in \mathbb{N}$ tal que $pai^{(t)}[v] = u$ então dizemos que u é **ancestral** de v ou que v é **descendente** de u. No caso particular de $pai[u] = v$ dizemos que u é **filho** de v. Vamos chamar de **patriarca** o vértice inicial de um percurso em profundidade, isso é, o único vértice $v \in V$ que tem $pai[v] = \texttt{nil}$.

Observação 10. Do exercício 71 podemos estabelecer as seguintes equivalências

(a)	$[chega[v], sai[v]] \cap [chega[u], sai[u]] = \varnothing$	v não é descendente de u, nem u é descendente de v
(b)	$[chega[v], sai[v]] \subset [chega[u], sai[u]]$	v é descendente de u
(c)	$[chega[v], sai[v]] \supset [chega[u] sai[u]]$	u é descendente de v

e sabemos (proposição 10) que para quaisquer dois vértices distintos $u, v \in V(G)$ exatamente uma das possibilidades acima vale após a execução do algoritmo.

Para facilitar a referência, reproduzimos abaixo o algoritmo de busca rotulada apresentado na seção 1.7.2

Algoritmo 13: BP(G, w).

Dado : um grafo G e um vértice w.
Devolve: busca em profundidade em G a partir de w com rótulos $\mathtt{chega}$, $\mathtt{sai}$ e $\mathtt{pai}$ em cada vértice.

```
1 cont ← cont + 1;
2 chega[w] ← cont;
3 para cada u ∈ N(w) faça
4     se chega[u] = 0 então
5         pai[u] ← w;
6         BP(G, u);
7 cont ← cont + 1;
8 sai[w] ← cont.
```

Nas linhas 3 e 4 do algoritmo 13 testamos para cada $u \in N(w)$ se $\mathtt{chega}[u] = 0$, ou seja se u não foi visitado. Se a condição é *falsa*, então dizemos que a aresta $\{w, u\}$ é uma **aresta de retorno**. Em outras palavras, aresta de retorno é toda aresta que não é da forma $\{v, \mathtt{pai}[v]\}$ após uma busca em profundidade como no algoritmo 13.

Exemplo 21. A figura 3.3 a seguir ilustra as definições que acabamos de ver com uma busca em profundidade no grafo do exemplo 20 dado pela listas de adjacências ordenadas em ordem crescente e a ordem em que os vértices são visitados é representada de cima para baixo e da esquerda para a direita.

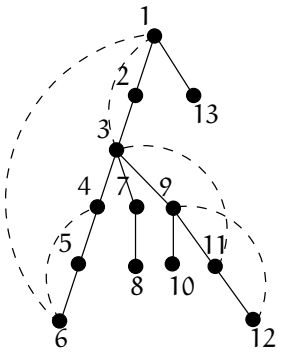


Figura 3.3: Visão esquemática de uma busca em profundidade no grafo do exemplo 20. O vértice 1 é o *patriarca* da busca, as arestas tracejadas são arestas de *retorno*, 2 é *ancestral* de 11 pois $2 = \mathtt{pai}^{(3)}[11] = \mathtt{pai}\big[\mathtt{pai}\big[\mathtt{pai}[11]\big]\big]$ e, reciprocamente, 11 é *descendente* de 2.

O nosso objetivo é caracterizar, usando uma busca em profundidade, os vértices de um grafo conexo que são articulações. Antes, vamos provar dois resultados simples. Lembramos que estamos assumindo G conexo e com pelo menos três vértices.

Consideremos a execução de uma busca em profundidade rotulada BP(G, w).

Proposição 20. *Se* $\{x, y\} \in E(G)$ *então ou* x *é descendente de* y *ou* y *é descendente de* x.

Demonstração. Podemos supor $\{x, y\}$ aresta de retorno, o outro caso é imediato.

Se $\mathtt{chega}[x] < \mathtt{chega}[y]$ então na primeira vez que a aresta $\{x, y\}$ é visitada é a partir de y (corresponde a $u = x$ e $w = y$ na linha 3 do algoritmo 13) e nesse momento x ainda está na pilha, pois a aresta $\{x, y\}$ ainda não foi visitada a partir de x. Portanto $\mathtt{chega}[x] < \mathtt{chega}[y] < \mathtt{sai}[y] < \mathtt{sai}[x]$ e pela observação 10 y é descendente de x.

Se $\mathtt{chega}[y] < \mathtt{chega}[x]$ então deduzimos de maneira análoga que x é descendente de y. □

Proposição 21. *Se v_1 e v_2 são filhos de u então não há descendente de v_1 adjacente a descendente de v_2, para qualquer $u \in V(G)$.*

Demonstração. Se v_1 e v_2 são filhos de u então v_1 não é descendente de v_2 nem v_2 é de v_1, ou seja, $[chega[v_1], sai[v_1]] \cap [chega[v_2], sai[v_2]] = \varnothing$.

Se u é um descendente de v_1 então $chega[v_1] \leq chega[u] < sai[u] \leq sai[v_1]$. Se w é um descendente de v_2 então $chega[v_2] \leq chega[w] < sai[w] \leq sai[v_2]$; logo concluímos que $[chega[u], sai[u]] \cap [chega[w], sai[w]] = \varnothing$ e pelo resultado anterior $\{u, w\} \notin E(G)$. □

O seguinte resultado caracteriza as articulações com relação a uma busca em profundidade. A partir desta caracterização escreveremos um algoritmo que determina as articulações de um grafo conexo.

Teorema 22. *Seja G um grafo não-trivial. O vértice $u \in V(G)$ é uma articulação se e somente se numa busca em profundidade*

(1) u é patriarca e tem mais de um filho; ou

(2) u não é patriarca mas tem um filho v tal que nenhuma arestas de retorno dos descendentes de v tem como outro extremo um ancestral próprio de u.

Demonstração. Vamos provar que se vale a afirmação (1) ou (2) do enunciado para o vértice $u \in V(G)$ então u é uma articulação.

Suponha que u é patriarca com mais de dois filhos e sejam v_1 e v_2 dois desses filhos. Como não há descendente de v_1 adjacente a descendente de v_2 todo caminho com extremos v_1 e v_2 passa por u, ou seja, o grafo $G - u$ é desconexo pois nele não há caminho com extremos v_1 e v_2; assim u é articulação.

Suponha agora que u não é patriarca e seja w o $pai[u]$. Seja v_0 um filho de u com descendentes $\{v_1, v_2, \ldots, v_k\}$. Pela proposição 21 qualquer caminho com extremos v_i ($i \in \{0, 1, \ldots, k\}$) e w ou tem u como vértice interno ou tem uma aresta de retorno $\{v_j, x\}$, para algum $j \in \{0, 1, \ldots, k\}$ e x ancestral próprio de u. Se tais arestas não existem então todo caminho com extremos w e um descendente de v_0 passa por u assim em $G - u$ é desconexo.

Agora, vamos supor que u é uma articulação de um grafo G não-trivial e consideremos uma execução de $BP(G, w)$. Sejam x e y dois vértices de G que não são extremos de caminho em $G - u$. Todo caminho em G com extremos x e y são da forma $x, z_1, z_2, \ldots, z_k, u, z_{k+1}, \ldots, z_t, y$.

Sabemos que ou (i) x é descendente de y, ou (ii) y é descendente de x ou (iii) nem x é descendente de y, nem y de x. No caso (i) temos, pelo fato de u ser articulação e pela proposição 21, que u é ancestral de x e é descendente de y, logo há um caminho $x, z_1, z_2, \ldots, z_k, u, z_{k+1}, \ldots, z_t, y$ com nenhuma aresta sendo de retorno. Se houver uma arestas de retorno entre um descendente de z_k e um ancestral de z_{k+1} então em $G - u$ haverá cominho com extremos x e y. O caso (ii) é análogo, portanto, não há tais arestas de retorno.

Resta o caso em que nem x é descendente de y, nem y de x. Notemos que u é um ancestral comum de x e y. Nesse caso, tome z o ancestral comum de x e y com maior valor de $chega[\,]$. Se $z = w$ então $u = w$ e, portanto, u é patriarca e tem pelo menos 2 filhos. Se u não é patriarca, consideremos um caminho $x, z_1, z_2, \ldots, z_k, u, z_{k+1}, \ldots, z_t, y$ com todas as arestas com extremos pai-e-filho. Como todo caminho entre x e y contém u e $G - u$ é desconexo então z_k não tem descendente adjacente a um ancestral de u ou z_{k+1} não tem descendente adjacente a um ancestral de u. □

Exemplo 22. Considere a representação gráfica da figura 3.3. O vértice 1 é uma articulação pelo caso (1) do lema. O vértice 3 cai no caso (2) com $v = 7$ e 9 cai no caso (2) com $v = 10$ (note que $v = 11$ não serve para classificar 9 no caso (2)).

Pelo teorema 22, para determinar se $u \in V(G)$ é uma articulação basta executar um busca em profundidade a partir de qualquer vértice, testar se o patriarca tem mais de um filho e se todo vértice tem todo filho com um descendente adjacente a um ancestral. A seguinte função é uma

busca em profundidade modificada para identificar articulações baseada no teorema 22. O valor devolvido por esta função com argumento $w \in V(G)$ é

$$m = \min\{chega[x]: x \text{ é ancestral de } u \text{ adjacente a um descendente de } u, \text{ para todo } u \text{ filho de } w\}. \tag{3.3}$$

Notemos que se u é filho de w e $m \geq chega[w]$, então w é articulação pelo teorema 22.

Função `BP-Art(G,w)`.

```
1  cont ← cont + 1;
2  chega[w] ← cont;
3  min ← cont;
4  para cada u ∈ N[k] faça
5      se chega[u] = 0 então
6          m ← BP(u);
7          se m < min então  min ← m;
8          se m ≥ chega[w] então  art[w] ← art[w] + 1;
9      senão se chega[u] < min então  min ← chega[u];
10 devolva(min).
```

Análise e correção do algoritmo BP-Art. O algoritmo é, essencialmente, uma busca em profundidade.

Teorema 23. *A complexidade para determinar as articulações de* $G = (V, E)$ *é* $O(|V| + |E|)$.

Para provar que o algoritmo funciona corretamente, vamos provar que $art[w] = 0$ se e somente se w não é uma articulação.

Note que $art[w]$ só é incrementado se vale a condição $m \geq chega[w]$, onde m é o valor retornado por $BP(G, u)$ quando u é um filho de w, isso significa que para toda aresta $\{x, y\}$, onde y é um descendente de u e x um ancestral de w vale que $chega[x] \geq chega[w]$; pelo teorema 22 isso significa que w é uma articulação.

Exercícios

Exercício 98. Prove que se todos os vértices de G conexo têm grau par então G não tem **aresta de corte**, ou seja, uma aresta cuja remoção resulta num grafo desconexo.

Exercício 99. Mostre que se $\Delta(G) \leq 2$ então os componentes conexos de G ou são caminhos ou são circuitos.

Exercício 100. Seja H um subgrafo gerador de G. Mostre que se H é conexo então G é conexo.

Exercício 101. Seja G um grafo com n vértices. Considere os graus dos vértices em ordem crescente, $\delta(G) = d_1 \leq d_2 \leq \cdots \leq d_n = \Delta(G)$. Mostre que se $d_k \geq k$ vale para todo k com $0 < k < n - \Delta(G)$, então G é conexo.

Exercício 102. Seja G o grafo definido sobre o conjunto de vértices $\{0, 1, \ldots, n-1\}$ com $\{u, v\} \in E(G)$ se, e somente se, $u - v \equiv \pm k \mod n$.

1. Dê um condição necessária e suficiente sobre n e k para que G seja conexo.
2. Determine o número de componentes conexos em função de n e k.

Exercício 103. Escreva um algoritmo com complexidade $O(|V|+|E|)$ que determine os componentes biconexos de um grafo.

Exercício 104 (Teorema de Brooks, 1941). Prove que se G é conexo, não é completo e não é um circuito ímpar então

$$\chi(G) \leq \Delta(G).$$

(Dica: indução na ordem do grafo.)

3.2 Conexidade de grafos

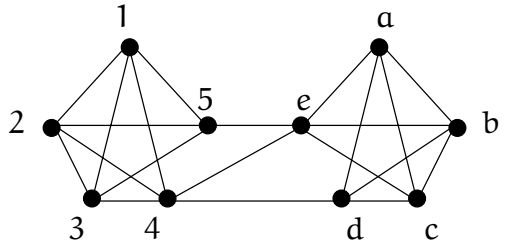
Dado um subconjunto $U \subset V(G)$ denotamos por $G - U$ o subgrafo induzido por $V(G) \setminus U$. Para todo $k \in \mathbb{N}$, dizemos que um grafo G é k-**conexo** se

- $|V(G)| > k$ e
- para todo $U \subset V(G)$, se $|U| < k$ então $G - U$ é conexo.

Claramente, todo grafo não-vazio é 0-conexo. Ainda, todo grafo não-trivial conexo é 1-conexo e todo grafo biconexo com pelo menos três vértices é 2-conexo.

Segue imediatamente da definição que todo grafo k-conexo também é ℓ-conexo para todo natural $\ell < k$.

Exemplo 23. O grafo do diagrama abaixo é 0-conexo, 1-conexo e 2-conexo mas não é 3-conexo porque a remoção do conjunto de vértices $\{e, d\}$ resulta num grafo desconexo.



A **conexidade** de G é o maior inteiro k para o qual G é k-conexo

$$\kappa(G) = \max\{k \in \mathbb{N}\colon G \text{ é } k\text{-conexo}\}. \tag{3.4}$$

No exemplo 23 temos $\kappa(G) = 2$. Para o grafo completo com n vértices temos $\kappa(K^n) = n - 1$ para todo $n > 1$.

Analogamente, podemos definir a aresta conexidade de um grafo. Denotamos por $G - F$ o grafo $(V, E \setminus F)$. Dizemos que G é k-**aresta-conexo** se

- $|V(G)| > 1$ e
- para todo $F \subset E(G)$, se $|F| < k$ então $G - F$ é conexo.

O grafo do exemplo 23 é 0-aresta-conexo, 1-aresta-conexo, 2-aresta-conexo e 3-aresta-conexo, mas não é 4-aresta-conexo.

Segue da definição que todo grafo não-trivial é 0-aresta-conexo, todo grafo não-trivial e conexo é 1-aresta-conexo. Ainda, todo grafo k-aresta-conexo é ℓ-aresta-conexo para todo natural $\ell < k$.

A **aresta-conexidade** de G é o maior inteiro k para o qual G é k-aresta-conexo.

$$\lambda(G) = \max\{k \in \mathbb{N}\colon G \text{ é } \lambda(G) = \max\{k \in \mathbb{N}\colon G \text{ é } k\text{-aresta-conexo}\}. \tag{3.5}$$

No exemplo 23 temos $\lambda(G) = 3$.

Note que o grafo completo com $n > 1$ vértices não é n-aresta-conexo pois a remoção de todas as arestas de $E(v)$, para qualquer v, torna esse vértice isolado. De uma maneira mais geral, temos que $\lambda(G) \leq \delta(G)$. No exemplo 23 vale que $\lambda(G) < \delta(G)$.

Usando o fato descrito no parágrafo acima no teorema 1, página 10 podemos deduzir que o número de arestas num grafo é

$$|E(G)| \geq \frac{1}{2}|V(G)|\lambda(G), \tag{3.6}$$

Logo, o número mínimo de arestas num grafo k-aresta-conexo é

$$|E(G)| \geq \frac{k}{2}|V(G)|. \tag{3.7}$$

Qual é o menor número de arestas que um grafo k-conexo pode ter? Veremos que esse número é $k|V(G)|/2$. Antes demonstraremos o seguinte resultado.

Teorema 24. *Para todo* G *com pelo menos 2 vértices*

$$\lambda(G) \geq \kappa(G). \tag{3.8}$$

Demonstração. Vamos provar por indução que para todo $k \in \mathbb{N}$ a seguinte sentença é verdadeira para qualquer grafo G com pelo menos 2 vértices:

$$\textit{se } \lambda(G) = k \textit{ então } \kappa(G) \leq k. \tag{3.9}$$

Primeiro provamos a base da indução. A sentença vale para $k = 0$ pois se $\lambda(G) = 0$ então G é desconexo e, portanto, $\kappa(G) = 0$.

Para um $k \geq 1$ fixo assumimos, por hipótese, que se $\lambda(G) = k-1$ então $\kappa(G) \leq k-1$ para qualquer grafo G com pelo menos 2 vértices.

Fixamos G um grafo qualquer com pelo menos 2 vértices e $\lambda(G) = k$. Vamos mostrar o passo indutivo que $\kappa(G) \leq k$.

Pela definição de aresta-conexidade sabemos que existe $F \subseteq E(G)$ com $|F| = k$ e $G - F$ desconexo. Escolha uma aresta qualquer $e \in F$ e consideremos o grafo $G - e$.

Pela escolha de e, no grafo $G - e$ existe $F' \subset E(G - e)$ tal que $|F'| = k-1$ e $(G - e) - F'$ é desconexo. Ainda, $\lambda(G - e) = k - 1$ pois o grafo $G - e$ não pode ter um subconjunto de arestas de cardinalidade $k-2$ cuja remoção desconecta $G - e$; caso contrário G teria um subconjunto de arestas de cardinalidade $< k$ cuja remoção o tornaria desconexo.

Pela definição de conexidade ou $|V(G-e)| = \kappa(G-e)+1$ ou existe $U \subset V(G-e)$ com $\kappa(G-e)$ vértices e $(G-e)-U$ desconexo. Pela hipótese indutiva temos que $\kappa(G-e) \leq k-1$.

No primeiro caso temos $|V(G)| = |V(G-e)| \leq k$, portanto $\kappa(G) < k$, ou seja, a sentença (3.9) acima vale.

No segundo caso vale que, ou $G - U$ é desconexo e temos $\kappa(G) \leq |U| \leq k-1$, ou $G - U$ é conexo e a remoção de $\{e\}$ torna $G - U$ desconexo e temos que se $e = xy$ então $G - U - x$ é desconexo, portanto $\kappa(G) \leq |U| + 1 \leq k$. Notemos que $G - U - x = K^1$ não ocorre pois, nesse caso, teríamos $|V(G)| = |U| + 2 = |V(G-e)| + 1$, uma contradição.

Pelo Princípio do Indução Finita a sentença (3.9) vale para todo $k \in \mathbb{N}$ e isso prova o teorema. □

Da equação 3.6 e do teorema que acabamos de provar tiramos que para todo G com pelo menos dois vértices

$$|E(G)| \geq \frac{1}{2}|V(G)|\kappa(G). \tag{3.10}$$

Portanto, o número mínimo de arestas num grafo k-conexo é

$$|E(G)| \geq \frac{k}{2}|V(G)|.$$

Observação 11. Algoritmos eficientes para determinar a conexidade são baseados no teorema de Menger (veja exercício 116) e técnicas de fluxo em redes (seção 6.4).

Exercícios

Exercício 105. Determine $\kappa(G)$ e $\lambda(G)$ nos casos P^k, C^k, K^k, k-cubo (definido no exercício 84), e $K^{m,n}$.

Exercício 106. Dê uma cota inferior para o grau médio de um grafo k-conexo com n vértices.

Exercício 107. É verdade que se G é k-aresta-conexo então $|E(U, \overline{U})| \geq k$ para todo $U \subset V(G)$ não-vazio?

Exercício 108. Prove que se G é 3-regular então $\kappa(G) = \lambda(G)$.

Exercício 109. Prove que todo grafo k-conexo ($k \geq 2$) com pelo menos $2k$ vértices contém um circuito de comprimento maior ou igual a $2k$.

Exercício 110. Prove que todo grafo 2-conexo minimal contém um vértice de grau 2.

Exercício 111. Prove que se $\delta(G) \geq |V(G)| - 2$ então $\kappa(G) = \delta(G)$.

Exercício 112. Prove que se $\delta(G) \geq |V(G)|/2$ então $\lambda(G) = \delta(G)$.

Exercício 113. Prove ou dê um contra-exemplo: dado $k \geq 1$, se G é k-conexo então para todo $U \subseteq V(G)$ com $|U| > k$ o subgrafo $G[U]$ é k-conexo.

Exercício 114. Dado um grafo G qualquer, não necessariamente conexo, os **blocos** do grafo G são:

(1) os subgrafos 2-conexo maximais de G,

(2) as arestas de corte e

(3) os vértices de grau zero.

A partir de G construímos o **grafo de blocos** de G, denotado por B_G da seguinte forma: B_G tem um vértice para cada bloco de G e um vértice para cada articulação dos componentes conexos de G. As arestas são dadas pelos pares vb, onde v corresponde a um vértice de corte de G e b um bloco de G, de forma que "$v \in b$", isto é, o vértice de corte (articulação) em G correspondente a v pertence ao bloco de G que corresponde a b. A figura abaixo mostra um grafo e o grafo de blocos correspondente.

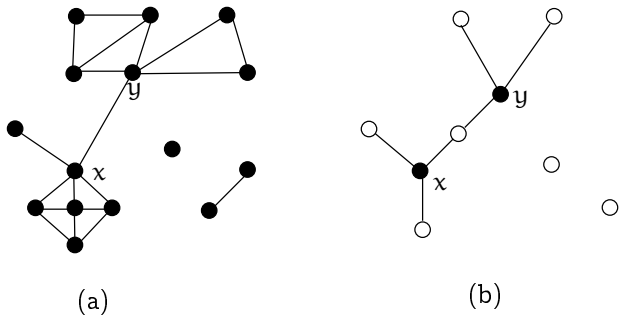


Figura 3.4: (a) Um grafo com articulações x e y e (b) seu grafo de blocos.

Prove que

(a) cada aresta de G pertence a um único bloco,

(b) G é a união de seus blocos,

(c) dois blocos distintos de G têm no máximo um vértice em comum,

(d) se $x \in V(G)$ é um vértice que pertence a dois blocos de G então x é um vértice de corte.

Exercício 115. Descreva um algoritmo que determina os blocos de um grafo. Qual a complexidade do algoritmo?

Exercício 116 (Teorema de Menger, 1927). Prove que se G é k-conexo ($k \geq 1$) se e somente se para quaisquer $u, v \in V(G)$ existem k caminhos disjuntos nas arestas que ligam u a v.

Vale a versão do Teorema de Menger para k-aresta-conexidade?

3.2.1 Construção de grafos k-conexos minimais

Nessa seção vamos mostrar como construir grafos de uma dada conexidade e com o menor número possível de arestas. Dados $k \geq 2$ e $n > k$ vamos construir o grafo $H_{n,k} = (V, E)$ sobre $V = \{0, 1, \ldots, n-1\}$ com

$$\left\lceil \frac{k|V(G)|}{2} \right\rceil$$

arestas. O conjunto de arestas de $H_{n,k}$ depende da paridade de k:

1. Se k é par, digamos $k = 2r$, então

$$E = \{\{i, j\} : -r \leq j - i \leq r \text{ com as operações módulo } n\}.$$

2. Se k é ímpar, digamos $k = 2r + 1$, então

   (2.1) Se n é par,

$$E = E(H_{n,k-1}) \cup \big\{\{i, i + (n/2)\}\colon 1 \leq i \leq n/2\big\}.$$

   (2.2) Se n é ímpar,

$$\begin{aligned} E = E(H_{n,k-1}) \quad &\cup \quad \big\{\{0,(n-1)/2\},\{0,(n+1)/2\}\big\} \\ &\cup \quad \big\{\{i, i + (n+1)/2\}\colon 1 \leq i < (n-1)/2\big\}. \end{aligned}$$

Para determinar $|E(H_{n,k})|$, no caso (1) as arestas são dadas pelo equação (1), onde $k = 2r$. O grau do vértice i é o número de inteiros módulo n no intervalo $[i - r, i + r]$, que corresponde ao próprio i, ou seja, $2r$. Assim, $2|E(H_{n,k})| = nk$ pelo Teorema 1. No outro caso, deixamos a verificação para o leitor.

Lema 25. *O grafo $H_{n,k}$ é k-conexo.*

Demonstração. Nesta demonstração as operações aritméticas nas quais os operandos são vértices são feitas módulo n.

Para mostrar que esse grafo é k-conexo, $k = 2r$, suponhamos que exista $U \subset V(H_{n,k})$ com $|U| < 2r$ e $H_{n,k} - U$ desconexo. Tome dois vértices i e j em componentes distintas de $H_{n,k} - U$ e considere as seqüências de vértices $S = (i, i + 1, i + 2, \ldots, j - 1, j)$ e $T = (j, j + 1, j + 2, \ldots, i - 1, i)$, onde as adições são módulo n. Desde que $|U| < 2r$ temos que $|U \cap S| < r$ ou $|U \cap T| < r$. Vamos assumir que $|U \cap S| < r$, o outro caso é análogo. Com isso, podemos deduzir que existe em $S \setminus U$ uma subseqüência $P = i_0, \ldots, i_t$ de vértices distintos tais que $i_0 = i$, $i_t = j$ e $|i_\ell - i_{\ell+1}| \leq r \pmod n$, logo i_ℓ e $i_{\ell+1}$ são adjacentes para todo $0 \leq \ell < t$. De fato, considere a seqüência de vértices $i, i + r, i + 2r, \ldots, i + tr$ em S, onde t é o menor natural tal que $i + (t - 1)r < j \leq i + tr$. Como $|U| < r$ podemos escolher i_ℓ tal que $i + (\ell - 1)r < i_\ell \leq i + \ell r$.

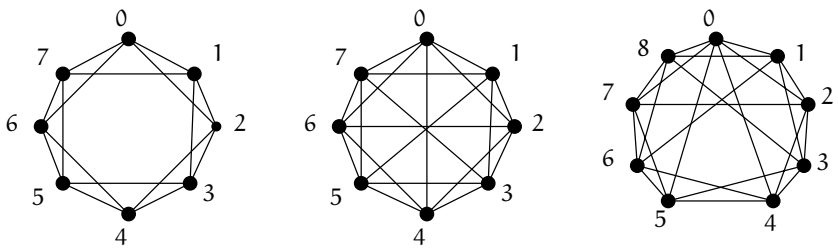
Tal seqüência P é um caminho com extremos i e j, o que contradiz o fato deles estarem em componentes distintas.

Agora, suponha $k = 2r + 1$ com n par e que exista $U \subset V(H_{n,k})$ com $|U| \leq 2r$ e $H_{n,k} - U$ desconexo. Tome dois vértices i e j em componentes distintas de $H_{n,k} - U$ e considere as seqüências de vértices S e T como acima. Se $|U \cap S| < r$ ou $|U \cap T| < r$ então temos um caminho de i até j por dedução análoga ao caso anterior. Vamos assumir que $|U \cap S| = r$ e que $|U \cap T| = r$.

Pra não haver uma seqüência como a P definida acima, é preciso que os r vértices de U em S sejam vértices consecutivos $i + s + 1, i + s + 2, \ldots, i + s + r$. Analogamente, os r vértices de U em T devem ser consecutivos, digamos $j+t+1, j+t+2, \ldots, j+t+r$. Nesse caso, i é um vértice do caminho $Q = j+t+r+1, j+t+r+2, \ldots, i+s$ e j é um vértice do caminho $R = i+s+r+1, i+s+r+2, \ldots, j+t$. Resta notar que deve haver uma aresta da forma $\{v, v + (n/2)\}$ com v em R e $v + (n/2)$ em Q, contradizendo o fato de i e j pertencerem a componentes conexos distintos.

Finalmente, o caso n ímpar e k ímpar é deixado como exercício para o leitor. □

Exemplo 24. Abaixo temos, respectivamente, os diagramas dos grafos $H_{8,4}$, $H_{8,5}$ e $H_{9,5}$.



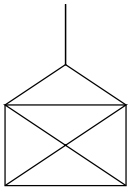
3.3 Grafos eulerianos e grafos hamiltonianos

Em 1796 Euler resolveu o problema conhecido como as *sete pontes de Königsberg* no que é considerando o primeiro resultado em Teoria dos Grafos. O problema é decidir se é possível atravessar cada uma das pontes uma única vez e retornar ao ponto de partida (veja a figura na página 2). Em outras palavras, dado um grafo queremos saber se existe um passeio fechado onde toda aresta parece exatamente uma vez. Grafos que admitem tal passeio são ditos eulerianos.

Analogamente, podemos considerar o caso de vértices: decidir se um grafo com pelo menos três vértices contém um circuito gerador. Grafos que admitem uma resposta positiva são ditos hamiltonianos. Neste capítulo apresentamos essas famílias de grafos e veremos que os eulerianos são facilmente caracterizados por uma propriedade que pode ser testada eficientemente, enquanto que não é conhecida uma boa caracterização de grafos hamiltonianos.

3.3.1 Grafos eulerianos

É possível desenhar a figura abaixo sem tirar o lápis do papel e sem repetir nenhum traço?



Vamos reformular essa pergunta na terminologia de Teoria dos Grafos. Dizemos que num grafo $G = (V, E)$ uma seqüência alternada vértice–aresta–vértice que não repete arestas

$$v_0 e_1 v_1 \dots v_{k-1} e_k v_k, \tag{3.11}$$

com $e_i = \{v_{i-1}, v_i\} \in E$, para todo $i \in \{1, 2, \dots, k\}$, e $e_i \neq e_j$ para todo $i \neq j$, é chamada de **trilha**.

Observação 12. Num grafo a seqüência dada em (3.11) pode ser representado por

$$v_0 v_1 \dots v_{k-1} v_k,$$

com $\{v_{i-1}, v_i\} \in E$. No entanto, tudo o que é dito nesta seção vale também para multigrafos e, nesse sentido, $v_i e_i v_{i+1}$ é difirente de $v_i f_i v_{i+1}$ para e_i e f_i arestas distintas com extremos v_i e v_{i+1}.

Exemplo 25. O seguinte diagrama representa o grafo que modela o problema dado no começo da seção

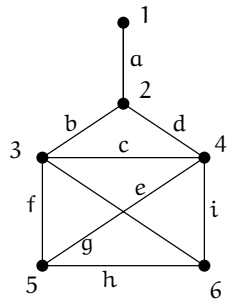


Figura 3.5: Exemplo: 1a2b3c4d2 e 4d2b3c4e5f3 são trilhas no grafo.

Se em (3.11) temos $v_0 = v_k$, então dizemos que a trilha é uma **trilha fechada**. Uma trilha que passa por todas as arestas do grafo é chamada de **trilha euleriana** e uma trilha fechada que passa por todas as arestas do grafo é chamada de **trilha euleriana fechada**. Um grafo que contenha uma trilha euleriana fechada é dito **grafo euleriano**.

Teorema 26 (Teorema de Euler, 1735)**.** *Um grafo conexo é euleriano se, e somente se, todo vértice tem grau par.*

Demonstração. Se um grafo conexo é euleriano, então todo vértice ocorre numa trilha euleriana fechada T; um vértice que ocorre k vezes na trilha T tem grau 2k.

Por outro lado, suponha que um grafo conexo tem todos os seus vértice de grau par. Seja $T = v_0e_1v_1 \dots v_{\ell-1}e_\ell v_\ell$ uma trilha com o maior número possível de arestas, com isso a trilha contém todas as arestas sujos extremos são os vértices v_0 e v_ℓ. Como todos os vértices têm grau par $v_0 = v_\ell$.

Agora, suponha que e é uma aresta que não aparece na trilha. Note que podemos assumir $e = uv_i$ para algum $v_i \in T$, pois o grafo é conexo. Mas, isso implica que $uev_ie_{i+1} \dots e_\ell v_\ell e_1v_1 \dots e_{i-1}v_i$ é uma trilha com mais arestas que T. Portanto, T passa por todas as arestas do grafo. □

Exercícios

Exercício 117. Mostre que se G tem no máximo dois vértices de grau ímpar então G contem uma trilha euleriana.

Exercício 118. Mostre que se G é euleriano então LG (veja exercício 16) é euleriano.

Exercício 119. Um grafo euleriano pode ter uma aresta de corte?

Exercício 120. Prove que um grafo conexo é euleriano se pode ser particionado em circuitos aresta-disjuntos.

Exercício 121. O algoritmo de Fleury, de 1883, descobre uma trilha euleriana fechada se o grafo dado for euleriano.

Algoritmo 15: Fleury(G)

Dado : um grafo euleriano G.
Devolve: um trilha euleriana fechada.

```
1 S ← ∅;
2 escolha u ∈ V(G);
3 enquanto E(G) ≠ ∅ faça
4     insira u no final de S;
5     escolha e = {u, w} ∈ E(G) e, se for possível, de forma que G − e tenha o mesmo número de
      componentes conexos que G;
6     u ← w;
7     G ← G − e.
8 devolva S.
```

Prove que se G é euleriano então a trilha definida pela seqüência vértices adjacentes $S = (u_1, u_2, \dots, u_\ell)$ construída pelo algoritmo é euleriana. Determine a complexidade do algoritmo.

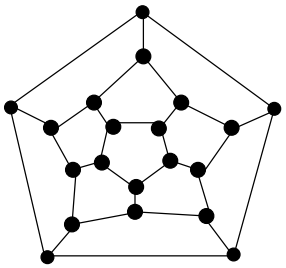
3.3.2 Grafos hamiltonianos

Um circuito $C \subseteq G$ é um **circuito hamiltoniano** em G se $V(C) = V(G)$. Um grafo que contém circuito hamiltoniano é chamado **grafo hamiltoniano**.

Exemplo 26. Um grafo que pode ser representado pelo seguinte diagrama é chamado de *dodecaedro*. O dodecaedro é um grafo hamiltoniano, em particular, foi o grafo que deu origem ao termo.

Uma condição necessária para um grafo ser hamiltoniano é que a remoção de um subconjunto de vértices resulta num número limitado de componentes conexos.

Proposição 27. *Se G é hamiltoniano, então o número de componentes conexos de* $G - S$ *é no máximo* $|S|$ *para todo* $S \subset V(G)$ *não-vazio.*



Demonstração. Basta notar que o número de componentes de $G - S$ é no máximo o número de componentes de $C - S$ que é no máximo $|S|$ para todo circuito hamiltoniano $C \subseteq G$. □

Vejamos um contra-exemplo para a recíproca do resultado acima. O grafo de Petersen, mostrado na figura abaixo, não é hamiltoniano. Pode-se verificar caso-a-caso que a remoção de qualquer subconjunto próprio e não-vazio de vértices S, o número de componentes do grafo obtido é sempre menor ou igual a $|S|$.

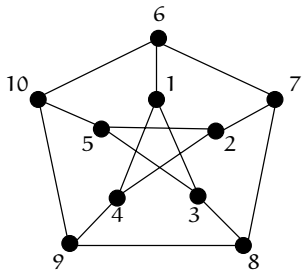


Figura 3.6: Grafo de Petersen.

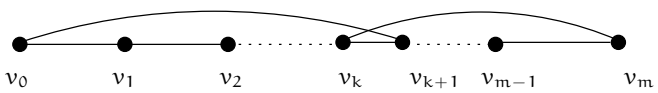
Uma condição suficiente para um grafo ser hamiltoniano é o seguinte resultado.

Teorema 28 (Teorema de Dirac, 1952)**.** *Para todo grafo* $G = (V, E)$ *com* $n \geq 3$ *vértices, se* $\delta(G) \geq n/2$ *então* G *é hamiltoniano.*

Demonstração. Seja $P = v_0, \ldots, v_m$ o maior caminho em G e defina os conjuntos

$$A = \{v_i \in V(P) : \{v_0, v_{i+1}\} \in E\} \quad \text{e} \quad B = \{v_j \in V(P) : \{v_0, v_j\} \in E\}. \tag{3.12}$$

Observamos que A e B são subconjuntos de $\{v_0, v_1, \ldots v_{m-1}\}$ com pelo menos $n/2 > (m-1)/2$ vértices e pelo Princípio da Casa dos Pombos $A \cap B \neq \varnothing$. Escolha $v_k \in A \cap B$ e considere o circuito



$C = v_0, v_{k+1}, \ldots, v_m, v_k, \ldots, v_0$.

Se C não é hamiltoniano então existem $u \in V(G) - V(C)$ e $v_j \in V(C)$ tais que $\{u, v_j\} \in E(G)$. Reescrevendo o circuito C acima como $u_0 = v_j, u_1, \ldots, u_m, u_0$ então temos o caminho $u, u_0, u_1, \ldots, u_m$ com uma aresta a mais que P, absurdo. Logo C é hamiltoniano. □

Claramente, nem todo grafo hamiltoniano tem grau mínimo alto, como contra-exemplo para a recíproca do resultado anterior basta considerar qualquer circuito com pelo menos 5 vértices, que é hamiltoniano e 2-regular.

Grafos hamiltonianos não são, até o momento, caracterizados por uma propriedade não-trivial que possa ser verificada eficientemente; decidir se um grafo é hamiltoniano é um problema NP-completo.

Exercícios

Exercício 122. Mostre que se G não é 2-conexo então G não é hamiltoniano.

Exercício 123. Prove que se G é bipartido com bipartição $V(G) = A \cup B$ onde $|A| \neq |B|$, então G não é hamiltoniano.

Exercício 124. Mostre que o n-cubo é hamiltoniano para todo $n \geq 2$.

Exercício 125. Um **caminho hamiltoniano** em G é um caminho que passa por todos os vértices de G. Mostre que se G contém um caminho hamiltoniano então o número de componentes conexos de $G - S$ é no máximo $|S| + 1$, para todo subconjunto próprio de vértices S.

Exercício 126. Prove que se G é tal que todo par de vértices u, v não-adjacentes vale $d(u) + d(v) \geq |V(G)| - 1$, então G contém um caminho hamiltoniano.

Exercício 127. Prove: $d(u) + d(v) \geq |V(G)|$ para todo par u, v de vértices não adjacentes, então G hamiltoniano se e somente se $G + uv$ hamiltoniano.

Exercício 128. Mostre que se G é um grafo euleriano então LG é hamiltoniano. Dê um exemplo onde a recíproca não vale.

Exercício 129. Dê um exemplo de grafo com n vértices, grau mínimo $\lfloor n/2 \rfloor$ e não-hamiltoniano, para todo $n \geq 3$.

ÁRVORES E FLORESTAS

Este capítulo apresenta a classe dos grafos que são conexos e, em certo sentido, minimais. Da equação (3.10), na página 49, podemos deduzir que um grafo de conexidade 1 sobre o conjunto de vértices V tem pelo menos $\lceil |V|/2 \rceil$ arestas. Não é uma tarefa difícil nos convencermos de que grafos com conexidade 1 e esse número mínimo de arestas não existem. Grafos conexos e com o menor número possível de arestas são estudados nesse capítulo.

Apresentamos também o problema da *árvore geradora de custo mínimo*: dado um grafo conexo com pesos nas arestas, determinar um um subgrafo gerador conexo minimal e, ainda mais, de custo total mínimo. Este problema é bastante conhecido e estudado há bastante tempo; apresentaremos dois algoritmos para o problema: o algoritmo de Kruskal, proposto em 1956 por Joseph Kruskal, e o algoritmo de Prim, de 1957 e devido a Robert Prim (o mesmo algoritmo foi inventado um tempo antes, em 1930, pelo matemático Tcheco Vojtěch Jarník). O primeiro algoritmo para esse problema de que se tem notícia é o algoritmo de Otakar Borůvka de 1926.

4.1 Árvores, definição e caracterização

Uma **floresta** é um grafo sem circuitos. Os componentes conexos de uma floresta são árvores, ou seja, uma **árvore** é um grafo conexo e sem circuitos.

Notemos que o grau mínimo de toda árvore com pelo menos três vértices é 1; isso porque se o grau mínimo fosse pelo menos 2 então a proposição 11 garante que o grafo teria circuito. Chamamos de **folha** todo vértice de grau 1 numa árvore.

Antes de enunciar e provar uma caracterização vamos vamos definir mais uma operação em grafo. Sejam G um grafo e $x, y \in V(G)$. Denotamos por $\mathbf{G + xy}$ o grafo

$$G + xy = \big(V(G), E(G) \cup \{\{x, y\}\}\big).$$

Teorema 29. *As seguintes afirmações são equivalentes para todo grafo* $G = (V, E)$*:*

(1) G *é árvore;*

(2) *para quaisquer* $x, y \in V$ *existe um único caminho em* G *com extremos* x *e* y*;*

(3) G *é conexo minimal:* G *é conexo e* $G - e$ *é desconexo, para qualquer* $e \in E$*;*

(4) G *é acíclico maximal:* G *é acíclico e* $G + xy$ *contém circuito, para quaisquer* $x, y \in V$ *não adjacentes em* G.

Demonstração. Para G trivial o teorema vale trivialmente. Se G tem ordem 2 então $G = K^2$ e a verificação é fácil, deixa-mo-lá para o leitor. Vamos supor que G tem pelo menos 3 vértices e vamos mostrar (1)⇒(2)⇒(3)⇒(4)⇒(1).

Que (1) ⇒ (2): Seja G uma árvore. Como G é conexos, existe um caminho $P = x, x_1, x_2, \ldots, x_n, y$. Suponha que exista um caminho $Q = x, y_1, y_2, \ldots, y_m, y$, $Q \neq P$. Defina $x_0 = y_0 = x$, $x_{n+1} = y_{m+1} = y$ e os índices

$$\begin{aligned} p &= \min\{i\colon i \geq 0 \text{ e } x_{i+1} \neq y_{i+1}\}, \text{ e} \\ q &= \min\{j\colon j > p \text{ e } x_j = y_\ell \text{ para algum } \ell > p\}. \end{aligned}$$

Esses índices estão bem definidos, como os caminhos são distintos $0 \leq p < \min\{m, n\}$ e $p < q \leq n+1$. Agora, $x_p, x_{p+1}, \ldots, x_q, y_{\ell-1}, y_{\ell-2}, \ldots, y_p$ é um circuito em G, uma contradição. Assim, o caminho com extremos x e y é único.

Que (2) ⇒ (3): Seja G tal que (2) vale. Por hipótese G é conexo. Para toda aresta $e = \{x, y\} \in E$ temos que $P = x, y$ é o único caminho com extremos x e y, portanto, $G - e$ é desconexo.

Que (3) $\Rightarrow$ (4): Seja G conexo minimal. Se G contém circuito, então para qualquer $e \in E(G)$ que pertença a um circuito temos que $G - e$ é conexo (proposição 19), uma contradição. Agora, seja $x, y \in V$ não adjacentes em G; como G é conexo existe um caminho, digamos $P = x, v_1, \ldots, v_k, y$, com extremos x e y. Em $G + xy$ temos o circuito $x, v_1, \ldots, v_k, y, x$.

Que (4) $\Rightarrow$ (1): Seja G um grafo acíclico maximal. Como G é acíclico só precisamos mostrar que é conexo. Observamos que se não existe caminho com extremos x e y, então $\{x, y\} \notin E(G)$ logo $G + xy$ não contém circuito, uma contradição.

Dessa forma, fica estabelecida a equivalência das proposições. □

Como conseqüência do teorema 29 temos o seguinte corolário.

Corolário 30. *Uma árvore com n vértices é um grafo conexo com o menor número possível de arestas dentre todos os grafos conexos com n vértices.* □

O número de arestas de caracteriza árvores. Por um lado temos o seguinte resultado.

Lema 31. *Toda árvore com n vértices tem $n - 1$ arestas.*

Demonstração. Vamos provar por indução em $n \geq 1$ que a seguinte afirmação vale: *se um grafo com n vértices é uma árvore então o número de arestas é* $n - 1$. Se o grafo é trivial, isto é $n = 1$, então o número de arestas é 0. Dado $n \geq 2$, vamos assumir que toda árvore com $n - 1$ vértices tem $n - 2$ arestas e seja G uma árvore com n vértices. Tome $v \in V(G)$ uma folha de G. O grafo $G - v$ é uma árvore (justifique) com $n - 1$ vértices e pela hipótese indutiva $|E(G - v)| = n - 2$. Como $|E(v)| = 1$ e $|E(G - v)| = |E(G) \setminus E(v)|$, temos que $|E(G)| = n - 1$. Portanto, pelo Princípio de Indução Finita, a afirmação vale para todo $n \geq 1$. □

Antes de provar a recíproca desse resultado, introduziremos alguns conceitos.

Se o subgrafo gerador $T \subseteq G$ é uma árvore então chamamos T de **árvore geradora** de G.

Todo grafo conexo tem uma árvore geradora: Seja G um grafo conexo e seja $T \subseteq G$ um subgrafo gerador de G, conexo e com o menor número de arestas possível. Se T contém circuito então para qualquer aresta e de um circuito $T - e$ é conexo e tem menos arestas que T, uma contradição.

Desse fato, podemos concluir que a recíproca do lema 31: suponha que G é conexo, com n vértices e $n - 1$ arestas. Seja $T \subseteq G$ uma árvore geradora de G. Como acabamos de mostrar T tem $n - 1$ arestas, logo $T = G$. Decorre dessa igualdade que G é uma árvore.

Teorema 32. *Um grafo com n vértices é uma árvore se, e somente se, é conexo e o número de arestas é* $n - 1$. □

Um algoritmo que recebe um grafo conexo e determina uma árvore geradora do grafo conexo é fácil e rápido e já sabemos como fazer: é o subgrafo induzido pelo conjunto de arestas $M = \{\{v, pai[v]\}: v \in V(G) \text{ e } pai[v] \neq nil\}$, onde o vetor $pai[\,]$ é determinado pela execução do algoritmo 4(veja os exercícios 68 e 82).

Exemplo 27. Um grafo conexo e uma árvore geradora determinada por uma busca em profundidade $BP(G, 1)$, algoritmo 4 são mostrados na figura 4.1 abaixo.

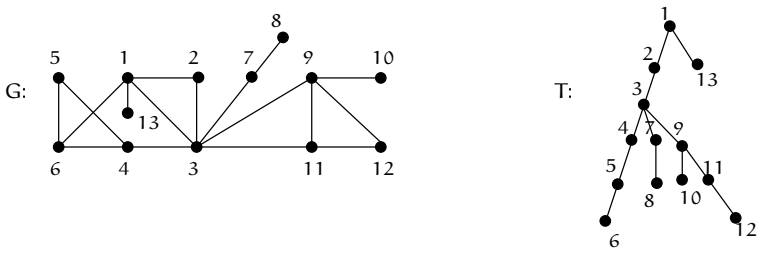


Figura 4.1: T é a árvore geradora de G definida pelo vetor pai de uma busca em profundidade, algoritmo 4.

Exercícios

Exercício 130. Determine o número de arestas de uma floresta com n vértices e com k componentes conexos, em função de n e k.

Exercício 131. Desenhe todas as árvores não-isomorfas com 5 vértices e todas as árvores não-isomorfas com 7 vértices e com grau máximo pelo menos 4.

Exercício 132. Mostre que toda árvore T tem pelo menos $\Delta(T)$ folhas.

Exercício 133. Prove que o conjunto de arestas $\{pai[v], v\}$ definido por uma busca em profundidade rotulada induz uma árvore.

Exercício 134. Seja G um grafo. Mostre que as seguintes afirmações são equivalentes:

(a) G é conexo e $|E(G)| = |V(G)| - 1$;
(b) $|E(G)| = |V(G)| - 1$ e G não contém circuito.
(c) G é uma árvore;

Exercício 135. Considere uma floresta F com n vértices e m arestas.

(a) Qual é o número máximo de vértices num componente conexo de F?
(b) Mostre que há pelo menos $\max\{n - 2m, 0\}$ vértices de grau zero.
(c) Mostre que se $m < n/2$ então resta pelo menos um vértice de grau zero.

Exercício 136. Seja T uma árvore, u e v dois vértices não adjacentes em T e $e \in E(T)$ uma aresta do circuito em $T + uv$. Mostre que

$$\mathbf{T + uv - e} = \big(V(T), E(T) \cup \big\{\{u, v\}\big\} \setminus \{e\}\big)$$

é uma árvore.

Dizemos que essa árvore foi obtida por uma **operação elementar** a partir da árvore T.

Exercício 137. Sejam $T_1, T_2 \subset G$ árvores geradoras distintas de um grafo conexo G. Mostre que T_2 pode ser obtida a partir de T_1 através de uma seqüência de operações elementares.

Exercício 138. Prove que se uma aresta é de corte (veja exercício 98) num grafo G então ela pertence a todas as árvores geradoras de G.

Exercício 139. Seja T uma árvore de ordem t. Mostre que qualquer grafo com grau mínimo pelo menos $t - 1$ contém um subgrafo isomorfo a T.

Exercício 140. Mostre que se G é um grafo conexo então o grafo de blocos B_G (exercício 114) é uma árvore.

Exercício 141. Seja G um grafo conexo com n vértices e m arestas, $m \geq n$. Se $T \subset G$ é uma árvore geradora de G então o número de arestas de G que não estão em T é $m - n + 1$. Denotaremos por $e_1, e_2, \ldots, e_{m-n+1}$ tais arestas em $E(G) \setminus E(T)$. Em $T + e_i$ temos um circuito que denotamos por C_i, para todo $1 \leq i \leq m - n + 1$. Chamaremos $C_1, C_2, \ldots, C_{m-n+1}$ de *circuitos fundamentais de* G com respeito a T. Definimos $0 \cdot C_i = \varnothing$ e $1 \cdot C_i = E(C_i)$. Mostre que para todo circuito $C \subset G$ existe uma seqüência $(b_1, b_2, \ldots, b_{m-n+1}) \in \{0, 1\}^{m-n+1}$ tal que $E(C)$ é dado por

$$b_1 \cdot C_1 \oplus b_2 \cdot C_2 \oplus \cdots \oplus b_{m-n+1} \cdot C_{m-n+1},$$

ou seja, todo circuito é combinação linear de circuitos fundamentais. Defina um espaço vetorial a partir das informações acima.

4.2 Árvores geradoras de custo mínimo em grafos com pesos nas arestas

Dado um grafo conexo com pesos nas arestas $G = (V, E, \rho)$, onde $\rho\colon E \to \mathbb{R}$, definimos o **custo de um subgrafo** $H \subset G$ como

$$c(H) = \sum_{e \in E(H)} \rho(e).$$

O problema no qual estamos interessados a partir de agora é: qual é o custo do subgrafo gerador conexo de G "mais barato"? Em outras palavras, queremos determinar $S \subseteq E(G)$ que induz uma árvore geradora de G e tal que

$$c(G[S]) = \min\{c(T)\colon T \text{ é árvore geradora de } G\}.$$

Uma árvore geradora de G de custo mínimo também é chamada de **árvore geradora mínima** do grafo G.

A seguir apresentamos os algoritmos de Jarník–Prim e Kruskal para o problema de determinar uma árvore geradora mínima. Os algoritmos de Jarník–Prim e Kruskal são algoritmos gulosos, que é uma técnica de projeto de algoritmos para resolver problemas de otimização, esses algoritmos baseam-se na escolha que parece ser a melhor no momento (ótimo local) e terminam com a solução ótima (ótimo global).

4.2.1 Algoritmo de Jarník–Prim para árvore geradora mínima

O algoritmo de Jarník–Prim recebe um grafo conexo $G = (V, E, \rho)$ e devolve $S \subseteq E$ tal que $G[S]$ é uma árvore geradora mínima de G. O algoritmo parte de um conjunto unitário, digamos $U = \{v\}$ para algum $v \in V$, e a cada rodada o algoritmo acrescenta a U o vértice de $\overline{U}$ que é extremo da aresta de menor custo em $E(U, \overline{U})$, até que $U = V$. No final, as arestas escolhidas induzem uma árvore geradora mínima.

Algoritmo 16: Jarník–Prim(G)

Dado : um grafo conexo G com pesos nas arestas.
Devolve: árvore geradora de custo mínimo.

```
1 escolha v ∈ V(G);
2 U ← {v};
3 S ← ∅;
4 enquanto U ≠ V(G) faça
5     escolha {u, w} ∈ E(U, Ū) de peso mínimo no corte;
6     insira {u, w} em S;
7     v ← {u, w} ∩ Ū;
8     insira v em U;
9 devolva (V, S).
```

Notemos que esse algoritmo é uma ligeira modificação do algoritmo 3 (pela proposição 18 os laços desses dois algoritmos têm a mesma condição) e, como sabemos, S induz uma árvore geradora de G. Entretanto, não sabemos se tal árvore é mínima.

Análise e correção do algoritmo de Jarník–Prim. Como no correção do algoritmo de Dijkstra, usaremos a técnica do invariante do laço.

Teorema 33. *Para $k \in \mathbb{N}$, após k iterações do enquanto na linha 4 vale que (U, S) é uma subárvore de uma árvore geradora mínima de G.*

Demonstração. Pela construção do algoritmo, o par (U, S) é um grafo. Denote por S_k e U_k os conjuntos S e U mantidos pelo algoritmo após a k-ésima iteração do laço na linha 4. A prova é por indução em k.

Para $k = 0$ a afirmação no enunciado vale pois $S_0 = \varnothing$ e $U_0 = \{v\}$ e (U_0, S_0) é subárvore de toda árvore geradora mínima de G.

Suponha que (U_{k-1}, S_{k-1}) é uma subárvore da árvore geradora mínima $T_{\min}$. Sejam $w \in V(G)$ e $\{u, w\} \in E(G)$ tais que $U_k = U_{k-1} \cup \{w\}$ e $S_k = S_{k-1} \cup \{\{u, w\}\}$. Vamos mostrar que (U_k, S_k) é uma subárvore de alguma árvore geradora mínima de G.

Que (U_k, S_k) é uma árvore deixamos para a verificação do leitor. Se $\{u, w\} \in E(T_{\min})$ então $T_{\min}$ contém (U_k, S_k), logo podemos supor que $\{u, w\} \notin E(T_{\min})$.

Pelo teorema 29 o grafo $T_{min} + uw$ contém um circuito. Esse circuito tem uma aresta $e \in E_{T_{min}}(U_{k-1}, \overline{U_{k-1}})$ pois $u \in U_{k-1}$ e $w \in \overline{U_{k-1}}$, logo o único caminho em T_{min} com extremos u e w deve conter uma aresta desse corte, portanto $T_{min} + uw - e$ é árvore geradora de G. Pela escolha do algoritmo na linha 5 $\rho(\{u, w\}) \leq \rho(e)$ e como T_{min} tem custo mínimo $T_{min} + uw - e$ é uma árvore geradora de custo mínimo que contém T_k. O teorema segue do Princípio da Indução Finita. □

Como a cada rodada do laço um vértice é inserido em U, que começa com um vértice, o laço será executado $|V| - 1$ vezes, após o que $U = V$, portanto (U, S) é árvore geradora mínima de G. Com isso terminamos a prova de correção do algoritmo de Jarník–Prim e passamos a estudar a complexidade desse algoritmo.

A complexidade de tempo do algoritmo de Jarník–Prim depende da implementação do passo 5 do algoritmo. Se ordenamos as arestas o custo é $O(|E| \log |E|)$ e cada busca na linha 5 gasta $O(|E|)$. Assim o custo total é $O(|E| \log |E|) + O(|V||E|) = O(|V||E|)$. Tomando um pouco mais de cuidado podemos melhorar o tempo do algoritmo significativamente. Agora, vamos mostrar uma implementação do algoritmo de Jarník–Prim de complexidade $O(|E| \log |V|)$. Vejamos como fica usando uma fila de prioridades implementada por uma *heap* binária. O algoritmo usa dois vetores indexados por V e uma fila de prioridades:

$chave[\,]$ em cada instante da execução do algoritmo $chave[v]$ armazena o peso da aresta de menor peso em $E(U, \overline{U}) \cap E(v)$;

$pai[\,]$ as arestas $\{v, pai[v]\}$ são as arestas escolhidas pelo algoritmo;

Q fila de prioridades indexada por $\overline{U}$ e com chave $chave[\,]$ — quanto menor o valor de $chave$ maior a prioridade.

Algoritmo 17: Jarník–Prim(G)

Dado : um grafo conexo G com pesos nas arestas.
Devolve: árvore geradora de custo mínimo.

```
1  escolha v ∈ V(G);
2  para cada u ∈ V(G) faça
3      chave[u] ← ∞;
4      pai[u] ← nil;
5  chave[v] ← 0;
6  U ← ∅;
7  construa uma fila de prioridades Q indexada por V(G) e com prioridades chave[ ];
8  enquanto Ū ≠ ∅ faça
9      u ← extrai mínimo de Ū;
10     para cada t ∈ N(u) faça
11         se t ∈ Ū e ρ({u, t}) < chave[t] então
12             chave[t] ← ρ({u, t});
13             pai[t] ← u.
```

As 7 primeiras linhas são executadas em tempo $O(|V|)$, incluindo-se aí o tempo para construir a fila de prioridades. A análise agora segue como exatamente como no caso do algoritmo de Dijkstra, resultando em $O((|V| + |E|) \log |V|) = O(|E| \log |V|)$, pois o grafo é conexo.

Teorema 34. *A complexidade do algoritmo* Jarník–Prim(G) *é* $O(|E(G)| \log |V(G)|)$. □

4.2.2 Algoritmo de Kruskal para árvore geradora mínima

A idéia do algoritmo de Kruskal também é bastante simples, a cada passo escolhemos a aresta mais barata dentre as que ainda não foram escolhidas, desde que ela não forme circuito com as arestas

já escolhidas:

Algoritmo 18: Kruskal(G)

Dado : um grafo conexo G com pesos nas arestas.
Devolve: árvore geradora de custo mínimo.

```
1 S ← ∅;
2 F ← fila das arestas em ordem não-decrescente de peso;
3 para cada e ∈ F faça
4     se S ∪ {e} não induz circuito em G então
5         insira e em S;
6 devolva (V, S).
```

Análise e correção do algoritmo de Kruskal. Vamos provar que o algoritmo de Kruskal determina uma árvore geradora de custo mínimo de um grafo conexo e com pesos nas arestas.

Teorema 35. *O grafo* (V, S) *devolvido por* Kruskal(G) *é uma árvore geradora mínima de* G.

Demonstração. Dado G conexo e com pesos nas arestas, sejam S o conjunto construído pelo algoritmo e T uma árvore geradora mínima de G com o maior número possível de arestas em comum com S. Vamos mostrar que $E(T) = S$.

Suponha $S \setminus E(T) \neq \varnothing$ e defina $m = \min\{j : e_j \in S \setminus E(T)\}$, onde o mínimo é com relação a seqüência $(e_1, e_2, \ldots, e_m)$ em que as arestas foram inseridas no conjunto S pelo algoritmo.

Por definição $T + e_m$ contém um circuito e nesse circuito deve existir uma aresta f que não está em S. O conjunto de arestas $\{e_1, \ldots, e_{m-1}\} \cup \{f\}$ está em T, portanto, não induz circuito, mas como f não foi escolhida pelo algoritmo de Kruskal tem-se $\rho(f) \geq \rho(e_m)$.

Logo $c(T - f + e_m) \leq c(T)$ e como T é de custo mínimo também $T - f + e_m$ é de custo mínimo e, ainda, com mais arestas em comum com S que T, contrariando a escolha de T, portanto $S \setminus E(T) = \varnothing$, ou seja $S \subseteq E(T)$.

Para concluir, $E(T) \setminus S = \varnothing$ pois, dado que $S \subseteq E(T)$, qualquer $e \in E(T)$ satisfaz a condição da linha 4 do algoritmo. □

Notemos que enquanto o conjunto S evolui de $\varnothing$ até definir uma árvore geradora ele sempre define uma subfloresta de G, ou seja, uma coleção de subconjuntos disjuntos de vértices (os componentes conexos) e o teste na linha 4 acima verifica se a aresta $e = \{x, y\}$ não liga vértices do mesmo conjunto (caso contrário, teremos um circuito) e nesse caso, na linha 5, une o conjunto onde está o vértice x com o conjunto do vértice y. Assim, precisamos de estruturas de dados que, dinamicamente, representem e manipulem conjuntos disjuntos de modo eficiente. Estruturas como essa são conhecidas na literatura como estruturas para união-e-busca (*union-find*) ou *estruturas de dados para conjuntos disjuntos* (veja apêndice A.1). Essas estruturas mantém dinamicamente uma família de subconjuntos disjuntos com um elemento de cada subconjunto eleito como o *representante* do subconjunto e temos as operações

`faz(x)` cria o conjunto unitário $\{x\}$, com representante x;

`busca(x)` devolve o representante do conjunto ao qual x pertence;

`união(x, y)` substitui os conjuntos que contém x e y pela união desses conjuntos (e determina um representante para essa união).

Dentre as estruturas de dados mais eficientes para implementar conjuntos disjuntos, chamamos a atenção para uma delas: a representação por floresta com as heurísticas *união por* rank e *busca com compressão de caminhos* (veja A.1.1, página 91). Essa representação com as heurísticas mencionadas tem um excelente desempenho assintótico.

Denotamos por $\lg^*(n)$ o número de vezes que temos que iterar a função lg até que o valor obtido seja menor ou igual a 1, por exemplo, $\lg^* 2^{16} = 4$. Estimativas apontam que o número de átomos no universo observável é 10^{80} e $\lg^* 10^{80} = 4$.

O seguinte resultado é demonstrado no apêndice, teorema 67 na seção A.1.1.

Teorema 36. *A complexidade de* m *operações* faz/busca/união, *das quais as* n *primeiras são* faz, *onde* n *é o número de elementos do conjunto universo, é* $O((m+n)\lg^*(n))$. □

Reescrevendo o algoritmo de Kruskal com as considerações feitas acima sobre estruturas de dados para conjuntos disjuntos

Algoritmo 19: Kruskal(G)

Dado : um grafo conexo G com pesos nas arestas.
Devolve: árvore geradora de custo mínimo.

```
1 S ← ∅;
2 F ← fila das arestas em ordem não-decrescente de peso;
3 para cada v ∈ V(G) faça faz(v);
4 para cada {u, v} ∈ F faça
5     se busca(u) ≠ busca(v) então
6         insira {u, v} em S;
7         união(u, v);
8 devolva (V, S).
```

Nesse caso, o número de operações faz, busca e união no algoritmos de Kruskal é menor que $2(|E| + |V|)$. A complexidade de tempo do algoritmo de Kruskal é $O((|E| + |V|)\lg^* |V|)$ para as operações (teorema 36) mais $O(|E| \log |E|)$ para a ordenação das arestas; resultando

Corolário 37. *A complexidade de* Kruskal(G) *é* $O(|E| \log |E|)$.

Exercícios

Exercício 142. Seja G um grafo conexo com pesos na arestas. Prove que se e é uma aresta de peso mínimo em G então e pertence a alguma árvore geradora mínima de G.

Exercício 143. Seja G um grafo conexo com pesos na arestas. Prove que se e é uma aresta de peso máximo em G e pertence a um circuito de G então existe uma árvore geradora mínima de $(V(G), E(G) \setminus \{e\})$ que também é uma árvore geradora mínima de G. Mostre que o mesmo vale para toda aresta de peso máximo de todo circuito de G.

Exercício 144. Seja G um grafo conexo com pesos na arestas. Prove que para qualquer $U \subsetneq V(G)$ não-vazio, a aresta de menor custo em $E(U, \overline{U})$ tem que pertencer a toda árvore geradora de G.

Exercício 145. Seja $T \subset G$ uma árvore geradora de um grafo G com pesos nas arestas. Mostre que T é uma árvore geradora mínima se e somente se para toda $e \in E(G) \setminus E(T)$, o único circuito C de $T + e$ é tal que todas as arestas de C custam não mais que $\rho(e)$.

Exercício 146. Mostre que se para todo corte em G existe uma única aresta de custo mínimo no corte, então a árvore geradora de custo mínimo de G é única. A recíproca dessa afirmação vale? Justifique.

Exercício 147. Seja G um grafo conexo com pesos positivos nas arestas. Para toda árvore geradora mínima T e todo caminho $P \subset T$, P é um caminho mínimo em G?

Exercício 148. Mostre que para toda árvore geradora mínima $T \subset G$ existe uma ordenação nas arestas de G tal que T é a árvore devolvida pelo algoritmo de Kruskal.

Exercício 149. Suponha que todos os pesos das arestas de G são positivos. Mostre que qualquer subconjunto de arestas que induz um subgrafo conexo e tem peso total mínimo é uma árvore. Dê um exemplo com pesos não-positivos onde a conclusão não vale.

Exercício 150. Considere a seguinte estratégia genérica para computar uma árvore geradora de custo mínimo. Seja $G = (V, E, \rho)$ um grafo conexo com pesos nas arestas. Dado $S \subset E(G)$, dizemos que uma aresta de $\{u, v\} \in E(G)$ é **boa para** S se $S \cup \big\{\{u, v\}\big\} \subset E(T)$ para alguma árvore geradora mínima T de G.

Algoritmo 20: AGM(G)

Dado : um grafo conexo G com pesos nas arestas.
Devolve: árvore geradora de custo mínimo.

```
1 S ← ∅;
2 enquanto G[S] não é árvore geradora faça
3     escolha uma aresta f boa para S em E(G) \ S;
4     insira f em S;
5 devolva (V, S).
```

(a) Primeiro, suponha que sempre existe pelo menos uma aresta boa pra ser escolhida em cada iteração do enquanto e que um *oráculo* entrega essa aresta para algoritmo sempre que a linha 3 é executada. Prove que AGM(G) *constrói uma árvore geradora mínima de* G. (Dica: determine e prove um invariante para o laço.)

(b) Agora, vamos provar que sempre há uma aresta boa pra ser escolhida pelo oráculo. Prove que *se* T *é uma árvore geradora mínima de* G *e* $S \subset E(T)$ *então para todo* $U \subset V(G)$ *tal que o corte* $E(U, \overline{U})$ *não contém arestas de* S*, uma aresta* $\{u, v\}$ *de custo mínimo no corte* $E(U, \overline{U})$ *é boa para* S. (Dica: dados G, S, T como no enunciado, tome U e $e \in E(U, \overline{U})$ como enunciado e considere 2 casos, se $e \in E(T)$ e se $e \notin E(T)$.)

(c) Por fim, use o resultado do exercício acima para dar uma prova da correção dos algoritmos de Jarník–Prim e Kruskal, isto é prove que esses algoritmos sempre escolhem arestas boas.

EMPARELHAMENTOS

Emparelhamento em um grafo nada mais é do que um conjunto independente de arestas; é um dos tópicos mais estudados da Teoria dos Grafos devido a ampla variedade de aplicações (uma referência excelente para emparelhamentos é [10]). Neste capítulo o conceito de emparelhamento é apresentado na primeira seção e na segunda seção a ênfase é dada ao caso especial de emparelhamentos em grafos bipartidos. Por fim, tratamos de aspectos algorítmicos em grafos bipatidos.

5.1 Emparelhamento

Num grafo $G = (V, E)$, dizemos que $M \subseteq E$ é um **emparelhamento** se as arestas de M são duas-a-duas não-adjacentes, ou seja, $e \cap f = \varnothing$ para quaisquer $e, f \in M$. De modo equivalente, chamamos M de emparelhamento se $G[M]$ é um subgrafo 1-regular.

Exemplo 28. Considere o grafo dado pelo diagrama da figura 5.1 abaixo. O conjunto $M = \{\{2,4\},\{5,6\}\}$ é um emparelhamento; note que não há arestas do emparelhamento com extremo nos vértices 1 e 3.

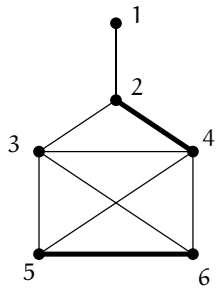


Figura 5.1: $M = \{\{2,4\},\{5,6\}\}$ é um emparelhamento no grafo representado acima.

Seja G um grafo e M um emparelhamento em G. Quando um vértice $v \in V(G)$ pertence a alguma aresta $e \in M$, dizemos que v é **coberto** por M. Dessa forma, pela definição de emparelhamento, quando v é coberto por M existe uma única aresta $e \in E(v) \cap M$. Se todo elemento de $V(G)$ é coberto por alguma aresta de M, então M é chamado de **emparelhamento perfeito** em G.

Há grafos que não admitem emparelhamento perfeito como, por exemplo, os circuitos de comprimento ímpar; de fato, qualquer emparelhamento em C^ℓ tem no máximo $\lfloor \ell/2 \rfloor$ arestas, logo circuitos de comprimento par admitem emparelhamento perfeito, mas os de comprimento ímpar não. No exemplo da figura 5.1 o emparelhamento $\{\{1,2\},\{4,5\},\{3,6\}\}$ é perfeito, assim como o emparelhamento $\{\{1,2\},\{3,5\},\{4,6\}\}$.

Um emparelhamento em G com o maior número possível de arestas é chamado de **emparelhamento máximo**, isto é, um emparelhamento com

$$\mu(\mathbf{G}) = \max\Big\{|M|\text{: } M \text{ é emparelhamento em } G\Big\} \tag{5.1}$$

arestas.

Exemplo 29. No caso dos circuitos C^ℓ é fácil ver que $\mu(C^\ell) = \lfloor \ell/2 \rfloor$. O mesmo vale para grafos completos $\mu(K^n) = \lfloor n/2 \rfloor$. No caso de caminhos, $\mu(P^\ell) = \lfloor \ell/2 \rfloor$ para todo $\ell > 0$.

No estudo de emparelhamentos em grafos surge um tipo especial de caminho, onde as arestas alternam entre aresta do emparelhamento e aresta fora do emparelhamento. Um caminho $P =$

$x_1, x_2, \ldots, x_k$, para $k \geq 1$, em G cujas arestas alternam entre as arestas de $E(G) \setminus M$ e as arestas de M isto é,

$$\begin{cases} \{x_i, x_{i+1}\} \notin M \text{ se } i \text{ é ímpar e} \\ \{x_i, x_{i+1}\} \in M \text{ se } i \text{ é par,} \end{cases}$$

para todo $i \in \{1, 2, \ldots, k-1\}$, é chamado de **M-alternante**.

Se os extremos de um caminho M-alternante *não* são cobertos por M então chamamos esse caminho de **M-aumentante**. Como o nome sugere, a existência de um caminho M-aumentante P em G significa que podemos obter um emparelhamento em G com mais arestas que M.

Exemplo 30. Na figura 5.1 temos $M = \{\{2,4\},\{5,6\}\}$ e o caminho M-aumentante P com $E(P) = \{\{1,2\},\{2,4\},\{4,5\},\{5,6\},\{6,3\}\}$. A diferença simétrica desses conjuntos é

$$\begin{aligned} &\left(\{\{2,4\},\{5,6\}\} \cup \{\{1,2\},\{2,4\},\{4,5\},\{5,6\},\{6,3\}\}\right) \setminus \left(\{\{2,4\},\{5,6\}\} \cap \{\{1,2\},\{2,4\},\{4,5\},\{5,6\},\{6,3\}\}\right) \\ &\quad = \{\{1,2\},\{4,5\},\{6,3\}\} \end{aligned}$$

que é um emparelhamento com uma aresta a mais que M.

Proposição 38. *Sejam* G *um grafo,* M *um emparelhamento em* G *e* P *um caminho* M*-aumentante. A diferença simétrica* $M \triangle E(P) = (M \cup E(P)) \setminus (M \cap E(P))$ *é um emparelhamento em* G *com uma aresta a mais que* M*.*

Demonstração. Sejam G, M e P como no enunciado. De P é M-aumentante

$$\begin{aligned} |M \triangle E(P)| &= |(M \cup E(P)) \setminus (M \cap E(P))| \\ &= \left|\left(M \setminus (M \cap E(P))\right) \cup \left(E(P) \setminus (M \cap E(P))\right)\right| \\ &= |M| - |M \cap E(P)| + |E(P)| - |M \cap E(P)| \\ &= |M| - |M \cap E(P)| + |M \cap E(P)| + 1 \\ &= |M| + 1. \end{aligned}$$

Resta provar que $M \triangle E(P)$ é emparelhamento. Suponha que não, então existem duas arestas de $M \triangle E(P)$ adjacentes, digamos que $e, f \in M \triangle E(P)$ com $e \cap f = x$. Dessa forma, x deve ser um vértice interno em P, portanto, deve estar coberto por uma aresta $g \in M$, logo $x \in e \cap g$ e ambas arestas estão em M, absurdo. □

O seguinte teorema é uma caracterização de emparelhamento máximo em função dos caminhos aumentantes. Esse resultado é fundamental no projeto de um algoritmo eficiente que determina emparelhamentos máximos; mais adiante veremos esse algoritmo para grafos bipartidos.

Teorema 39 (Teorema de Berge, 1957)**.** *Um emparelhamento* M *em* G *é máximo se, e somente se,* G *não contém caminho* M*-aumentante.*

Demonstração. Vamos mostrar que se um emparelhamento não é máximo então há um caminho aumentante. A recíproca dessa afirmação é a proposição acima.

Seja M um emparelhamento que não é máximo e M^* um emparelhamento máximo. Considere o subgrafo induzido pelas arestas dos dois emparelhamentos $H = G[M \cup M^*]$. Como $\Delta(H) \leq 2$ os componentes conexos de H são circuitos e caminhos (exercício 99).

Os circuitos têm que ser de comprimento par, por definição de emparelhamento. Se todos os caminhos tiverem comprimento par então teremos $|M| = |M^*|$, logo existe um caminho P de comprimento ímpar e com mais arestas de M^* o que implica que os extremos do caminho não são cobertos por M. Esse caminho é M-aumentante em G. □

Exercícios

Exercício 151. Prove que uma árvore qualquer tem no máximo um emparelhamento perfeito.

Exercício 152. Considere um grafo G de n vértices. Mostre que um emparelhamento em G tem no máximo $n/2$ arestas.

Exercício 153. Mostre que o k-cubo admite emparelhamento perfeito, para todo $k \geq 2$.

Exercício 154. Duas pessoas jogam um jogo sobre um grafo G alternadamente selecionando vértices distintos $v_0, v_1, v_2, \ldots$ tais que, para $i > 0$, v_i é adjacente a v_{i-1}. O último jogador que conseguir selecionar um vértice vence o jogo.

Mostre que o primeiro jogador tem uma estratégia para vencer o jogo se e somente se o grafo G não tem um emparelhamento perfeito.

Exercício 155. Mostre que $\mu(G) = \alpha(LG)$ (veja a definição de LG no exercício 16).

Exercício 156. Prove que se M é emparelhamento em G então existe um emparelhamento máximo que cobre todos os vértices cobertos por M.

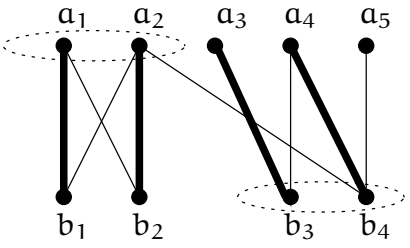
Exercício 157 (Teorema de Tutte, 1947). Denotemos $c_i(G)$ o número de componentes conexos do grafo G com um número ímpar de vértices. Mostre que se G tem um emparelhamento perfeito se, e somente se, $c_i(G - S) \leq |S|$ para todo conjunto S de vértices.

5.2 Emparelhamentos e coberturas em grafos bipartidos

Por toda esta seção adotaremos as seguintes convenções: as partes de vértices independentes de um grafo bipartido G são denotadas por A e B e escrevemos $G = (A \cup B, E)$, também, convencionamos que caminhos M-alternante têm um extremo descoberto em A.

Uma **cobertura por vértices** em um grafo G (não necessariamente bipartido) é um subconjunto $U \subseteq V(G)$ tal que $e \cap U \neq \varnothing$ para toda aresta $e \in E$, ou seja, toda aresta de G encontra U.

Exemplo 31. No exemplo da figura abaixo temos o diagrama de um grafo bipartido, as arestas em destaque definem um emparelhamento M. O subconjunto de vértices $\{a_1, a_2, b_3, b_4\}$ é uma cobertura pois *todas* as arestas do grafo tem pelo menos um desses vértices como extremo.



Uma **cobertura mínima** é uma cobertura com o menor número possível de vértices que é denotado por $\nu(G)$,

$$\nu(G) = \min\Big\{|U|\colon U \text{ é cobertura por vértices em } G\Big\}. \tag{5.2}$$

Teorema 40 (Teorema de König, 1931). *Num grafo bipartido* $G = (A \cup B, E)$ *o tamanho de um emparelhamento máximo é igual ao tamanho de uma cobertura mínima, ou seja*

$$\mu(G) = \nu(G). \tag{5.3}$$

Demonstração. Para qualquer cobertura U e qualquer emparelhamento M vale que toda aresta de M tem que ter pelo menos um extremo em U, portanto $|M| \leq |U|$. Em particular,

$$\mu(G) \leq \nu(G). \tag{5.4}$$

Agora, considere M um emparelhamento máximo em G e vamos construir uma cobertura $C \subseteq V(G)$ da seguinte maneira: para cada aresta $\{a, b\} \in M$, onde $a \in A$ e $b \in B$, escolhemos b para

C se b é extremo de algum caminho M-alternante, caso contrário, escolhemos a. Note que dessa forma temos $|C| = |M| = \mu(G)$.

Vamos mostrar que C é uma cobertura. Considere uma aresta qualquer $\{a, b\} \in E(G)$, onde $a \in A$ e $b \in B$, e vamos mostrar que essa aresta encontra C. Se $\{a, b\} \in M$ então a aresta encontra C. Vamos supor que $\{a, b\}$ não é aresta de M. Como M é máximo, ou a ou b ou ambos são cobertos por M.

Se M não cobre a então b pertence ao um caminho M-alternante $P = a, b$ e como b é coberto por M, nesse caso, $b \in C$.

Vamos supor que M cobre a e $\{a, b'\} \in M$ para algum $b' \in B$, $b' \neq b$. Se $a \in C$ então não há o que provar. Agora, se $a \notin C$ então $b' \in C$, mas isso significa que b' está no extremo de algum caminho M-alternante que denotamos por P. Se b está em P então b é extremo de um caminho M-alternante e com isso $b \in C$. Caso contrário, também haverá um caminho M-alternante com extremo b: ou P, a, b no caso $a \notin V(P)$, ou P', b caso $a \in V(P)$, onde P' denota um subcaminho alternante de P com um extremo em a. Em ambos os casos teremos $b \in C$ pois, como M é máximo, não pode haver caminho M-aumentante em G. Em todos os casos a aresta $\{a, b\}$ encontra C, portanto, C é cobertura. Logo $|C| \geq \nu(G)$ e

$$\mu(G) \geq \nu(G). \tag{5.5}$$

Das equações (5.4) e (5.5) concluímos que $\nu(G) = \mu(G)$. □

Observação 13 (Cobertura em grafos não-bipartidos). O teorema de König não vale para grafo não-bipartido, como mostra o K^3, por exemplo, nele $\mu(K^3) = 1$ enquanto que $\nu(K^3) = 2$. Computar a cobertura mínima em grafos não -bipartidos é um problema NP-difícil (exercício 171) enquanto que μ pode ser computado em tempo polinomial (veja [7]).

O seguinte resultado dá uma condição necessária e suficiente para que exista um emparelhamento que cobre umas das partes de um grafo bipartido.

Teorema 41 (Teorema de Hall, 1935)**.** *Em todo grafo bipartido* $G = (A \cup B, E)$ *existe um emparelhamento que cobre* A *se, e somente se,*

$$|N(S)| \geq |S| \text{ para todo } S \subseteq A. \tag{5.6}$$

Demonstração. Sejam $G = (A \cup B, E)$ um grafo bipartido, M um emparelhamento que cobre A e $S \subseteq A$. Para cada $x \in S$ denote por v_x o vértice de B tal que $\{x, v_x\} \in M$. Certamente, $v_x \in N(S)$. Ainda, se $x \in S$ e $y \in S$ com $x \neq y$ então $v_x \neq v_y$, logo $|N(S)| \geq |S|$.

Agora, suponha que $|N(S)| \geq |S|$ para todo $S \subseteq A$ e seja U uma cobertura mínima em G. Definimos $A' = A \cap U$, $B' = B \cap U$, $A'' = A \setminus A'$ e $B'' = B \setminus B'$.

Se não existe um emparelhamento que cobre A então do teorema 40 deduzimos $|U| < |A|$. Como $|U| = |A'| + |B'|$ e $|A| = |A'| + |A''|$, se $|U| < |A|$ então $|B'| < |A''|$.

Ainda, não há arestas de A'' para B'', pois elas não estariam cobertas por U, ou seja $N(A'') \subseteq B'$. Portanto

$$|N(A'')| \leq |B'| < |A''|, \tag{5.7}$$

contrariando a hipótese. □

Essa demonstração é bastante simples pois todo o trabalho já foi feito no Teorema de König. Vejamos uma demonstração que não depende de outros resultados.

Demonstração alternativa do teorema de Hall. Vamos provar por indução em $|A|$ que se $|N(S)| \geq |S|$ para todo $S \subseteq A$ então existe um emparelhamento que cobre A.

Se $|A| = 1$ então $|N(A)| \geq 1$, portanto, existe um emparelhamento que cobre A. Seja $G = (A \cup B, E)$ um grafo bipartido que satisfaz (5.6) e suponha que todo grafo bipartido $(A' \cup B', E)$ com $|A'| < |A|$ que satisfaz a condição de Hall (5.6) tem um emparelhamento que cobre A. A demonstração segue em dois casos.

Caso 1: $|N_G(S)| > |S|$ para todo $S \subset A$ não-vazio. Escolha uma aresta $\{a, b\} \in E$ e considere o grafo bipartido $G' = G - a - b$ sobre $A' = A \setminus \{a\}$ e $B' = B \setminus \{b\}$. Nesse caso, para cada $S \subseteq A' \subset A$ vale que $|N_{G'}(S)| \geq |S|$ e pela hipótese indutiva concluímos que existe M que cobre A'. Portanto $M \cup \big\{\{a, b\}\big\}$ cobre A.

Caso 2: $|N(S)| = |S|$ para algum $S \subset A$ não-vazio. O grafo bipartido induzido $H = G[S \cup N(S)]$ satisfaz a condição de Hall (os vizinhos de S em H são os mesmo vizinhos em G) e pela hipótese indutiva podemos concluir que existe um emparelhamento M em H que cobre S. Agora, considere o subgrafo bipartido induzido $J = G[\overline{S} \cup \overline{N_G(S)}]$ e suponha que exista $X \subseteq \overline{S}$ tal que no grafo J vale $|N_J(X)| < |X|$. Teremos no grafo G

$$|N_G(S \cup X)| = |N_H(S) \cup N_J(X)| = |N_H(S)| + |N_J(X)| < |S| + |X|$$

contrariando a hipótese de G satisfazer a condição de Hall. Assim $|N_J(X)| \geq |X|$ para todo $X \subseteq \overline{S}$ e, pela hipótese indutiva, existe um emparelhamento M' em J que cobre $\overline{S}$. Para concluir a demonstração é suficiente observar que $M \cup M'$ é um emparelhamento em G que cobre A. □

Corolário 42 (Forma defectiva do teorema de Hall)**.** *Em todo grafo bipartido* $G = (A \cup B, E)$ *existe um emparelhamento que cobre* A *a menos de* d *vértices se, e somente se,*

$$|N(S)| \geq |S| - d \text{ para todo } S \subseteq A. \tag{5.8}$$

Exercícios

Exercício 158. Prove que se G é bipartido então existe um emparelhamento que cobre todos os vértices de grau $\Delta(G)$.

Exercício 159. Prove que se G é bipartido e regular então G tem emparelhamento perfeito.

Exercício 160. Seja $G = (A \cup B, E)$ um grafo bipartido com $|A| = |B| = n$. Mostre que se não existe um emparelhamento perfeito em G então existe um subconjunto S com $|S| \leq \lceil n/2 \rceil$ tal que $|N(S)| = |S| - 1$ e ou $S \subset A$ ou $S \subset B$.(Dica: considere um conjunto T minimal violando a condição de Hall, dada pela equação (5.6).)

Exercício 161. Prove o corolário 42. (Dica: Adicione d vértices novos a B e faça-os adjacentes a todos os vértices de A.)

Exercício 162. Seja $G = (A \cup B, E)$ um grafo bipartido e $\{A_1, A_2\}$ uma partição de A e $\{B_1, B_2\}$ uma partição de B. Mostre que se $N(A_1) \subseteq B_1$ então $N(B_2) \subseteq A_2$ e $B_1 \cup A_2$ é uma cobertura.

Exercício 163. Seja $G = (A \cup B, E)$ um grafo bipartido e M um emparelhamento em G. Seja $U \subset V(G)$ o conjunto dos vértices cobertos por M e W o conjunto dos vértices de todos os caminhos M-alternantes que têm um dos extremos em $A \setminus U$. Prove que $(B \cap W) \cup (A \setminus W)$ é uma cobertura em G.

Exercício 164. Prove que todo emparelhamento máximo em $G = (A \cup B, E)$ tem cardinalidade

$$\min_{U \subseteq A} |A| - |U| + |N(U)|.$$

Exercício 165. O **permanente** de uma matriz quadrada $M = M(u, v)$, onde $(u, v) \in A \times B$, é o número

$$\operatorname{perm}(M) = \sum_{\pi} \prod_{u} M(u, \pi(u)),$$

onde a soma se estende a todas as bijeções $\pi\colon A \to B$.

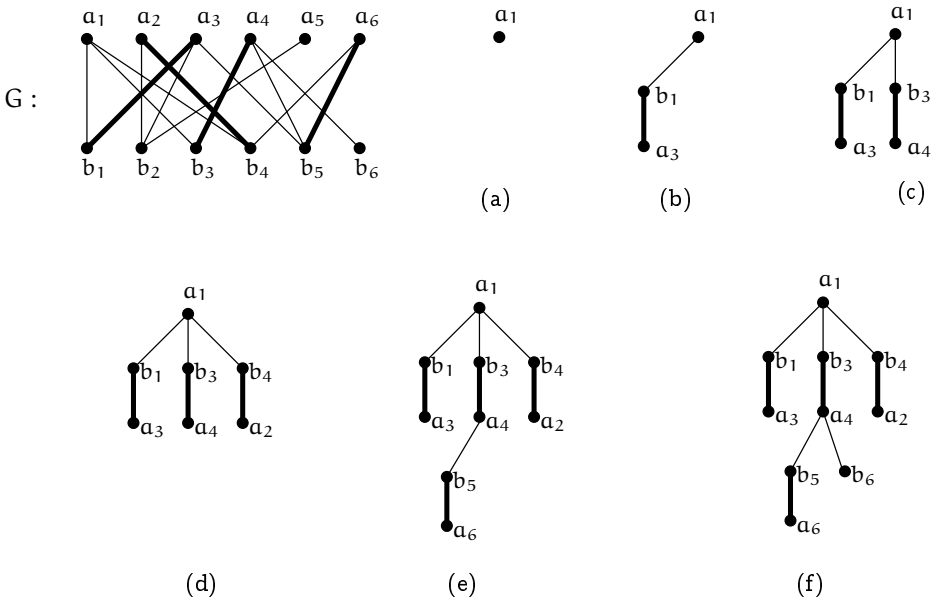
Seja $G = (A \cup B, E)$ um grafo bipartido tal que $|A| = |B|$. Seja M a matriz indexada por $A \times B$ e definida por $M(u, v) = 1$ se $\{u, v\}$ é uma aresta de G e $M(u, v) = 0$ caso contrário. Mostre que o permanente de M é igual ao número de emparelhamentos perfeitos em G.

Exercício 166. Na primeira prova que apresentamos do teorema de Hall deduzimos o resultado do teorema de König. Esses teoremas são, de fato, equivalentes. Demonstre o teorema de König ($\mu > \nu$) a partir do teorema de Hall.

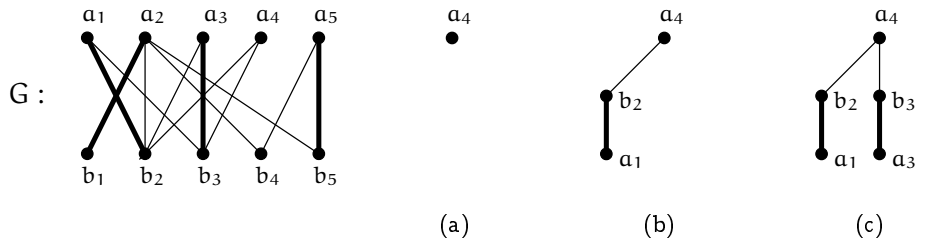
5.3 Algoritmo de Edmonds

O algoritmo abaixo recebe um grafo bipartido $G = (A \cup B, E)$ e devolve um emparelhamento que cobre A ou um subconjunto $S \subseteq A$ tal que $|N(S)| < |S|$ cuja existência é garantida pelo teorema de Hall.

A idéia do algoritmo é construir caminhos alternantes a partir de um vértice não-coberto em A. Por exemplo, no grafo da figura abaixo, o algoritmo começa pelo vértice a_1 não-coberto por M. O próximo passo é escolher um vizinho de a_1. Se existir algum vizinho não coberto, então achamos um caminho aumentante, caso contrário uma aresta de M tem extremo nesse vizinho e temos um caminho alternante, no exemplo a_1, b_1, a_3. O próximo passo é continuar a busca a partir de um vizinho dos vértices da forma a_i já escolhidos. Notemos, entretanto, que basta buscar tais vizinhos dentre os vértices ainda não escolhidos, no nosso exemplo, após o estágio representado pela figura (e) abaixo não há necessidade de considerar a aresta $\{a_3, b_5\}$ pois o novo caminho alternante definido por essa aresta seria redundante para nossos propósitos. No estágio dado pela figura (f) chegamos a um caminho aumentante.



Quando o algoritmo descobre um caminho aumentante, usa-o para obter um emparelhamento com mais arestas e recomeça o processo. Caso contrário, teremos construído caminhos alternantes que começam num vértice descoberto e todos terminam num vértice coberto. Os vértices desses caminhos definem um conjunto que viola a condição de Hall. Por exemplo, na figura abaixo o algoritmo começa pelo vértice não-coberto a_4. Esse vértice tem os vizinhos b_2 e b_3 que estão cobertos pelas arestas a_1b_2 e a_3b_3 respectivamente. O vértice a_1 já tem todos os seus vizinhos escolhidos, assim como a_3, e não se pode mais estender os caminhos. Nesse caso $N(\{a_1, a_3, a_4\}) = \{b_1, b_2\}$ e $\{a_1, a_3, a_4\}$ é um obstáculo para um emparelhamento cobrir A.



Essa idéia está formalizada na seguinte prova do teorema de Hall.

Terceira demonstração do Teorema de Hall. Sejam $G = (A \cup B, E)$ um grafo bipartido tal que $|N(S)| \geq |S|$ para todo $S \subseteq A$ e M um emparelhamento máximo em G. Suponha que M não cobre A e seja $u \in A$ não coberto por M.

Denote por C o subconjunto de $A \cup B$ formado por todos os vértices alcançáveis a partir de u por caminho M-alternante. Tome $S = C \cap A$ e $T = C \cap B$.

Certamente, $T \subset N(S)$. Agora, se existe $x \in S$ com um vizinho fora de T, digamos $b \in B \setminus T$, então u e b são extremos de um caminho M-aumentante, contrariando o fato de M ser máximo, portanto, $T = N(S)$. Como T e $S - \{u\}$ são cobertos por M (justifique), temos $|T| = |S| - 1$, logo $|N(S)| = |S| - 1 < |S|$ o que contraria a hipótese sobre G, logo M cobre A. □

Não é difícil extrair um algoritmo dessa demonstração, esse algoritmo pode ser escrito da seguinte maneira.

Algoritmo 21: Edmonds(A, B, E)

Dado : um grafo bipartido $G = (A \cup B, E)$.
Devolve: emparelhamento que cobre A ou um obstáculo de Hall $S \subseteq A$.

```
1  comece com M vazio;
2  se existe u ∈ A não coberto por M então
3      (S, T) ← ({u}, ∅);
4  senão devolva(M);
5  se N(S) = T então devolva(S);
6  senão escolha b ∈ N(S) \ T;
7  se existe a ∈ V \ S tal que {a, b} ∈ M então
8      insira a em S;
9      insira b em T;
10     volte para 5;
11 senão
12     seja P o caminho M-aumentante de u até b;
13     atribua a M o emparelhamento M ⊕ E(P);
14     volte para 2.
```

Análise e correção do algoritmo de Edmonds. A correção do algoritmo segue facilmente da terceira prova do teorema de Hall. O tempo para a busca de um caminho M-aumentante é $O(|E|)$ no pior caso. Fazendo uma busca pra cada vértice resulta em $O(|V||E|)$.

Exercícios

Exercício 167. Refaça os exercícios 2 e 21.

Exercício 168. Uma escola secundária abriu vagas para contratação de 6 docentes para as seguintes áreas: Matemática, Química, Física, Biologia, Psicologia e Ecologia. Para que um(a) candidato(a) se inscreva ele(a) deve informar a área em que se graduou e as áreas correlatas em que se sente à vontade para lecionar. A escola recebeu seis inscrições para estas posições, da seguinte maneira:

Candidato	**Áreas**					
	Matem.	Química	Física	Biol.	Psicol.	Ecol.
Antônio		×	×			
Bernardo			×	×	×	×
Cássia	×	×	×			
Débora		×		×	×	×
Evandro	×	×				
Fernanda	×		×			

Execute um algoritmo conhecido que determine o maior número de professores que a escola pode contratar.

Exercício 169. Faça uma análise detalhada da complexidade do algoritmo de Edmonds.

Exercício 170. Modifique o algoritmo Edmonds(A, B, E) para que ele devolva um emparelhamento máximo.

Exercício 171. Como foi dito na observação 2, página 14, determinar $\alpha(G)$ é um problema NP-difícil. Suponha que exista um algoritmo que computa $\nu(G)$ em tempo polinomial para qualquer grafo G. Mostre como computar $\alpha(G)$ em tempo polinomial.

Exercício 172. Escreva um algoritmo polinomial para computar $\alpha(G)$ quando G é bipartido.

Exercício 173. Escreva um algoritmo que, dado G, determine um emparelhamento *maximal* em G, isto é, emparelhamento M tal que não exista um emparelhamento M' tal que $M \not\subset M'$. Use esse algoritmo para determinar uma cobertura S tal que $|S| \leq 2\nu(G)$.

Exercício 174. Seja $G = (V, E, \rho)$ um grafo com pesos não negativos nas arestas. O peso de um emparelhamento M em G é

$$p(M) = \sum_{e \in M} \rho(e).$$

Defina

$$\nu_\rho(G) = \max\{p(M) \colon M \text{ é emparelhamento em } G\}$$

o peso de um emparelhamento de peso máximo em G. Usando a idéia do Kruskal, escrevemos o seguinte algoritmo

Algoritmo 22: Emparelhamento_quase_máximo(G)

Dado : um grafo G com pesos não-negativos nas arestas.
Devolve: um emparelhamento em G.

```
1 M ← ∅;
2 F ← fila das arestas em ordem não-crescente de peso;
3 para cada e ∈ F faça
4     se M ∪ {e} é um emparelhamento em G então insira e em M;
5 devolva M.
```

Prove que a resposta do algoritmo satisfaz $p(M) \geq \nu_\rho(G)/2$.

GRAFOS DIRIGIDOS

Neste capítulo trabalharemos com grafos dirigidos. Em alguns situações, como calcular distância, o caso dirigido é uma simples generalização do caso não-dirigido, em outras situações, como conexidade, o problema é outro. Lembramos que, formalmente, um **grafo dirigido** é um par (V, E) onde V é um conjunto finito (vértices) e $E \subset V \times V$ com a restrição de $(v, v) \notin E$ para todo $v \in V$.

Dizemos que a aresta $(v, u) \in E$ *sai de* v e *chega em* u. Também, definimos os seguintes conjuntos para cada $v \in V$

$$E^-(v) = \{(v, u) \in V \times V : (v, u) \in E\} \quad \text{e} \quad N^-(v) = \{u \in V : (v, u) \in E\}; \tag{6.1}$$
$$E^+(v) = \{(u, v) \in V \times V : (u, v) \in E\} \quad \text{e} \quad N^+(v) = \{u \in V : (u, v) \in E\}. \tag{6.2}$$

Definimos o **grau de saída** de $v \in V$ e o **grau de entrada** de $v \in V$, respectivamente, por

$$d^-(v) = |N^-(v)| \ \text{ e } \ d^+(v) = |N^+(v)|. \tag{6.3}$$

O grafo subjacente ao grafo dirigido G é o grafo

$$SG = \left(V(G), \bigcup \{\{u, v\} : (u, v) \in E\}\right)$$

Neste capítulo expomos alguns tópicos em grafos dirigidos, para mais sobre o assunto veja [1].

6.1 Representação computacional e Percurso

Como fizemos antes, no caso não dirigido (seção 1.5.1), assumiremos que nos problemas computacionais os vértices de G são $\{1, 2, \ldots, |V(G)|\}$ (caso contrário construímos um isomorfismo) e uma *lista de adjacências dirigida* é um vetor $N[\,]$ de listas ligadas, a lista $N[i]$ contém os vizinhos $N^-(i)$ do vértice i e cada nó dessa lista é composto por uma variável que armazena um vértice e um ponteiro que aponta para o próximo nó da lista.

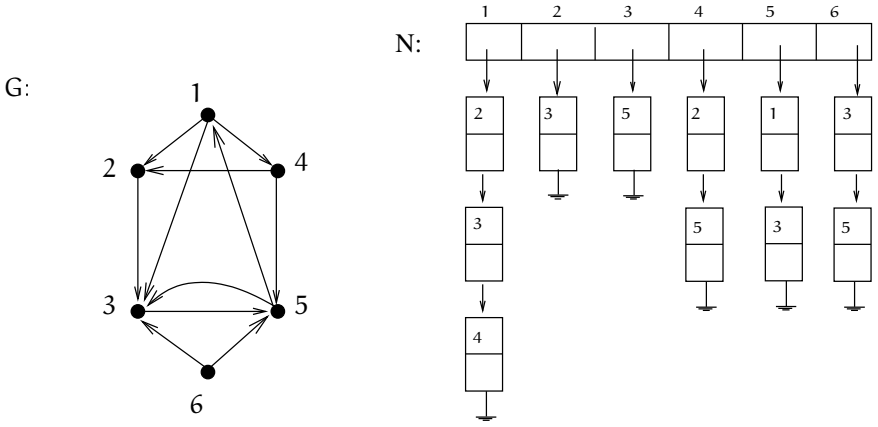


Figura 6.1: Diagrama de um grafo dirigido e uma representação por lista de adjacências.

O resultado de uma busca em profundidade rotulada, algoritmo 4 na página 28, num grafo dirigido é uma classificação das arestas em quatro tipos, ao invés de dois do caso não-dirigido. Usaremos a terminologia da seção 3.1.1, se existe algum inteiro $t \in \mathbb{N}$ tal que $pai^{(t)}[v] = u$ então dizemos que u é **ancestral** de v ou que v é **descendente** de u. No caso particular de $pai[u] = v$ dizemos que u é **filho** de v.

Após uma busca em profundidade rotulada as arestas são classificadas em

1. *aresta pai–filho* são as arestas da forma $(v, pai[v])$;
2. *aresta de retorno ascendente* são as arestas (u, v) tais que v é um ancestral de u (veja a seção 3.1.1), nesse caso $[chega[u], sai[u]] \subset [chega[v], sai[v]]$;
3. *aresta de retorno descendente* são as arestas (u, v) tais que v é um descendente de u com $pai[v] \neq u$, nesse caso $[chega[v], sai[v]] \subset [chega[u], sai[u]]$; e
4. *aresta transversal* são as outras arestas, da forma (u, v) tais que $chega[v] < sai[v] < chega[u] < sai[u]$.

Exemplo 32. A figura abaixo esquematiza uma busca em profundidade rotulada no grafo dirigido do exemplo 6.1,

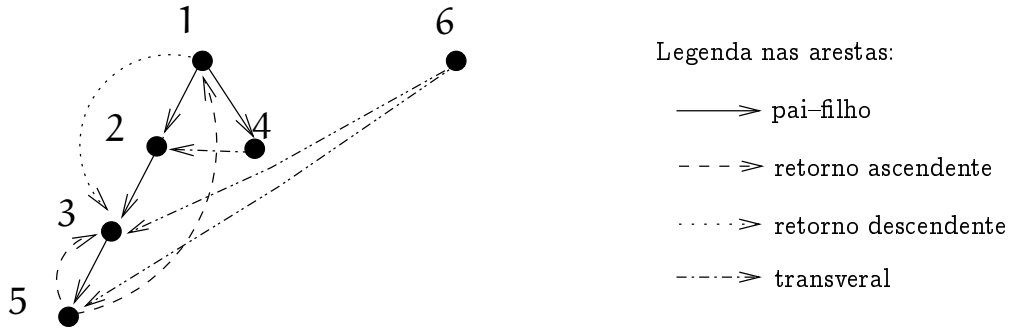


Figura 6.2: Busca em profundidade rotulada.

Um **caminho orientado** é uma seqüência de vértices distintos $P = x_1, x_2, \ldots, x_k$ tal que $(x_i, x_{i+1}) \in E(P)$ para todo $i \in \{1, 2, \ldots, k-1\}$. Nesse caso, dizemos que P é um caminho orientado *de* x_1 *para* x_k e usamos a notação $(x_1 \rightarrow x_k)$**-caminho**. Um **circuito orientado** é uma

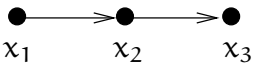


Figura 6.3: Representação de um caminho orientado com 3 vértices.

seqüência de vértices distintos, a menos do primeiro e do último, $C = x_1, x_2, \ldots, x_k, x_1$ tal que $(x_i, x_{i+1}) \in E(C)$ para todo $i \in \{1, 2, \ldots, k-1\}$, e $(x_k, x_1) \in E(C)$

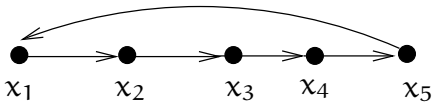


Figura 6.4: Diagrama de um circuito orientado com 5 vértices.

Exercícios

Exercício 175 (**Fecho transitivo**). Se $D = (V, E)$ é um grafo dirigido então o fecho transitivo de D é o grafo $D^* = (V, E^*)$ onde

$$E^* = \{(i, j) \in V \times V\text{: existe um } (i \rightarrow j)\text{-caminho em } D\}.$$

Escreva um algoritmo de tempo $O(|V|^3)$ para determinar o fecho transitivo de um grafo dirigido.

Exercício 176 (**Ordenação topológica**). Dado um grafo orientado G que não contém circuito orientado, determinar uma ordenação $v_1 \prec v_2 \prec \cdots \prec v_n$ de $V(G)$, onde $u \prec v$ se $(u, v) \in E(G)$. Essa ordenação é chamada de ordenação topológica dos vértices de G.

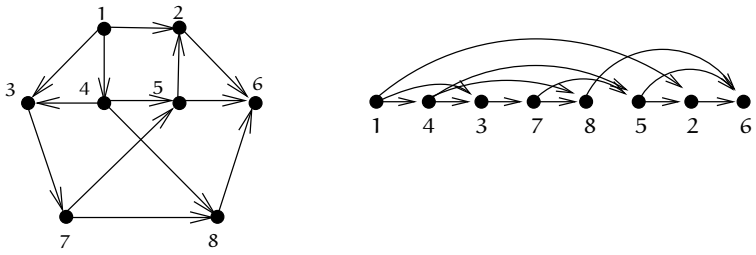


Figura 6.5: Um grafo orientado e uma ordenação topológica de seus vértices: $1 \prec 4 \prec 3 \prec 7 \prec 8 \prec 5 \prec 2 \prec 6$.

(a) Prove que G admite uma ordenação topológica se e somente se G não contém circuito orientado.
(b)Prove que G não tem circuito orientado se e somente se em qualquer busca em profundidade não ocorre aresta de retorno ascendente.
(c) Escreva um algoritmo de tempo $O(|V| + |E|)$ para determinar uma ordenação topológica de um grafo orientado sem circuito orientado.
(d) Prove que o algoritmo está correto.

6.2 Caminhos mínimos em grafos dirigidos com pesos nas arestas

Além da orientação os grafos desta seção têm peso nas arestas, ou seja, esses grafos são definidos por uma tripla (V, E, ρ) com $\rho\colon E \to \mathbb{R}$. O **comprimento** de um caminho orientado $P = x_1, x_2, \ldots, x_k$ é definido, naturalmente, como a soma dos pesos (comprimentos) das arestas nesse caminho

$$c(P) = \sum_{i=1}^{k-1} \rho(x_i, x_{i+1}),$$

e a **distância** entre dois vértices é o comprimento do menor caminho orientado que os liga,

$$\mathrm{dist}(u, v) = \min\{c(P)\colon P \text{ é um } (u \to v)\text{-caminho}\}$$

quando existe algum caminho, caso contrário convencionamos que $\mathrm{dist}(u, v) = \infty$. Um $(u \to v)$-caminho de comprimento $\mathrm{dist}(u, v)$ é chamado de **caminho mínimo**.

Exemplo 33. O seguinte diagrama representa um grafo dirigido com pesos nas arestas

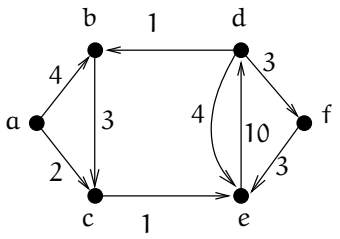


Figura 6.6: Diagrama de um grafo dirigido com peso nas arestas. Note que $\mathrm{dist}(c, e) \neq \mathrm{dist}(e, c)$, $\mathrm{dist}(a, f) = 10$.

Observação 14. Num grafo dirigido G podemos ter

$$\mathrm{dist}(u, v) \neq \mathrm{dist}(v, u)$$

e vale a seguinte forma da desigualdade triangular, para todo $(v,u) \in E(G)$

$$dist(s,u) \leq dist(s,v) + \rho(v,u) \tag{6.4}$$

onde, como anteriormente, $x \leq \infty$ e $x + \infty = \infty$ para qualquer $x \in \mathbb{R} \cup \{\infty\}$.

Não é difícil provar que o algoritmo de Dijkstra, página 37, computa o seguinte problema: dado G dirigido e com pesos positivos nas arestas representado por uma lista de adjacências como foi definida na seção 6.1 acima e dado $s \in V(G)$, determinar $dist(s,v)$ para todo $v \in V(G)$. No que segue veremos um algoritmo que funciona no caso em que podem haver arestas com peso negativo (veja exercício 89), desde que o grafo não contenha um circuito orientado onde a soma dos pesos nas arestas seja negativa.

6.2.1 Algoritmo de Bellman–Ford

Seja G um grafo dirigido com pesos nas arestas que podem ser negativos. Um circuito **circuito negativo** C em G é um circuito orientado onde a soma dos pesos das arestas é negativo.

Notemos que num circuito negativo podemos ficar dando voltas e a cada volta os comprimentos diminuem, assim uma tentativa ingênua de projetar um algoritmo para computar distâncias em G poderia facilmente entrar em *loop* para sempre.

O algoritmo de Bellman–Ford recebe um grafo dirigido com pesos $G = (V, E, \rho)$, onde ρ pode assumir valores negativos e um vértice s, devolve um valor booleano indicando a não-existência em G de um circuito negativo e um $(s \to v)$-caminho para algum $v \in V(C)$. No caso do valor devolvido ser *verdadeiro*, o algoritmo computou corretamente as distâncias $dist(s,v)$ para todo $v \in V$.

A idéia do algoritmo é a seguinte. Seja v um vértice de G e

$$P = s, v_1, \ldots, v_{k-1}, v$$

um $(s \to v)$-caminho mínimo com k arestas. Claramente, temos $k \leq |V| - 1$. A idéia principal é repetir $|V| - 1$ vezes a versão dirigida do algoritmo Relaxação(u,w), página 37, para cada aresta

para cada $(u,w) \in E(G)$ **faça** Relaxação(u,w).

onde

Algoritmo 23: Relaxação(u,w)
1 **se** $d[w] > d[u] + \rho(u,w)$ **então** $d[w] \leftarrow d[u] + \rho(u,w)$.

assim, na primeira rodada de relaxações em E(G) temos que $dist_G(s,v_1)$ está determinado; na segunda, $dist_G(s,v_2)$ está determinado (veja o lema 14, que também vale no caso dirigido). Na k-ésima rodada $dist_G(s,v)$ está determinado (veja exercício 179 a seguir). Como qualquer caminho tem no máximo $|V| - 1$ arestas, no final das repetições as distâncias estarão determinadas.

Para detectar um circuito negativo basta testar se há alguma aresta $(u,w) \in E(G)$ tal que $d[w] > d[u] + \rho(u,w)$ depois das $|V| - 1$ repetições do trecho de algoritmo dado acima.

Algoritmo 24: Bellman–Ford(G,s)

Dado : um grafo G com pesos ρ nas arestas e um vértice $s \in V(G)$.
Devolve: *verdadeiro* no caso em que $dist_G(s,v)$, para todo $v \in V(G)$, foi corretamente calculado, e *falso* caso G tenha um circuito negativo alcançável a partir de s.

```
1 para cada v ∈ V faça d[v] ← ∞;
2 d[s] ← 0;
3 cont ← 1;
4 enquanto cont ≤ |V| − 1 faça
5     para cada (u,w) ∈ E(G) faça Relaxação(u,w);
6     cont ← cont + 1;
7 para cada (u,w) ∈ E(G) faça
8     se d[w] > d[u] + ρ(u,w) então devolva(falso);
9 devolva(verdadeiro).
```

Análise e correção do algoritmo de Bellman–Ford. O resultado a seguir prova que o algoritmo devolve corretamente o valor booleano prometido: *verdadeiro* se não há circuito negativo alcançável por s e *falso* caso contrário, indicando que os valores $d[v]$ calculados não valem. A prova de que o algoritmo computa corretamente as distâncias é deixada para o leitor no exercício 179.

Lema 43. Bellman–Ford(G) *devolve* verdadeiro *se* G *não contém circuito negativo e devolve* falso *caso contrário.*

Demonstração. Seja $C = v_0, v_1, \ldots, v_k, v_0$ um circuito de peso negativo, isto é,

$$\rho(v_k, v_0) + \sum_{i=1}^{k} \rho(v_{i-1}, v_i) < 0,$$

e tal que existe $(s \to v_0)$-caminho dirigido em G. Suponha que o algoritmo devolve *verdadeiro*. Então $d[v_i] \leq d[v_{i-1}] + \rho(v_{i-1}, v_i)$ para todo $i \in \{1, 2, \ldots, k\}$ e somando para toda aresta de C

$$\begin{aligned}
&\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} \big(d[v_{i-1}] + \rho(v_{i-1}, v_i)\big) \Longrightarrow \\
&\sum_{i=1}^{k} d[v_i] - \sum_{i=1}^{k} d[v_{i-1}] \leq \sum_{i=1}^{k} \rho(v_{i-1}, v_i) \Longrightarrow \\
&\sum_{i=1}^{k} \rho(v_{i-1}, v_i) \geq d[v_k] - d[v_0] \geq 0
\end{aligned}$$

contradizendo o fato de C ter peso negativo.

Quando não há circuito negativo, após o término, para cada $(u, w) \in E$ temos

$$d[w] = \mathrm{dist}_G(s, w) \leq \mathrm{dist}_G(s, u) + \rho(u, w) = d[u] + \rho(u, w),$$

portanto o algoritmo devolve *verdadeiro*. □

Para a complexidade, notemos que o tempo dos laços aninhados nas linhas 4 e 5 predomina e resulta em $O(|V||E|)$.

Exercícios

Exercício 177. Suponha que $v_1, v_2, \ldots, v_k$ é um caminho orientado de comprimento mínimo de v_1 para v_k num grafo dirigido G com pesos nas arestas. Prove que $v_i, \ldots, v_j$ é um caminho mínimo de v_i para v_j em G para quaisquer i, j com $1 \leq i < j \leq k$.

Exercício 178. Prove a desigualdade (6.4).

Exercício 179 (Correção do algoritmo de Bellman–Ford). Seja G um grafo dirigido com pesos nas arestas. Suponha as inicializações das três primeiras linhas do algoritmo de Bellman–Ford. Prove que independente do número de vezes que a Relaxação foi executado, sempre valem:

(b) $d[v] \geq \mathrm{dist}_G(s, v)$ para todo $v \in V(G)$, e uma vez que vale a igualdade o valor de $d[v]$ não muda após qualquer execução de Relaxação;

(c) se $\mathrm{dist}_G(s, v) = \infty$ então $d[v] = \infty$;

(d) se $s, \ldots, u, w$ é um $(s \to w)$-caminho mínimo e $d[u] = \mathrm{dist}_G(s, u)$ então Relaxação(u, w) resulta em $d[w] = \mathrm{dist}_G(s, w)$;

(e) seja $s = v_0, v_1, v_2, \ldots, v_{k-1}, v_k = w$ um caminho mínimo. Se ocorrem as relaxações em (v_0, v_1), (v_1, v_2),..., (v_{k-1}, v_k), nessa ordem e com possíveis outras relaxações intermediárias, então teremos no final da seqüência $d[w] = \mathrm{dist}_G(s, w)$.

Exercício 180. Modifique o algoritmo de Bellman–Ford para que resulte $d[v] = -\infty$ para todo vértice v tal que o $(s \to v)$-caminho mínimo encontra um circuito negativo.

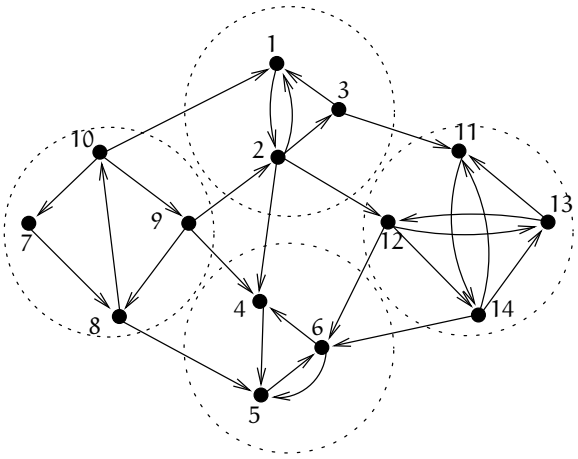
Exercício 181. Suponha que existe um algoritmo $\mathcal{A}(G, u, v)$ que determina em tempo polinomial um $(u \to v)$-caminho mínimo no grafo dirigido com pesos reais nas arestas G. Mostre como usar esse algoritmo para determinar se G tem um circuito (orientado) hamiltoniano em tempo polinomial. (*Dica:* atribua peso -1 a todas as arestas do grafo e compute $\mathcal{A}(G, u, v)$ para todo $(v, u) \in E(G)$.)

Observamos que o problema de determinar se um grafo é hamiltoniano é NP-completo, portanto a existência de uma algoritmo $\mathcal{A}$ como acima estabelece que P=NP.

6.3 Componentes fortemente conexos

Um *componente fortemente conexo* W num grafo dirigido G é um subconjunto de vértices maximal com respeito a seguinte propriedade: para quaisquer $u, v \in W$ existem um $(u \to v)$-caminho orientado e um $(v \to u)$-caminho orientado, ambos contidos em $G[W]$.

Exemplo 34. O diagrama abaixo mostra um grafo dirigido, cada um dos quatro círculos identifica um componente fortemente conexo do grafo.



O problema que estudamos agora é: dado um grafo dirigido G determinar os componentes fortemente conexos de G. Pra isso, definimos os seguintes dois grafos construídos a partir do grafo dirigido G

grafo transposto: denotado por G^T é o grafo dirigido obtido invertendo-se o sentido das arestas em $E(G)$: $G^T = \big(V(G), \{(u, v) \in V \times V\colon (v, u) \in E(G)\}\big)$.

grafo das componentes: denotado por G^* é o grafo orientado com um vértice para cada componente fortemente conexo de G e (c_1, c_2) é uma aresta em G^* se, e somente se, existe uma aresta que sai de um vértice de c_1 e chega num vértice de c_2 (veja figura 6.7(a) abaixo).

Observação 15. Os grafos G e G^T têm os mesmos componentes fortemente conexos. De fato, se u e v estão no mesmo componente W de G então o $(u \to v)$-caminho em G com todos os vértices em W é um $(v \to u)$-caminho em G^T com todos os vértices em W e, analogamente, o $(v \to u)$-caminho em G é um $(v \to u)$-caminho em G^T, portanto, u e v estão no componente W de G^T.

Observação 16. O grafo G^* dos componentes fortemente conexos de G não contém circuito orientado. De fato, dados $v, v' \in c_i$ e $u, u' \in c_j$, para $i \neq j$, se existe $(u \to v)$-caminho em G então não pode existir $(v' \to u')$-caminho em G, caso contrário esses vértices estariam no mesmo componente fortemente conexo. Com isso, temos que G^* admite ordenação topológica.

Exemplo 35. Na figura abaixo mostramos (a) o grafo dos componentes fortemente conexos do grafo dirigido mostrado no exemplo 34 e também (b) uma ordenação topológica dos vértices do grafo das componentes.

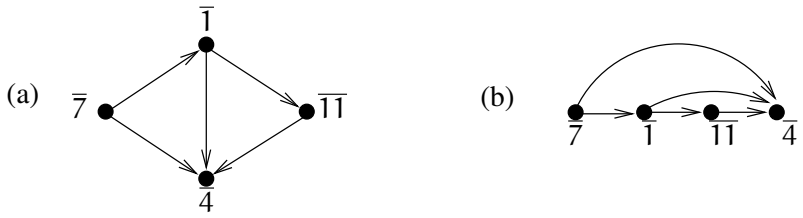


Figura 6.7: (a) Grafo dos componentes fortemente conexos do grafo do exemplo 34, $\overline{1} = \{1,2,3\}$, $\overline{4} = \{4,5,6\}$, $\overline{7} = \{7,8,9,10\}$ e $\overline{11} = \{11,12,13,14\}$. (b) Ordenação topológica do grafo das componentes.

Agora, vamos descrever um algoritmo que, dado um grafo dirigido G, determina os componentes fortemente conexos de G. Em linhas gerais, a estratégia para computar os componentes fortemente conexos de um grafo dirigido G é

Algoritmo 25: CFC(G)

1 BP(G): busca em profundidade rotulada em G para determinar $sai[v]$ para todo $v \in V(G)$;
2 Compute G^T;
3 BP(G^T): busca em profundidade em G^T com o laço principal modificado para escolher os vértices em ordem decrescente dos valores de $sai[\,]$ computado no passo 1;
4 Devolva cada árvore da segunda busca como um componente fortemente conexo de G.

No que segue sempre que nos referirmos aos valores de $chega[\,]$ e $sai[\,]$ trata-se do conteúdo desses vetores após a primeira busca em profundidade. Os seguintes resultados ajudam a entender o funcionamento do algoritmo. As demonstrações serão deixadas como exercício.

Lema 44. *Sejam W e W' dois componentes fortemente conexos de* G. *Suponha que existe uma aresta $(u,v) \in E(G)$ tal que $u \in W$ e $v \in W'$. Então*

$$\max_{u \in W} sai[u] > \max_{v \in W'} sai[v].$$

Idéia da demonstração. Se a busca em profundidade rotulada chega no componente W antes de chegar no componente W', então todos os vértices de W' são descendentes do primeiro vértice visitado em W, o valor de $sai[\,]$ desse vértice é maior que o valor de $sai[\,]$ de todo vértice de W'. Se a busca em profundidade chega primeiro num vértice de W', então todos os vértices de W' serão visitados antes de qualquer visita a um vértice de W, portanto os valores de $sai[\,]$ dos vértices em W são maiores que os valores de $sai[\,]$ dos vértices de W'. □

Corolário 45. *Sejam W e W' dois componentes fortemente conexos de* G. *Suponha que existe uma aresta $(u,v) \in E(G^T)$ tal que $u \in W$ e $v \in W'$. Então*

$$\max_{u \in W} sai[u] < \max_{v \in W'} sai[v].$$

Demonstração. Imediato da definição de G^T e do resultado acima pois G e G^T têm os mesmos componentes conexos. □

A busca em profundidade do passo 3 começa no componente fortemente conexo W que tem o vértice $k \in V(G)$ com $\max_v sai[v]$. Essa busca visita todos os vértices de W. Pelo corolário 45 somente esses vértices serão visitados, pois não há arestas de W para W' com $\max_{v \in W'} sai[v] < \max_{v \in W} sai[v]$, para qualquer outro componente fortemente conexo W' em G. Na segunda rodada do laço interno da busca em profundidade, o vértice u é tal que $sai[u] = \max\{sai[v]\colon v \in V(G) - W\}$ e, novamente pelo corolário 45, os vértices das componentes W'' tais que $\max_{v \in W''} sai[v] < sai[u]$ não são visitados; os únicos vértices fora do componente fortemente conexo de u que poderiam ser visitados são os da componente W, mas que já estão *visitados*; e assim por diante.

Análise e correção do algoritmo CFC. Para facilitar a prova de que o algoritmo funciona corretamente vamos, primeiro, dar uma versão mais detalhada do algoritmo.

O algoritmo CFC(G) descrito a seguir faz uma busca em profundidade em G e depois uma busca em profundidade em G^T onde os vértices são visitados respeitando a ordenação descrita acima.

Em seguida mostramos que o algoritmo CFC( ) funciona corretamente. Lembramos que $sai[\,]$ refere-se aos valores computados na primeira chamada da busca em profundidade.

O seguinte algoritmo usa o vetor $cfc[\,]$ para identificar, no final da execução, os componentes fortemente conexos de G.

A busca em profundidade utilizada fica da seguinte forma

Algoritmo 26: BP(G, v, cc)

Dado : um grafo dirigido G.
Devolve: busca em profundidade rotulada modificada.

```
1 cfc[v] ← cc;
2 chega[v] ← cont;
3 cont ← cont + 1;
4 para cada u ∈ N⁻(v) faça
5     se chega[u] = 0 então BP(G, u, cc);
6 sai[v] ← cont;
7 cont ← cont + 1.
```

onde o parâmetro cc rotula os vértices de acordo com o componente fortemente conexo ao qual ele pertence, e o algoritmo CFC fica

Algoritmo 27: CFC(G)

Dado : um grafo dirigido G.
Devolve: componentes fortemente conexos.

```
  /* Busca em profundidade rotulada em G para computar sai[ ] */
1 para cada k ∈ V(G) faça
2     sai[k] ← 0;
3     chega[k] ← 0;
4 para cada k ∈ V(G) faça
5     se chega[k] = 0 então BP(G, k, 0);
  /* Ordene os vértices por ordem decrescente de sai[ ] */
6 L ← lista ordenada de V(G) por ordem decrescente de sai[ ];
  /* Determine o grafo transposto */
7 compute G^T;
  /* Busca em profundidade em G^T para determinar os componentes */
  /* Os vértices não-visitados são escolhidos de acordo com a ordem em L */
8 cont ← 0;
9 c ← 0;
10 para cada k ∈ V(G) faça
11     sai[k] ← 0;
12     chega[k] ← 0;
13 para cada k ∈ L faça
14     se chega[k] = 0 então
15         c ← c + 1;
16         BP(G^T, k, c);
```

Agora, damos uma prova de que o algoritmo CFC está correto.

Teorema 46. *O algoritmo* CFC(G) *computa corretamente os componentes fortemente conexos do grafo dirigido* G.

Demonstração. Vamos provar que, para todo $n \in \mathbb{N}$, após a n-ésima rodada do laço na linha 13

que *Se* $C = m$ *então* $W_i = \{v\colon cfc[v] = i\}$ *para* $i \in \{1, 2, \ldots, m\}\}$ *são* m *componentes conexos de* G.

Para $n = 0$, antes da primeira rodada, a afirmação acima vale pois $cfc[v] = 0$ para todo $v \in V(G)$. Suponha que após a n-ésima rodada do laço na linha 13 temos $c = m$ e $W_i = \{v\colon cfc[v] = i\}$ para $i \in \{1, 2, \ldots, m\}\}$ são m componentes fortemente conexos de G. Consideremos a rodada $n+1$. Se o teste na linha 14 é falso então $c = m$ e após a rodada $n+1$ temos os mesmos m componentes fortemente conexos de G. Vamos supor que o teste foi positivo, dessa forma $c = m+1$ e temos que provar que $W_i = \{v\colon cfc[v] = i\}$, $1 \leq i \leq m+1$, são $m+1$ componentes fortemente conexos de G.

Após a atribuição $c = m + 1$ na linha 15, temos uma chamada da busca em profundidade $BP(G^T, k, m+1)$, que faz $cfc[k] = m+1$ e os vértices não-visitados, alcançáveis a partir de k, serão descendentes de k no final da busca e terão $cfc[\,] = m+1$ na medida em que forem visitados. Seja W_{m+1} o componente fortemente conexo que contém k; como $k \in L$ vale que

$$sai[k] = \max_{v \in W_{m+1}} sai[v] > \max_{u \in V \setminus \bigcup_{i=1}^{m+1} W_i} sai[u], \tag{6.5}$$

portanto, pelo corolário 45, não há aresta $(u, v) \in W_{m+1} \times \left(V \setminus \bigcup_{i=1}^{m+1} W_i\right)$ em G^T. Como os vértices de $\bigcup_{i=1}^{m} W_i$ já foram visitados (o teste da linha 14 falha) os valores de $cfc[\,]$ desses vértices não são alterados por $BP(G^T, k, m+1)$. Assim, nenhum vértice fora de W_{m+1} será descendente de k no final de $BP(G^T, k, m+1)$. □

Observemos que se G é dado pela sua matriz de adjacências A, então a matriz transposta A^T representa o grafo transposto G^T, dessa forma obtemos uma representação implícita de G^T em tempo constante no sentido de que $A^T(i, j) = A(j, i)$. Se G é dado por uma lista de adjacências então a lista de adjacências de G^T pode ser computada em tempo $O(|V| + |E|)$ (verifique).

A linha 2 tem custo $O(|V|)$, o laço na linha 4 tem custo total $O(|V| + |E|)$, a ordenação custa $O(|V| \log |V|)$, o grafo G^T pode ser computado em tempo $O(|V| + |E|)$, a linha 8 tem custo $O(1)$, a 10 $O(|V|)$ e o laço na linha 13 tem custo $O(|V| + |E|)$.

Teorema 47. $CFC(G)$ *tem complexidade* $O(|V| \log |V| + |E|)$.

Exercícios

Exercício 182. Demonstre o lema 44.

Exercício 183. Mostre que o algoritmo 25 pode ser implementado com complexidade $O(|V| + |E|)$.

Exercício 184. Se ao invés de usarmos uma busca em profundidade em G^T por ordem decrescente de $sai[\,]$ no algoritmo 27 usarmos uma busca em profundidade em G por ordem crescente de $chega[\,]$ o algoritmo determinaria os componentes fortemente conexos do grafo?

Exercício 185. Se ao invés de usarmos uma busca em profundidade em G^T por ordem decrescente de $sai[\,]$ no algoritmo 27 usarmos uma busca em profundidade em G por ordem crescente de $sai[\,]$ o algoritmo determinaria os componentes fortemente conexos do grafo?

Exercício 186. Seja G um grafo dirigido e u e w vértices de G.

(a) Escrevemos $u \rightsquigarrow w$ se existe $(u \to w)$-caminho em G. Prove que $\rightsquigarrow$ é uma relação simétrica e transitiva em $V(G)$.

(b) Escrevemos $u \leftrightsquigarrow w$ se existem $(u \to w)$- e $(w \to u)$-caminhos em G. Prove que $\leftrightsquigarrow$ é uma relação de equivalência em $V(G)$. Nesse caso, quem são as classes de equivalência?

Exercício 187. Escreva um algoritmo para determinar se um grafo dirigido G é *semiconexo*: para $u, v \in V(G)$ ou $u \rightsquigarrow v$ ou $v \rightsquigarrow u$. Determine a complexidade do algoritmo.

Exercício 188. Escreva um algoritmo de complexidade $O(|V| + |E|)$ para computar o grafo G^* dos componentes fortemente conexos de G.

Exercício 189. Escreva um algoritmo de complexidade $O(|V| + |E|)$ para determinar o grafo transposto de um grafo dado por uma lista de adjacências.

6.4 Fluxo em redes

Vamos chamar de **rede** uma quádrupla $N = (G, c, s, t)$ onde G é um grafo dirigido, c é uma função $c\colon E(G) \to \mathbb{R}^+$ que atribui uma *capacidade* $c(e)$, para cada aresta e s e t são vértice de G tais que $d^+(s) = d^-(t) = 0$. Um s-t **fluxo**, ou simplesmente fluxo, em N é uma função $f\colon E \to \mathbb{R}$ tal que

1. para cada $e \in E$ vale $0 \leq f(e) \leq c(e)$;
2. para cada $v \in V \setminus \{s, t\}$ temos $\sum_{e \in E^-(v)} f(e) = \sum_{e \in E^+(v)} f(e)$.

A primeira restrição diz que o fluxo por uma aresta não pode exceder a capacidade dessa aresta, a segunda diz que em qualquer vértice diferente de s e de t o fluxo que "sai" é igual a fluxo que "entra" no vértice. Com isso, é fácil intuir que o fluxo que "sai" de s é igual ao que "entra" em t.

Proposição 48. *Se N é uma rede e f um fluxo, então*

$$\sum_{e \in E^-(s)} f(e) = \sum_{e \in E^+(t)} f(e). \tag{6.6}$$

Demonstração. Notemos que

$$\sum_{e \in E(G)} f(e) = \sum_{e \in E^-(s)} f(e) + \sum_{v \neq s,t} \sum_{e \in E^-(v)} f(e)$$

e que

$$\sum_{e \in E(G)} f(e) = \sum_{e \in E^+(t)} f(e) + \sum_{v \neq s,t} \sum_{e \in E^+(v)} f(e),$$

portanto

$$\sum_{e \in E^-(s)} f(e) = \sum_{e \in E^+(t)} f(e).$$

□

A quantidade na equação (6.6) é o **valor do fluxo** e é denotada por $\mathbf{val}(f)$

$$\mathrm{val}(f) = \sum_{e \in E^-(s)} f(e). \tag{6.7}$$

O problema que estamos interessados aqui é formulado da seguinte maneira: Dado uma rede $N = (G, c, s, t)$, determinar um s-t fluxo f de valor máximo, isto é, tal que $\mathrm{val}(f) \geq \mathrm{val}(g)$ para todo fluxo g na rede N.

Fluxo e corte em redes são conceitos estreitamente relacionados, como seria de se esperar. Em um grafo dirigido G um **corte separa os vértices** s **e** t, ou s-t **corte**, é um conjunto de arestas

$$E(S, \bar{S}) = \left\{(u, v) \in E(G)\colon u \in S \text{ e } v \in \bar{S}\right\} \text{ tal que } s \in S \text{ e } t \in \bar{S}$$

e a **capacidade do corte** é

$$c(S) = \sum_{e \in E(S, \bar{S})} c(e); \tag{6.8}$$

e

$$f(S) = \sum_{e \in E(S, \bar{S})} f(e) - \sum_{e \in E(\bar{S}, S)} f(e) \tag{6.9}$$

é o fluxo através do s-t corte S. Claramente, $f(S) \leq c(S)$ para todo s-t corte S, mais que isso, $\mathrm{val}(f) \leq c(S)$. De fato,

$$\begin{aligned} \mathrm{val}(f) - f(S) &= \sum_{e \in E^-(S)} f(e) - \sum_{e \in E(S, \bar{S})} f(e) + \sum_{e \in E(\bar{S}, S)} f(e) = \sum_{v \in S} \sum_{e \in E^-(v)} f(e) - \sum_{v \in S} \sum_{e \in E^+(v)} f(e) \\ &= \sum_{v \in S} \left(\sum_{e \in E^-(v)} f(e) - \sum_{e \in E^+(v)} f(e) \right) = 0. \end{aligned}$$

Proposição 49. *Para todo* s-t *fluxo* f *e todo* s-t *corte* S *vale* $\mathrm{val}(f) \leq c(S)$*, em particular*

$$\max\{\mathrm{val}(f)\colon f \text{ é um s-t fluxo}\} \leq \min\{c(S)\colon S \text{ é um s-t corte}\}. \tag{6.10}$$

Demonstração. Dado um s-t corte S, se s-t corte S, então $\mathrm{val}(f) = f(S)$ para todo s-t corte S, como o fluxo em cada aresta é limitado pela capacidade da aresta, $\mathrm{val}(f) = f(S) \leq c(S)$ para todo s-t corte S. □

Um s-t **corte mínimo** é um s-t corte de capacidade mínima, ou seja, o corte $E(S, \overline{S})$ é mínimo se $c(S) \leq c(S')$ para todo s-t corte S'.

Corolário 50. *Sejam* f *é um* s-t *fluxo e* S *é um* s-t *corte. Se* $\mathrm{val}(f) = c(S)$ *então* f *é um fluxo máximo e* S *um corte mínimo.*

Demonstração. Sejam f^* um s-t fluxo máximo e S^* um s-t corte mínimo. Então

$$\mathrm{val}(f) \leq \mathrm{val}(f^*) \leq c(S^*) \leq c(S).$$

De $\mathrm{val}(f) = c(S)$ vale as igualdades na equação acima. □

O principal resultado desta seção é que a capacidade de um corte mínimo é um limitante inferior para o fluxo máximo.

Seja $P = s, x_1, \ldots, x_{m-1}, t$ um caminho com extremos s e t no *grafo subjacente* ao grafo dirigido G e defina $x_0 = s$ e $x_m = t$. Seja f um s-t fluxo na rede $N = (G, c, s, t)$. Dizemos que P é f-**aumentante** se para cada $\{x_i, x_{i+1}\} \in E(P)$ temos

1. se $e = (x_i, x_{i+1}) \in E(G)$ então $f(e) < c(e)$; dizemos que e é uma aresta *direta* de P,
2. se $e = (x_{i+1}, x_i) \in E(G)$ então $f(e) > 0$, nesse caso dizemos que e é uma aresta *reversa* de P.

Se P é f-aumentante então, tome

$$\epsilon_1 = \min\{c(e) - f(e)\colon\ e \text{ é uma aresta direta de P}\} \tag{6.11}$$

$$\epsilon_2 = \min\{f(e)\colon\ e \text{ é uma aresta reversa de P}\} \tag{6.12}$$

$$\delta = \min\{\epsilon_1, \epsilon_2\}. \tag{6.13}$$

e defina g por

$$g(e) = \begin{cases} f(e) + \delta & \text{se } e \text{ é direta} \\ f(e) - \delta & \text{se } e \text{ é reversa} \\ f(e) & \text{caso contrário.} \end{cases} \tag{6.14}$$

Então, g é um fluxo e $\mathrm{val}(g) = \mathrm{val}(f) + \delta$, portanto, f não é máximo. A recíproca desse resultado, ou seja, se não existe caminho f-aumentante então f é máximo, também vale. Antes de demonstrar esse resultado, vejamos um exemplo.

Exemplo 36. Para exemplificar o papel das arestas reversas considere a rede representada abaixo

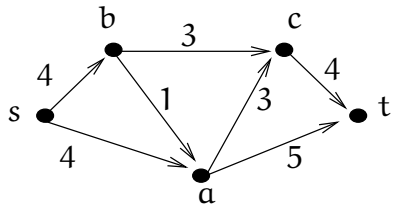


Figura 6.8: Uma rede. Os números representam as capacidades das arestas.

Agora, considere o seguinte fluxo na rede acima

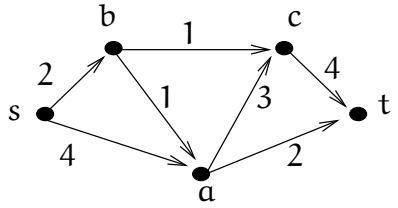


Figura 6.9: Um fluxo na rede da figura anterior. Os números representam o fluxo na aresta.

O único caminho aumentante nessa rede é o caminho s, b, c, a, t e (c, a) é uma aresta reversa. O fluxo obtido através desse caminho é representado por

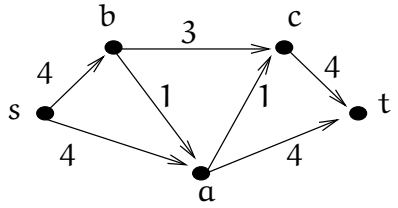


Figura 6.10: Uma rede. Os números representam as capacidades das arestas.

Teorema 51 (Ford e Fulkerson, 1956)**.** *Um* s-t *fluxo* f *numa rede* $N = (G, c, s, t)$ *é máximo se e somente se não há caminho* f*-aumentante em* N.

Demonstração. Provamos que se existe caminho f-aumentante então f não é máximo. Agora, suponha que não existe caminho f-aumentante. Defina S como o conjunto dos vértices v (incluindo s) para os quais há um caminho de f-aumentante com extremos s e v e considere o s-t corte $E(S, \bar{S})$.

Cada aresta e desse corte deve estar saturada, isto é, $f(e) = c(e)$ e para cada aresta $d \in E(\bar{S}, S)$ vale $f(d) = 0$, logo $f(S) = c(S)$ e como $val(f) = f(S)$ (veja demonstração da proposição 49) pelo corolário 50 f deve ser máximo. □

Corolário 52. *Um* s-t *fluxo* f *em* $N = (G, c, s, t)$ *é máximo se e somente se o conjunto* S *de todos os vértices alcançáveis a partir de* s *por um caminho* f*-aumentante é um subconjunto próprio de* $V(G)$. *Nesse caso,* $val(f) = c(S)$. □

Teorema 53 (Ford e Fulkerson, 1956)**.** *Se* $N = (G, c, s, t)$ *é uma rede onde* $c(e) \in \mathbb{Z}$ *para todo* $e \in E(G)$*, então existe um fluxo máximo* f *tal que* $f(e) \in \mathbb{Z}$ *para todo* $e \in E(G)$.

Demonstração. Seja N uma rede e tome $f_0(e) = 0$ para toda aresta e. Se f_0 não é máximo então existe um caminho f_0-aumentante tal que δ em (6.13) é um inteiro maior que 0. Tome $f_1 = g$, par g definida em (6.14) com $f = f_0$. Claramente, f_1 assume valores inteiros. Continuando dessa maneira teremos uma seqüência de fluxos inteiros $f_0, f_1, f_2, \ldots$ com valores estritamente crescente. Como o valor é limitado pela capacidade de qualquer corte, essa seqüência é finita e o último valor é máximo. □

Teorema 54 (Ford e Fulkerson, 1956)**.** *O valor máximo de um fluxo numa rede* N *é igual a capacidade mínima de um corte em* N*.*

Demonstração. O caso onde as capacidades são inteiras segue do teorema anterior. Se as capacidade são racionais então podemos recair no caso anterior multiplicando as capacidades pelo seu denominador comum. No caso real, precisamos de ferramentas que estão fora do nosso escopo, mas também vale o enunciado. □

Esses resultados dão origem ao seguinte algoritmo.

Algoritmo 28: Ford–Fulkerson(G, c, s, t)

Dado : uma rede (G, c, s, t).
Devolve: Um fluxo de valor máximo.

```
1 para cada (u,v) ∈ E(G) faça f(u,v) ← 0;
2 enquanto existe caminho f-aumentante faça
3     escolha um caminho f-aumentante;
4     determine δ como em (6.13);
5     defina g como em (6.14);
6     f ← g;
7 devolva(f).
```

Se as capacidades das arestas são inteiras então esse algoritmo rodo em tempo $O(\text{val}(f^*)|E|)$, onde f^* denota o fluxo máximo e seu valor é um limitante para o número de iterações do laço. Se as capacidades não são racionais o algoritmo pode não terminar (veja página 21 de [6]). Note que o algoritmo não é polinomial no tamanho da entrada (veja a observação 7). Ainda, no caso em que permitimos capacidades irracionais nas arestas o algoritmo pode não parar.

Exemplo 37. Esse é, talvez, o exemplo mais simples da situação em que o algoritmo de Ford–Fulkerson não pára, foi descoberto em 1993 por Uri Zwick [11]. Considere a rede representada na figura abaixo. O fluxo máximo nessa rede é $2x + 1$, onde x é um inteiro suficientemente grande. Se o algoritmo escolhe o caminho aumentante s, c, b, t, as capacidades residuais das arestas

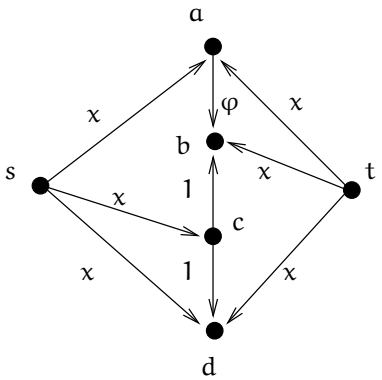


Figura 6.11: Exemplo de uma rede onde o algoritmo não converge para o fluxo máximo, onde $\varphi = \frac{\sqrt{5}-1}{2}$.

$(a, b), (c, b), (cd)$são, respectivamente, $\varphi, 0, 1$. Após $4n + 1$ aumentos as capacidades residuais são,

respectivamente, $\varphi^{2n-1}, 0, \varphi^{2n-2}$ e quando o número de aumentos cresce o valor do fluxo converge para $1 + 2\sum_{i\geq 1} \varphi^i = 4 + \sqrt{5}$.

Em 1972, Edmonds e Karp analisaram a seguinte versão para o algoritmo de Ford e Fulkerson. Seja f um fluxo em $N = (G, c, s, t)$ e defina o **grafo residual** como o grafo dirigido

$$R_f = \big(V(G), \{e \in E(G) \colon c_f(e) > 0\}\big),$$

onde $c_f(e) = c(e) - f(e)$. Considere o algoritmo de Ford–Fulkerson com a linha 3 implementada de modo que o caminho escolhido é o $s \to t$ caminho em G_f com o menor número de arestas, ou seja com $\mathrm{dist}_{SR_f}(s,t)$ arestas; isso pode ser feito por uma busca em largura.

Agora, vamos determinar uma cota superior para o número de aumentos feitos com essa estratégia.

Seja $(f_i)_{i\geq 0}$ ma seqüência de fluxos em $N = (G, c, s, t)$ com f_{i+1} definido a partir de f_i como em (6.14), através de um caminho f_i-aumentante de acordo com (6.14), para todo $i \geq 0$.

Proposição 55. *Para todo vértice $v \neq s, t$,*

$$\mathrm{dist}_{SG_{f_{i+1}}}(s, v) \geq \mathrm{dist}_{SG_{f_i}}(s, v)$$

Demonstração. Suponha que a afirmação é falsa e seja k o menor natural para o qual existe um vértice v tal que $\mathrm{dist}_{SG_{f_{k+1}}}(s, v) < \mathrm{dist}_{SG_{f_k}}(s, v)$. Seja $P = s, u_1, \ldots, u_\ell, v$ o caminho mínimo em $SG_{f_{k+1}}$, de modo que

$$\mathrm{dist}_{SG_{f_{k+1}}}(s, v) = \mathrm{dist}_{SG_{f_{k+1}}}(s, u_\ell) + 1. \tag{6.15}$$

Pela escolha de v

$$\mathrm{dist}_{SG_{f_{k+1}}}(s, u_\ell) \geq \mathrm{dist}_{SG_{f_k}}(s, u_\ell), \tag{6.16}$$

portanto, $\mathrm{dist}_{SG_{f_{k+1}}}(s, v) \geq \mathrm{dist}_{SG_{f_k}}(s, u_\ell) + 1$.

A demonstração agora segue em dois casos, dependendo se a aresta $\{u_\ell, v\} \in E(P)$ é uma aresta direta ou uma aresta reversa. Vamos supor que é uma aresta direta, o outro caso segue de dedução análoga.

Primeiro, notemos que se $(u_\ell, v) \in E(G_{f_k})$ então $\mathrm{dist}_{SG_{f_k}}(s, v) \leq \mathrm{dist}_{SG_{f_k}}(s, u_\ell) + 1$ pela desigualdade triangular; por (6.16) e (6.15) deduzimos que $\mathrm{dist}_{SG_{f_k}}(s, v) \leq \mathrm{dist}_{SG_{f_{k+1}}}(s, v)$, contrariando a hipótese assumida sobre v.

Agora, se $(u_\ell, v) \in E(G_{f_{k+1}})$ e $(u_\ell, v) \notin E(G_{f_k})$ então a aresta $\{u_\ell, v\} \in E(P)$ é uma aresta reversa no caminho aumentante de G_{f_k}; como esse caminho é mínimo, $\mathrm{dist}_{SG_{f_k}}(s, u_\ell) = \mathrm{dist}_{SG_{f_k}}(s, v) + 1$ e como $\mathrm{dist}_{SG_{f_{k+1}}}(s, v) \geq \mathrm{dist}_{SG_{f_k}}(s, u_\ell) + 1$, segue que $\mathrm{dist}_{SG_{f_{k+1}}}(s, v) \geq \mathrm{dist}_{SG_{f_k}}(s, u_\ell) + 2$, uma contradição. □

Teorema 56 (Edmonds e Karp, 1972)**.** *Com a estratégia acima,* Ford–Fulkerson *realiza* $O(|V||E|)$ *aumentos.*

Demonstração. Cada vez que um aumento é feito, pelo menos uma aresta no caminho aumentante utilizado é *crítica*, no sentido de que essa aresta limita o aumento no fluxo, ou seja, é determinante de δ em (6.13). Seja (i, j) uma aresta crítica no caminho f_k-aumentante. Suponha que a próxima vez que (i, j) apareça com uma aresta crítica seja no caminho f_ℓ-aumentante, para $\ell > k$. Nessa segunda ocorrência será com o sentido oposto ao da primeira ocorrência.

Suponha que (i, j) é uma aresta direta no caminho f_k-aumentante e uma aresta reversa no caminho f_ℓ-aumentante. Assim, $\mathrm{dist}_{SG_{f_k}}(s, j) = \mathrm{dist}_{SG_{f_k}}(s, i) + 1$ e $\mathrm{dist}_{SG_{f_\ell}}(s, i) = \mathrm{dist}_{SG_{f_\ell}}(s, j) + 1$.

Como $\mathrm{dist}_{SG_{f_k}}(s, j) \leq \mathrm{dist}_{SG_{f_\ell}}(s, j)$ temos que $\mathrm{dist}_{SG_{f_\ell}}(s, i) = \mathrm{dist}_{SG_{f_\ell}}(s, j) + 1 \geq \mathrm{dist}_{SG_{f_k}}(s, j) + 1 = \mathrm{dist}_{SG_{f_k}}(s, i) + 2$, ou seja, na segunda ocorrência da aresta como uma aresta crítica o caminho mínimo é duas arestas mais longo, pelo menos, que na primeira ocorrência.

Nenhum caminho de aumento tem mais que $|V| - 1$ arestas, logo nenhuma aresta pode ser crítica mais que $|V|/2$ vezes; como são no máximo $|E|$ arestas nos grafos residuais, temos no máximo $O(|V||E|)$ aumentos. □

Corolário 57. *O algoritmo* Ford–Fulkerson *com a estratégia de escolher o caminho aumentante com o menor número de arestas em cada rodada, determina a fluxo máximo numa rede em tempo* $O(|V||E|^2)$*, onde* $|V|$ *é o número de vértices e* $|E|$ *o número de arestas na rede.*

Demonstração. O passo 3 feito por uma busca em largura tem complexidade $O(|E|)$, com os $O(|V||E|)$ aumentos do teorema acima, resulta $O(|V||E|^2)$. □

Exercícios

Exercício 190. Prove que para quaisquer $S, T \subset V(G)$ vale que $f(S) = f(T)$. Derive desse fato que se $\mathrm{val}(f) = c(S)$ então f é máximo e S é mínimo.

Exercício 191. Seja f um fluxo que não é máximo numa rede. Seja g a função dada em (6.14). prove que g é um fluxo na mesma rede.

Exercício 192. labelexerc:menger2 Use o teorema do fluxo máximo–corte mínimo para derivar o teorema de Menger: Seja $N = (G, c, s, t)$ com c constante igual a 1. Então

- o valor de um fluxo máximo em N é igual ao número máximo de $(s \to t)$-caminhos arestas disjuntos em G.
- a capacidade de um corte mínimo é igual ao número mínimo de arestas cujas remoção destrói todos os $(s \to t)$-caminhos de G.

ALGORITMOS E ESTRUTURAS DE DADOS

A.1 Estruturas de dados para representar conjuntos disjuntos

Nessa seção veremos estruturas para implementar as operações $\mathtt{faz}$, $\mathtt{busca}$ e $\mathtt{união}$ utilizadas no algoritmo de Kruskal. Para efeito de comparação entre diferentes estruturas de dados para representar conjuntos disjuntos, as análises de desempenho dessas operações são realizadas em termos do número de operações $\mathtt{faz}$, $\mathtt{busca}$ e $\mathtt{união}$; doravante denotamos por m a quantidade de tais operações, e com a hipótese de que *as* n *primeiras operações são* $\mathtt{faz}$. Isso porque as vezes o tempo de pior caso de uma operação é uma super-estimativa para o custo da maioria das operações.

A.1.1 Estruturas de dados para representação de conjuntos disjuntos

O problema é representar dinamicamente uma família $\mathcal{S}$ de conjuntos disjuntos $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ sobre um universo $\mathcal{U}$ de modo que cada conjunto é identificado por um *representante* que é um elemento do próprio conjunto e que não tem nenhuma outra propriedade especial, o importante é que, dado que o conjunto não mudou, toda vez que se pergunta pelo representante a resposta deve ser a mesma. Para as nossas aplicações podemos assumir que $\mathcal{U} = \{1, 2, \ldots, n\}$.

Essas estruturas devem comportar as operações $\mathtt{faz}$, $\mathtt{busca}$ e $\mathtt{união}$ sobre $\mathcal{C}$:

$\mathtt{faz}(x)$ cria o conjunto unitário $C_{k+1} = \{x\}$ em $\mathcal{C}$

$\mathtt{busca}(x)$ devolve o representante do conjunto ao qual x pertence;

$\mathtt{união}(x, y)$ substitui em $\mathcal{C}$ os conjuntos que contém x e y pela união desses dois conjuntos; se $x \in C_x$ e $y \in C_y$ então $\mathtt{união}(x, y)$ remove os conjuntos C_x e C_y de $\mathcal{C}$ e acrescenta $C_x \cup C_y$ a $\mathcal{C}$, além disso escolhe-se um representante para $C_x \cup C_y$.

Representação usando vetor Usamos um vetor C indexado por $\mathcal{U}$ e a idéia é que dois elementos do conjunto universo x e y estão no mesmo conjunto se e somente se as posições x e y do vetor têm o mesmo conteúdo, assim $C_k = \{x \in \mathcal{U} : C[x] = k\}$. Escolhemos como representante do conjunto o menor elemento. Dessa forma, os algoritmos para as operações $\mathtt{faz}$, $\mathtt{busca}$ e $\mathtt{união}$ ficam da seguinte forma

Algoritmo 29: $\mathtt{faz}(x)$
1 $C[x] \leftarrow x$.

Algoritmo 30: $\mathtt{busca}(x)$)
1 **devolva**$(C[x])$.

Teorema 58. *O tempo gasto com* m *operações num universo com* n *elementos é* $O(mn)$. □

Algoritmo 31: união(x, y)

```
1 k ← min{busca(x), busca(y)};
2 para cada i de 1 até n faça
3     se C[i] = C[x]  ou  C[i] = C[y] então
4         C[i] ← k;
```

Observamos que o tempo de pior caso não está superestimado como mostra a seguinte seqüência de operações: faz(1), faz(2)...faz(n),

$$\begin{aligned} \text{união}(2,1) &\quad 1 \text{ atualização no vetor } C \\ \text{união}(3,2) &\quad 2 \text{ atualizações no vetor } C \\ \text{união}(4,3) &\quad 3 \text{ atualizações no vetor } C \\ \text{união}(5,4) &\quad 4 \text{ atualizações no vetor } C \\ \vdots &\quad \vdots \\ \text{união}(n,n-1) &\quad n-1 \text{ atualizações no vetor } C \end{aligned}$$

cujo custo é $\Theta(nm)$.

Corolário 59. *A da complexidade do algoritmo* Kruskal(G) *quando usamos vetor para representar e manipular conjuntos disjuntos é* $O(|V(G)||E(G)|)$ *para qualquer grafo conexo* G *com pesos nas arestas.*

Demonstração. Vimos que o tempo do Kruskal é $O(|E|\log|E|)$ mais o tempo gasto com as $O(|E|)$ operações, pelo teorema acima tempo $O(|V||E|)$. □

Representação usando listas ligadas Cada conjunto é dado por uma lista ligada. A cada elemento do universo está associado um nó com os atributos prox que aponta para o próximo elemento da lista e rep que aponta para o representante do conjunto.

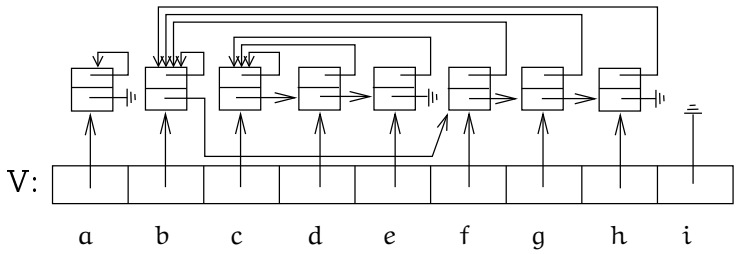


Figura A.1: Representação com listas ligadas dos subconjuntos $\{c, d, e\}$, $\{a\}$ e $\{b, f, g, h\}$ do universo $\{a, b, c, d, e, f, g, h, i\}$.

As funções faz e busca operam ligeiramente diferente da que vimos:

faz(x) cria um novo nó apontado por x;

busca(x) devolve um ponteiro para o representante do único conjunto ao qual x pertence;

união(x, y) une os conjuntos que contém x e y concatenando-se as listas e atualizando-se os ponteiros rep de todos elementos de uma das listas.

Uma busca (busca(x)) é feita em tempo constante e o custo de uma união (união(x, y)) é basicamente o custo das atualizações. Dessa forma, a complexidade de tempo das operações nessa representação é da mesma ordem de grandeza da representação por vetor. Adotando a seguinte heurística, a complexidade melhora substancialmente:

(‡) *Atualizar sempre a menor lista: pressupondo um atributo* $\ell[C]$ *que armazena o número de elementos de* C*, numa união de duas listas adicionamos a menor lista no final da maior lista.*

Para um elemento x do universo, a primeira vez que rep[x] é atualizado resulta numa lista de tamanho 2. A segunda vez que rep[x] é atualizado resulta numa lista de tamanho 4 e assim por diante. Logo, o número de atualizações que rep[ ] de um elemento qualquer é atualizado é $\lg n$.

Proposição 60. *Supondo a heurística* (‡) *se o representante de* x *muda* k *vezes, então o tamanho da lista que contém* x *é pelo menos* 2^k.

Demonstração. A prova é por indução em $k \in \mathbb{N}$ que vale: se o representante de x muda k vezes, então o tamanho da lista que contém x é pelo menos 2^k, para qualquer elemento x do universo.

Para $k = 0$, x é o único elemento na lista dele e portanto temos a base da indução. Para $k \geq 1$ assumimos que se $k-1$ atualizações foram feitas então a lista de x tem tamanho pelo menos 2^{k-1}.

Na próxima atualização do representante de x ocorre numa união com uma lista que tem pelo menos o mesmo número de elementos da lista de x, resultando numa lista com pelo menos $2 \cdot 2^{k-1} = 2^k$ elementos. O resultado segue do Princípio da Indução Finita. □

Corolário 61. *O representante de um elemento qualquer do universo muda no máximo* $\lfloor \lg n \rfloor$ *vezes.* □

Teorema 62. *O tempo para* m *operações num universo com* n *elementos é* $O(m + n \log n)$.

Demonstração. As operações faz e busca custam $O(m)$ e cada um dos n elementos tem rep[ ] atualizado $O(\log n)$ vezes. □

Corolário 63. *A complexidade do algoritmo de* Kruskal *sobre um grafo conexo* G *quando usamos listas ligadas para representar e manipular conjuntos disjuntos é* $O(|E(G)| \log |V(G)|)$.

Demonstração. São $O(|E|)$ operações que custam $O(|E| + |V| \log |V|)$ instruções mais o tempo de ordenação $O(|E| \lg |E|)$, que resulta na complexidade $O((|V| + |E|) \log |V|) = O(|E| \log |V|)$, pois G é conexo. □

Representação por floresta A idéia é que cada subconjunto seja representado por uma árvore de modo que cada elemento do subconjunto seja representado por um nó da árvore. Nessa árvore cada nó não-raiz aponta para um pai e a raiz é o representante (figura A.2); o pai da raiz é ela mesma. Dessa forma, faz cria em tempo constante uma nova árvore que contém somente a raiz, união é feita em tempo constante mudando-se o pai de uma das raízes (figura A.3) e busca, que tem tempo dependente da profundidade do elemento na árvore, retorna um ponteiro para uma raiz.

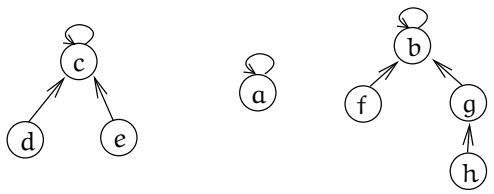


Figura A.2: Uma representação com árvores dos subconjuntos $\{c, d, e\}$, $\{a\}$ e $\{b, f, g, h\}$.

Claramente, essa estrutura pode ser muito ruim se a árvore formada for uma lista linear entretanto, com algumas heurísticas bastante simples o resultado final melhora substancialmente. Começamos com a seguinte heurística para união de dois subconjuntos, onde tamanho de uma árvore é o número de nós

União por tamanho: *árvore de menor tamanho é colocada como sub-árvore da árvore de maior tamanho.*

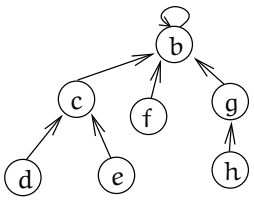


Figura A.3: Resultado de $\mathtt{união}(e, f)$.

Definimos a **altura de uma árvore** como a maior distância de um nó até a raiz e temos a seguinte relação entre tamanho e altura resultantes de uma seqüência de uniões com a heurística de união por tamanho.

Proposição 64. *Usando união por tamanho, o número de nós de cada árvore é pelo menos* 2^h*, onde* h *é a altura da árvore.*

Demonstração. Vamos denotar por $t(x)$ o número de nós na árvore que contém x e por $h(x)$ a altura da árvore que contém x. Vamos provar a proposição por indução no número de uniões: para todo $k \in \mathbb{N}$, após k uniões $t(x) \geq 2^{h(x)}$ para todo x.

Com 0 uniões estamos na configuração inicial, ou seja, $t(x) = 1$ e $h(x) = 0$ para todo x, ou seja, a base da indução é verdadeira. Para $k \geq 1$, vamos assumir que após $k-1$ uniões temos $t(x) \geq 2^{h(x)}$, para todo x. Se a k-ésima união é $\mathtt{união}(x, y)$ então $t(z) \geq 2^{h(z)}$ para todo z com $\mathtt{busca}(z) \notin \{\mathtt{busca}(x), \mathtt{busca}(y)\}$, ou seja, a relação entre tamanho e altura não muda para as árvores que não estão envolvidas na união. Vamos denotar por t' e h' o tamanho e a altura da árvore resultante da união e $t(x)$, $t(y)$, $h(x)$ e $h(y)$ os tamanhos e as alturas antes da k-ésima união. Podemos assumir, sem perda de generalidade, que $h(x) \geq h(y)$ e dessa forma na união teremos que o pai de y é x.

Com essa notação $h' = \max\{h(x), h(y)+1\}$. A demonstração segue em dois casos: (i) se $h' = h(x)$ então $t' = t(x) + t(y) \geq t(x)$ e pela hipótese de indução $t(x) \geq 2^{h(x)}$, logo $t' \geq 2^{h'}$; (ii) se $h' = h(y) + 1$ então $t' = t(x) + t(y) \geq 2t(y)$ e pela hipótese de indução $t(y) \geq 2^{h(y)}$, logo $t' \geq 2 \cdot 2^{h(y)} = 2^{h'}$. Em ambos os casos temos que após k uniões $t(x) \geq 2^{h(x)}$ para todo x. A proposição segue pelo Princípio da Indução Finita. □

Corolário 65. *O tempo de* m *operações num universo de tamanho* n *é* $O(m \log n + n)$.

Demonstração. Cada $\mathtt{faz}$ é feito em tempo $O(1)$ e cada $\mathtt{união}$ em tempo $O(1)$, resultando em $O(n)$. Cada $\mathtt{busca}$ é feito em tempo $O(\log n)$ resultando em $O(m \log n)$. □

Assumimos que cada nó x tem um atributo $t[x]$ que armazena o número de elementos na árvore enraizada em x os algoritmos ficam da seguinte forma

Algoritmo 32: $\mathtt{faz}(x)$

```
1 pai[x] ← x;
2 t[x] ← 1.
```

Algoritmo 33: $\mathtt{busca}(x)$

```
1 enquanto x ≠ pai[x] faça  x ← pai[x];
2 devolva(pai[x]).
```

Para descrever a união, usaremos o procedimento $\mathtt{link}$ a seguir.

Como o tempo das operações $\mathtt{busca}$ são proporcionais a altura parece ser mais natural uma heurística que leva em conta a altura e não o tamanho da árvore.

Algoritmo 34: $\mathrm{link}(x, y)$

```
1 t[x] ← t[x] + t[y];
2 pai[y] ← x.
```

Algoritmo 35: $\mathrm{união}(x, y)$

```
1 se t[busca(x)] ≥ t[busca(y)] então link(busca(x), busca(y));
2 senão link(busca(y), busca(x)).
```

União por altura: *usamos a altura, ao invés do tamanho pendurando a árvore mais baixa na raiz da árvore mais alta.*

Com essa heurística a proposição 64 continua valendo, e conseqüentemente o corolário. Os algoritmos para faz, busca e união são os algoritmos 32, 33 e 35 com t trocado por h, que é iniciado com 0. Em link só precisamos atualizar h se as árvore envolvidas na união têm a mesma altura,

Algoritmo 36: $\mathrm{link}(x, y)$

```
1 se h[x] = h[y] então h[x] ← h[x] + 1;
2 pai[y] ← x.
```

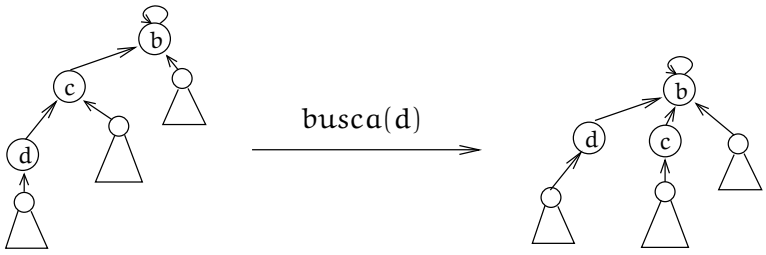
Comparando os tempos obtidos para lista ligada e árvores

$$O(m + n \log n) \times O(m \log n + n)$$

usando em ambas representações união por tamanho, concluímos que há situações onde o uso de árvores é pior. Esse fato remete-nos a uma próxima heurística.

Busca com compressão de caminhos e união por rank: *quando fazemos um* busca(x) *aproveitamos o percurso de* x *até a raiz da árvore e desviamos os ponteiros* pai[y]*, de todo* y *nesse caminho, para a raiz; na união penduramos a árvore de menor* rank *na raiz da de maior* rank.

Exemplo 38. Esquema de uma busca com compressão de caminhos.



Notemos que se estivermos usando união por altura, a compressão de caminhos modifica a altura da árvore e a atualização custa muito caro, portanto não a atualizamos e usamos h como uma estimativa superior para a altura, que chamaremos de **rank**.

O algoritmo para busca com compressão de caminhos fica

Algoritmo 37: $\mathrm{busca}(x)$

```
1 se x ≠ pai[x] então pai[x] ← busca(pai[x]);
2 devolva(pai[x]).
```

Essa duas heurísticas juntas, a união por *rank* e a compressão de caminhos é bastante eficiente. O próximo resultado, sobre a complexidade de m operações num universo de tamanho n, envolve

uma função $\alpha(m,n)$ que cresce muito vagarosamente; na prática podemos assumir que $\alpha = 4$. Não demonstraremos esse teorema (veja [4]), o resultado que mostraremos a seguir é um pouco menos preciso.

Teorema 66 (Tarjan, 1975)**.** *O tempo gasto com* m *operações num universo de tamanho* n *é* $O(m\alpha(m,n))$. □

Observação 17. A função $\alpha(m,n)$ é uma função que cresce muito vagarosamente. A função de Ackermann, dada por

$$A(m,k) = \begin{cases} 2^k & \text{se } m = 1, \\ A(m-1,2) & \text{se } m > 1 \text{ e } k = 1, \\ A(m-1,A(m,k-1)) & \text{caso contrário,} \end{cases}$$

é conhecida por crescer muito rapidamente, por exemplo $\text{UAU} = A(4,1) \sim 10^{80}$ (maior que o número estimado de átomos no universo observável), e

$$\alpha(m,n) = \min\{i : A(i, \lfloor m/n \rfloor) > \lg(n)\}.$$

Para efeitos práticos podemos assumir $\alpha < 5$.

Por exemplo

n	$\alpha(n,n)$
$0-2$	0
3	1
$4-7$	2
$8-2047$	3
$2048-$ UAU	4

Suponha uma seqüência qualquer de t operações das quais as n primeiras são faz. Particionamos os elementos de $\mathcal{U}$ de acordo com o *rank* final do nó associado: o Grupo 0 tem os elementos de *rank* 0 e 1; o Grupo 1 tem os elementos de *rank* 2; de um modo geral para $k > 2$ o Grupo k tem os elementos de *rank* em $(k, 2^k]$.

O *rank* dos elementos satisfazem as seguintes propriedades:

(i) quando um nó deixa de ser raiz o seu *rank* não muda nas próximas operações;

(ii) $\text{rank}[x] < \text{rank}[p[x]]$ para todo $x \in \mathcal{U}$ que não é raiz;

(iii) na sub-árvore enraizada em x se a altura é h então o número de elementos é pelo menos 2^h;

(iv) o número de nós de *rank* no intervalo $(k, 2^k]$ é no máximo $n/2^k$;

(v) o número de Grupos é $\lg^*(n)$, onde $\lg^*(n)$ é o número de vezes que temos que iterar lg até que o valor obtido seja menor ou igual a 1, por exemplo, $\lg^* 2^{16} = 4$.

O custo total das t operações é limitado superiormente pelo custo de $m \leq 2t$ operações busca. Vamos rastrear uma dessas operações. Num *busca(x)* os apontadores $p[y]$ no caminho de x até a raiz serão redirecionados para a raiz. Classificamos esses apontadores em 2 tipos:

1. Tipo 1 são os apontadores nos nós y tais que $p[y]$ está num grupo diferente de y, ou y é raiz, e

2. Tipo 2 são os apontadores dos nós que estão no mesmo grupo do pai.

As m operações redirecionam no máximo $\lg^*(n)$ apontadores do Tipo 1. Um apontador do Tipo 2 de um nó em $(k, 2^k]$ é redirecionado no máximo 2^k pois por (ii) a cada redirecionamento de $p[y]$ o apontador apontará para um nó de *rank* maior (para y que não é raiz ou filho de raiz) que o *rank*

do pai prévio, a partir daí por (i) o apontador será sempre do Tipo 1; como são no máximo $n/2^k$ nós o custo é no máximo

$$\sum_{i=0}^{\lg^*(n)} \frac{n}{2^k} 2 = O(n \lg^*(n)).$$

Com isso provamos

Teorema 67. *O tempo gasto com* m *operações num universo com* n *elementos é* $O((m+n)\lg^*(n))$.

Exercícios

Exercício 193. Demonstre as três propriedades de *rank* enunciadas acima.

Referências Bibliográficas

[1] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs*. Springer Monographs in Mathematics. Springer-Verlag London Ltd., London, 2001. Theory, algorithms and applications.

[2] Béla Bollobás. *Extremal graph theory*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1978.

[3] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990.

[4] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. The MIT Electrical Engineering and Computer Science Series. MIT Press, Cambridge, MA, 1990.

[5] Reinhard Diestel. *Graph theory*. Springer-Verlag, New York, second edition, 2000.

[6] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in networks*. Princeton University Press, Princeton, N.J., 1962.

[7] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1):23–38, 1986.

[8] Michael R. Garey and David S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[9] The Clay Mathematics Institute. Millennium Problems. `http://www.claymath.org/millennium`, May 2000. Acesso em 03/2008.

[10] L. Lovász and M. D. Plummer. *Matching theory*, volume 121 of *North-Holland Mathematics Studies*. North-Holland Publishing Co., Amsterdam, 1986. Annals of Discrete Mathematics, 29.

[11] Uri Zwick. The smallest networks on which the Ford-Fulkerson maximum flow procedure may fail to terminate. *Theoret. Comput. Sci.*, 148(1):165–170, 1995.

Índice Remissivo

Índice de Símbolos