Notas de aula em

TEORIA DOS GRAFOS

uma breve introdução com algoritmos

JAIR DONADELLI
Centro de Matremática, Computação e Cognição
Universidade Federal do ABC

28 de abril de 2016

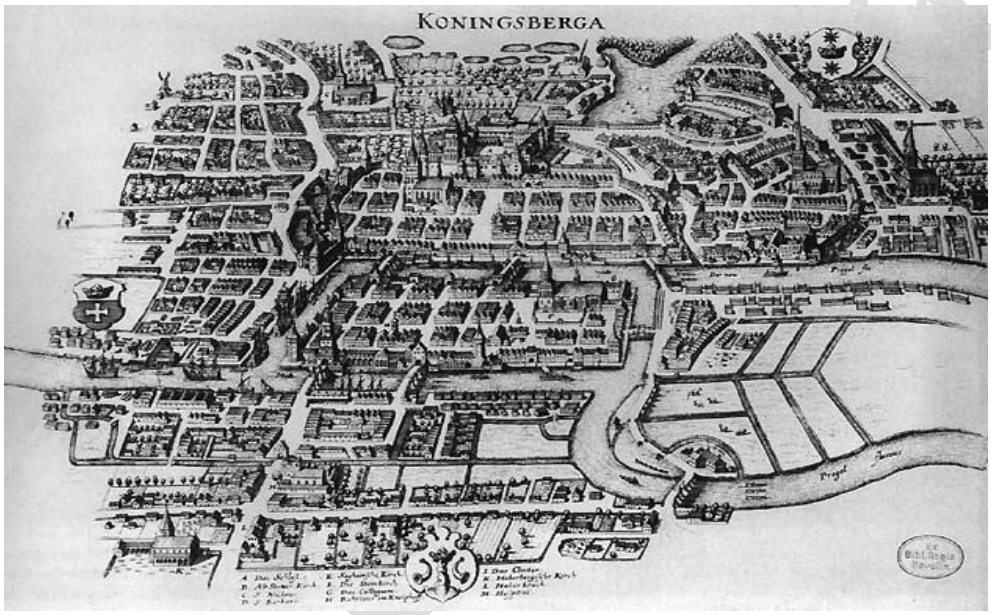
A versão eletrônica desse texto contém hyperlinks que ilustram a discussão de alguns tópicos da disciplina. Eu tomei o cuidado de fazer ligações com páginas WEB que tinham, no momento que fiz o acesso, informações corretas, entretanto essas páginas estão fora do meu controle e podem sofrer alterações,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

Sumário

Apresentação

"graphs are among the most ubiquitous models of both natural and human-made structures."[16]

O primeiro registro bibliográfico da Teoria dos Grafos tem como autor o ilustre matemático Leonhard Euler que em 1735 encontrou uma solução para os problema da pontes de Königsberg publicada em *Solutio problematis ad geometrian situs pertinentis*, Commentarii academiae scientiarum Petropolitanae 8, 1741, pp. 128-140. A cidade de Königsberg, capital da Prússia até 1945 e após a Segunda Guerra anexada pela Rússia passou a ser denominada Kalingrado, é dividida pelo Rio Pregel e possui duas grandes ilhas que estavam ligadas umas às outras e ao continente por sete pontes. O problema resolvido por Euler é decidir se é possível seguir um caminho que atravessa cada ponte exatamente uma vez e retorna ao ponto de partida.



Mapa de Königsberg em 1652. (domínio público)

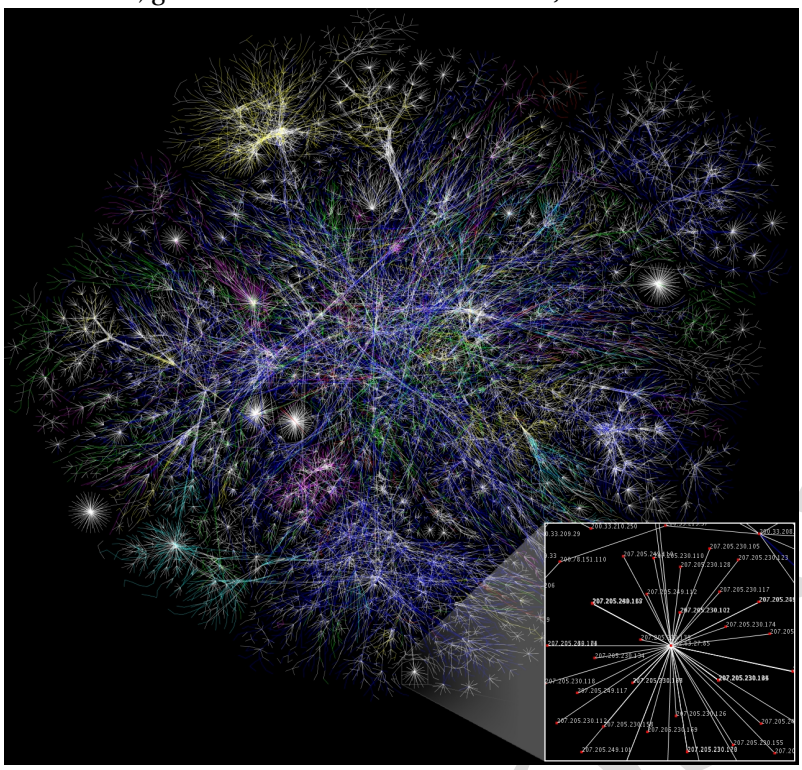
O termo *grafo* foi cunhado pelo matemático inglês James Joseph Sylvester em 1878 numa publicação na revista *Nature*

> "[...] *Every invariant and co-variant thus becomes expressible by a* graph *precisely identical with a Kekuléan diagram or chemicograph.* [...] *I give a rule for the geometrical multiplication of graphs, i.e. for constructing a* graph *to the product of in- or co-variants whose separate graphs are given.* [...]"

e o primeiro livro sobre o assunto, *Theorie der endlichen und unendlichen Graphen* (Teoria do Grafos finitos e infinitos), foi publicado em 1936 escrito pelo matemático húngaro Dénes Kőnig. Mais sobre os primórdios dessa história pode ser encontrado em [3].

Passados três séculos do trabalho seminal de Euler a Teoria dos Grafos tem tido cada vez mais destaque na Matemática com uso de técnicas sofisticadas e com conexões profundas com outros ramos da Matemática como Teoria da Medida [18], Teoria dos Grupos [21], Teoria de Códigos [22], Criptografia [14] e Complexidade Computacional. Além disso, uma grande variedade de aplicações em problemas modernos como os algoritmos de busca do *Google* e roteamento na *internet*, aplicações em diversos problemas computacionais de interesse prático como

alocação de registradores para otimização de código gerado por compiladores [5, 15], alocação de processos para processadores [8], e noutros problemas como alocação de frequência para rádios móveis [12] e em tráfego aéreo [2] fazem da Teoria dos Grafos uma ferramenta importante na solução de problemas de interesse prático. Cabe ainda citar as aplicações numa variedade de outras áreas como Física [9], Química [23] e a teoria das Redes Complexas (grosso modo, grafos com estruturas não triviais).



Grafo da Internet em 2005.[1]

"*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"

Neste texto apresentamos uma breve introdução à Teoria dos Grafos com apenas alguns dos temas mais importantes da disciplina. É um texto voltado para um primeiro contato com essa disciplina. Também são apresentados os algoritmos mais elementares. A enfase dada é para a solução algorítmica de problemas sem entrar em detalhes de implementação e, principalmente, sem aprofundamento em Estruturas de Dados. Por outro lado, a correção dos algoritmos é discutida.

Apresentamos no decorrer do texto vários exercícios com diferentes níveis de dificuldade, alguns poucos desses exercícios exigem conhecimento de tópicos de outras disciplinas.

As seções que começam com (∗) podem são opcionais.

[1] Essa imagem do *Wikimedia Commons* é devida ao usuário Matt Britt e está disponível sob a licença *Creative Commons Attribution 2.5*

Convenções notacionais

- $\mathbb{N}$ denota o conjunto dos números naturais;
- $\mathbb{Z}$ denota o conjunto dos números inteiros;
- $\mathbb{R}$ denota o conjunto dos números reais;
- $\mathbb{R}^+$ denota o conjunto dos números reais positivos;
- $|X|$ denota a cardinalidade do conjunto X;
- o conjunto $V \setminus U$ é o conjunto dos elementos de V que não estão em U;
- para $X \subseteq V$ denotamos por $\overline{X}$ o complemento de X em V, isto é, o conjunto $V \setminus X$;
- se U e V são conjuntos então $U \times V$ denota o produto cartesiano deles;
- $\binom{V}{k}$ denota o conjunto dos subconjuntos de V de cardinalidade k

$$\binom{V}{k} = \{X \subseteq V : |X| = k\};$$

  e de especial interesse é o caso $k = 2$;
- $\lfloor x \rfloor = \max\{n \in \mathbb{Z} : n \leq x\}$ para todo x real;
- $\lg(x)$ é a notação para logaritmo na base 2;
- para qualquer família de conjuntos não vazios $A_1, A_2, \ldots, A_m$

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

- denotamos por $A \triangle B$ a diferença simétrica dos conjuntos A e B, isto é,

$$A \triangle B = (A \cup B) \setminus (A \cap B) = (A \setminus B) \cup (B \setminus A);$$

- a família de conjuntos $A_1, A_2, \ldots, A_m$ é uma partição do conjunto V se

$$\bigcup_{i=1}^{m} A_i = V$$

  e A_i é disjunto de A_j sempre que $i \neq j$;
- P denota a classe dos problemas computacionais que podem ser decididos por algoritmos de complexidade polinomial no tamanho da entrada; NP denota a classe dos problemas computacionais cujas soluções podem ser verificados por algoritmos de complexidade polinomial no tamanho da entrada. Um problema NP-completo tem com característica o fato de que a descoberta de um algoritmo com complexidade de tempo polinomial no tamanho da entrada que resolve tal problema acarreta na existência de algoritmos de complexidade polinomial no tamanho da entrada para todo problema NP. Um dos problemas não resolvidos mais importantes da atualidade é o de resolver se vale a conjectura $\mathrm{P} \neq \mathrm{NP}$. Trata-se de um dos sete problemas do milênio [17], 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*;

- *Notação assintótica*: Sejam f_n e g_n sequencias de números reais, onde $f_n > 0$ para todo n suficientemente grande. Usaremos as seguintes notações para o comportamento assintótico (quando $n \to \infty$) dessas sequencias:
  - $g_n = O(f_n)$ se existem constantes positivas $c \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $|g_n| \leq c f_n$, para todo $n \geq n_0$;
  - $g_n = \Omega(f_n)$ se existem constantes positivas $C \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $g_n \geq C f_n$, para todo $n \geq n_0$;
  - $g_n = \Theta(f_n)$ se existem constantes positivas c, $C \in \mathbb{R}$ e $n_0 \in \mathbb{N}$ tais que $C f_n \leq g_n \leq c f_n$, para todo $n \geq n_0$.

1 | Conceitos básicos

Neste capítulo apresentamos os conceitos mais básicos da Teoria dos Grafos, também aqui convencionamos as notações que serão usadas por todo o texto. Nas primeiras seções, apresentamos a definição de grafo e alguns parâmetros e aspectos estruturais de grafos, na última seção aparecerão conceitos algorítmicos básicos, assim como representações de grafos apropriadas para tratar problemas computacionais.

1.1 Definições iniciais

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

Exemplo 1. Os conjuntos

$$V = \{1,2,3,4,5,6,7,8\} \text{ e } E = \big\{\{1,2\},\{1,5\},\{2,5\},\{3,4\},\{6,7\}\big\}$$

definem um grafo (V,E).

Se G denota o grafo que é definido pelo par (V,E) então escrevemos $G = (V,E)$. Quando nos referimos, genericamente, a um grafo G sem especificarmos o conjunto dos vértices e o conjunto das arestas que definem G esses passam a ser referidos como $V(G)$ e $E(G)$, 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 explicita a ordem de G, assim quando queremos ressaltar que G é um grafo de ordem n, para algum $n \in \mathbb{N}$, escrevemos G^n. Chamamos $G^0 = (\varnothing,\varnothing)$ de grafo vazio e *todo* grafo de ordem 1, genericamente denotado por G^1, de grafo trivial.

Todo grafo pode ser representado geometricamente por um diagrama da seguinte maneira. No plano, desenhamos um ponto para cada vértice e um segmento de curva ligando cada par de vértices que determina uma aresta. Claramente, a representação geométrica de um grafo não é única, mas para cada representação existe um único grafo. Um diagrama do grafo do exemplo 1 é apresentado na figura ao lado.

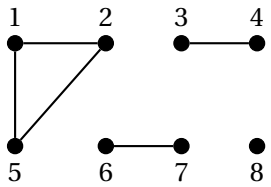


Figura 1.1: diagrama do grafo do Exemplo 1.

Denotemos por u e v dois vértices distintos do 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 da aresta; dizemos que duas arestas são arestas adjacentes se elas têm um vértice em comum; uma aresta é incidente nos seus extremos. Para um vértice v qualquer de G definimos o subconjunto das arestas do grafo que incidem em v

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

e o subconjunto dos vértices do grafo que são adjacentes a v

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

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

O número de vizinhos do vértice v é chamado de grau de v no grafo G. Os graus dos vértices de um grafo formam um dos seus parâmetros importantes e por isso recebem uma notação especial. Para um grafo $G = (V, E)$ não vazio nós definimos os seguintes parâmetros

$$\text{grau de } u \text{ em } G \quad - \quad d_G(u) = |N_G(u)| \tag{1.3}$$

$$\text{grau mínimo em } G \quad - \quad \delta(G) = \min\{d_G(u) \colon u \in V\} \tag{1.4}$$

$$\text{grau máximo em } G \quad - \quad \Delta(G) = \max\{d_G(u) \colon u \in V\} \tag{1.5}$$

$$\text{grau médio em } G \quad - \quad d(G) = \frac{1}{|V|} \sum_{u \in V} d_G(u) \tag{1.6}$$

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. Fazemos isso sempre que não há ambiguidade, ou perigo de confusão.

No exemplo 1 encontramos $d(8) = 0$, $d(7) = d(6) = d(4) = d(3) = 1$ e $d(5) = d(2) = d(1) = 2$. 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 $G = (V, E)$, definamos o conjunto $X = \{(u, e) \in V \times E \colon u \in e\}$ 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

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

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

Exercícios

Exercício 1. Um químico deseja embarcar os produtos P, Q, R, S, T, O, X usando o menor número de caixas. Alguns produtos não podem ser colocados numa mesma caixa porque reagem. Os produtos P, Q, R, X reagem entre si; P reage com O e com S e vice-versa; T também reage com O e com S 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 possível, determine o número de soluções.

Exercício 3. Defina todos os grafos sobre o conjunto de vértices $\{1,2,3,4\}$ e tamanho 6.

Exercício 4. Para todo $n \in \mathbb{N}$, qual é o número máximo de arestas que pode ter um grafo com n vértices?

Exercício 5. 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 6. Definimos 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 $G \cup H$ está bem definido?

Exercício 7. Definimos 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 $G \cap H$ está bem definida?

Exercício 8. 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 K^n. Prove que para qualquer grafo G vale que $G \cup \overline{G} = K^n$.

Exercício 9. 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 e o químico conhece uma listagem com m pares de produtos que reagem. Seja G o grafo que modela esse problema, onde vértices são produtos e arestas os pares que reagem. Denotemos por $\chi(G)$ o número de mínimo de caixas de modo que seja possível encaixotar os produtos com segurança. Prove que

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

(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 m de arestas para o grafo.)

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. 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 14. Prove que existem pelo menos dois vértices com o mesmo grau em todo grafo de ordem maior ou igual a dois.

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

Exercício 16. Prove que num grafo G com $\delta(G) > 0$ e $|E(G)| < |V(G)|$ existem pelo menos dois vértices de grau 1.

Exercício 17. Determine $\chi(G)$ e $\chi(LG)$ para os grafos definidos pelos diagramas da figura 1.2, em que χ é o parâmetro definido no exercício 9.

Exercício 18. 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.

1.2 Subgrafo

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 2. Considerando o grafo G do exemplo 1 temos que

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

são subgrafos de G, enquanto que

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

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 $U \subseteq V(G)$, o subgrafo induzido por U 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 é

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

Exemplo 3. 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.

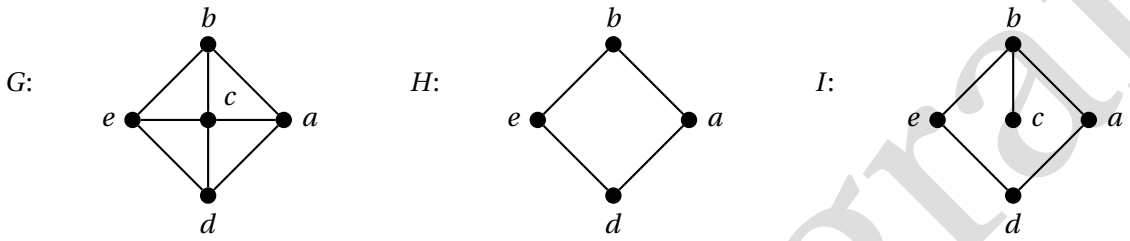


Figura 1.3: diagrama dos grafos G, H e I.

Um subgrafo $H \subseteq G$ onde $V(H) = V(G)$ é chamado de subgrafo gerador. No exemplo acima o grafo 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 $G[U] = K^k$ para algum natural k 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 $G[U] = (U, \varnothing)$ então U é chamado de conjunto independente ou conjunto estável de G, ou ainda, k-conjunto-independente ou k-estável no caso $|U| = k$.

Exemplo 4. O subgrafo G' do exemplo 2 é um 3-clique e G'' do exemplo 2 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 1. 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-difícil (veja [13], página 53), consequentemente, não é sabido se existe algoritmo que resolve algum desses dois problemas cuja complexidade de tempo é limitada por um polinômio em no tamanho de G.

(∗) Teorema de Mantel

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

Como G não contém triângulos a vizinhança de qualquer vértice é um conjunto estável. Se A um conjunto independente em G de cardinalidade máxima, então para todo vértice $v \in V$

$$d(v) \leq |A|. \tag{1.10}$$

Por A ser estável podemos classificar as arestas de G em duas classes: 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 G é $|E| = |E_1| + |E_2|$ e

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

Por outro lado, usando (1.10) deduzimos que

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

portanto

$$|E| \leq |A||\overline{A}| = |A|(|V| - |A|)$$

e o lado direito é máximo[1] quando $|A| = |V|/2$ ou $|A| = (|V| - 1)/2$, depeemdemdo da paridade de $|V|$, donde, finalmente, deduzimos

$$|E| \leq \frac{|V|^2}{4}. \tag{1.11}$$

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 $|E(G)| \leq |V(G)|^2/4$.* □

Ademais, há grafos sem triângulo para os quais vale a igualdade $|E(G)| = |V(G)|^2/4$. De fato, basta tomar para todo n par um grafo cujo conjunto de vértices é uma união $A \cup B$ de conjuntos disjuntos com $|A| = |B| = n/2$ e as arestas são todas aquelas com um extremo em A e o outro extremo em B. Tal grafo não tem triângulo e tem $|A| \cdot |B| = n^2/4$ arestas.

O Teorema de Mantel é um caso particular do famoso Teorema de Turán (veja o exercício 42), que foi o princípio de um ramo da Teoria dos Grafos chamada de Teoria Extremal de Grafos (veja mais sobre esse assunto em [4]).

Exercícios

Exercício 19. Quantos subgrafos tem o grafo $\big(\{1,2,3,4,5,6\}, \{\{1,2\}\}\big)$?

Exercício 20. Quantos subgrafos completos tem o grafo completo de ordem n?

Exercício 21. 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 contraexemplo para $G[U] = G[M]$.

Exercício 22. Descubra um subgrafo induzido de

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

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

Exercício 23. Mostre que em qualquer grafo G com pelo menos 6 vértices vale: ou G tem um 3-clique ou G tem um 3-conjunto-independente. (Dica: exercício 8 e princípio da casa dos pombos sobre $E_{K^6}(v)$, para algum vértice v.)

[1] Isso que pode ser deduzido do fato de que $a^2 + b^2 \geq 2ab$ que decorre de $(a-b)^2 \geq 0$, para quaisquer $a,\ b \in \mathbb{R}$.

Exercício 24. Dado um grafo G, denotamos por $\alpha(G)$ a cardinalidade do maior conjunto independente em G,

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

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

Exercício 25. Para todo grafo G, denotamos por $\omega(G)$ a cardinalidade do maior clique em G

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

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

Exercício 26. 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 27. 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 28. Redefina para todo grafo G o parâmetro $\chi(G)$ dado no exercício 9 em função dos conjuntos independentes de G.

Exercício 29. Prove que as duas desigualdades dadas a seguir valem para todo grafo G com pelo menos um vértice

$$\omega(G) \leq \chi(G) \tag{1.12}$$

$$\chi(G) \geq \frac{|V(G)|}{\alpha(G)}. \tag{1.13}$$

Exercício 30. Prove que todo grafo G satisfaz

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

1.3 Grafo bipartido e Cortes

Chamamos um grafo G de grafo bipartido se existem dois conjuntos estáveis A e B em G que particionam $V(G)$, isto é, A e B são tais que $A \cap B = \varnothing$, A e B são não vazios e $A \cup B = V(G)$.

Por exemplo, o seguinte grafo é bipartido

$$V(G) = \{1,2,3,4,5,6,7,8\} \tag{1.14}$$

$$E(G) = \big\{\{1,6\},\{6,2\},\{3,7\},\{3,8\},\{7,4\},\{7,5\}\big\}, \tag{1.15}$$

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 em G. Notemos que a bipartição pode não ser única. No caso do exemplo acima podemos escrever $V(G) = \{6,3,4,5\} \cup \{1,2,7,8\}$ pois tanto $\{6,3,4,5\}$ quanto $\{1,2,7,8\}$ são conjuntos independentes.

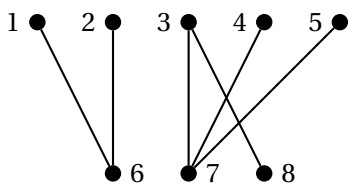


Figura 1.4: diagrama de um grafo bipartido definido por (1.14) e (1.15).

Para evitar ambiguidades escrevemos um grafo bipartido G com bipartição $\{A,B\}$ como $G=(A\cup B,E)$. Ademais, convencionamos que os grafos triviais e vazio são grafos bipartidos.

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

$$E(A,B)=\Big\{\{u,v\}\in E(G)\colon u\in A \text{ e } v\in B\Big\}; \tag{1.16}$$

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

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

Exemplo 5. A figura 1.5 abaixo mostra as arestas de $E(A,B)$ em azul para $A=\{0,1,8\}$ e $B=\{3,4,5,6\}$.

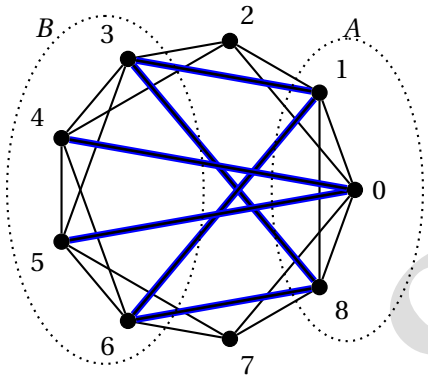


Figura 1.5: $E\big(\{0,1,8\},\{3,4,5,6\}\big)$ é formado pelas arestas $\big\{\{0,4\},\{0,5\},\{1,3\},\{1,6\},\{8,3\},\{6,8\}\big\}$.

O conjunto de arestas $E(A,\overline{A})$ é chamado de corte definido por A, também denotado por $\nabla_G(A)$. Da definição de corte podemos escrever, para todo grafo $G=(V,E)$, o conjunto de arestas de G como a seguinte união disjunta

$$E=E(G[A])\cup E(G[\overline{A}])\cup\nabla(A) \tag{1.17}$$

Exemplo 6. A figura 1.6 abaixo mostra o corte $\nabla(A)$ para $A=\{0,1,2,7,8\}$ formado pelas arestas que cruzam a reta vertical pontilhada

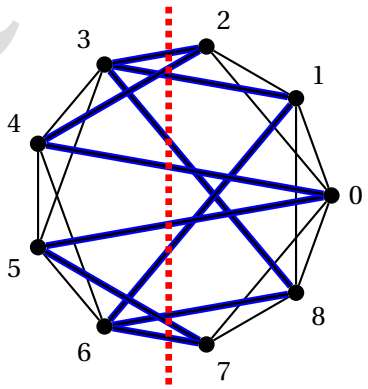


Figura 1.6: o corte $\big\{\{0,4\},\{0,5\},\{1,3\},\{1,6\},\{8,3\},\{6,8\},\{5,7\},\{6,7\},\{2,3\},\{2,4\}\big\}$ definido por $\{0,1,2,7,8\}$.

(∗) Um teorema de Erdős

Um resultado curioso exibido pelo matemático húngaro Paul Erdős afirma que todo grafo tem corte com pelo menos a metade das arestas do grafo. Seja G um grafo não-trivial qualquer. Tomemos $A\subset V(G)$ tal que o corte

definido por A tenha cardinalidade máxima sobre todos os cortes em G. Façamos H o subgrafo bipartido induzido pelas arestas do corte

$$H = \big(V(G), \nabla_G(A)\big).$$

Pela maximalidade do corte temos

$$d_H(v) \geq \frac{d_G(v)}{2} \tag{1.18}$$

pois, caso contrário, assumindo sem perda de generalidade que $v \in A$, temos que $A' = A \setminus \{v\}$ define um corte com mais arestas que o corte máximo (verifique). Usando o teorema 1 obtemos

$$|E(H)| = \frac{1}{2}\sum_v d_H(v) \geq \frac{1}{2}\sum_v \frac{d_G(v)}{2} = \frac{|E(G)|}{2}.$$

Teorema 3 (Erdős, 1965)**.** *Todo grafo G tem um corte de cardinalidade pelo menos $|E(G)|/2$.* □

O método probabilístico: Paul Erdős foi um do pioneiros e um grande divulgador do método probabilístico, um método não-construtivo para provar a existência de objetos matemáticos com um propriedade desejada. A ideia é mostrar que numa escolha aleatória de objetos num espaço de probabilidade apropriado, a probabilidade de que o resultado é do tipo prescrito é maior que zero, donde se conclui sua existência. Este método é aplicado a outras áreas da matemática, como a Teoria dos Números, a Análise, a Topologia e Geometria e a Ciência da Computação.

O teorema 3 pode ser provado com esse método. Seja G um grafo e tomemos $A \subseteq V(G)$ da seguinte forma: para cada vértice do grafo lançamos uma moeda, se o resultado for cara então colocamos o vértice em A, os resultados dos lançamentos são mutamente independentes. Dessa forma A define um corte em G. A probabilidade de uma aresta $\{u, v\} \in E(G)$ estar no corte $E(A, \overline{A})$ é a soma das probabilidades dos eventos "$u \in A$ e $v \notin A$" com "$u \notin A$ e $v \in A$". Cada um desses eventos tem probabilidade $1/4$, portanto, a probabilidade da aresta estar no corte é $1/2$. Desse modo, o número esperado de arestas no corte é $|E(G)|/2$, logo deve existir um corte com pelo menos $|E(G)|/2$ arestas.

Exercícios

Exercício 31. Seja G um grafo bipartido. Prove que todo subgrafo de G é bipartido.

Exercício 32. 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 33. Prove que G é bipartido se e somente se $\chi(G) < 3$.

Exercício 34. Um grafo é bipartido completo se tem todas as arestas possíveis entre as parte independentes. Formalmente, um grafo bipartido $G = (A \cup B, E)$ é dito bipartido completo se $E = \big\{\{a, b\} \subseteq V(G) \colon a \in A \text{ e } b \in B\big\}$. Um grafo bipartido completo sobre com partes de cardinalidade n e m é denotado por $K^{n,m}$. Determine $|E(K^{n,m})|$.

Exercício 35. Prove a afirmação da equação (1.17).

Exercício 36. Dê detalhes da prova da equação (1.18).

Exercício 37. Prove que para quaisquer $U, W \subseteq V(G)$, em um G é um grafo qualquer, vale que

$$E(U \triangle W, \overline{U \triangle W}) = E(U, \overline{U}) \triangle E(W, \overline{W}).$$

Exercício 38. Prove que para todo G e todo $U \subset V(G)$

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

Exercício 39. 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 $|\nabla(U)| = |N_G(U)|$? Justifique.

Exercício 40. Um grafo G é dito k-partido, para $k \in \mathbb{N}$, se existem k conjuntos independentes $A_1, A_2, \ldots, A_k$ 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 41. 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 superiormente por

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

Exercício 42 (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 43. Dê uma prova probabilística do seguinte fato: para todo G^n com $m > 0$ arestas vale que

$$\alpha(G^n) \geq \frac{n^2}{4m}. \tag{1.19}$$

Para tal sorteie $A \subseteq V(G^n)$ colocando vértices em A com probabilidade $n/(2m)$; conclua que o tamanho esperado de A é $n^2/(2m)$. Notemos que A pode não ser um conjunto independente; prove que a probabilidade de uma aresta ter ambos os extremos em A é $n^2/(4m^2)$ e que portanto o número esperado de vértices de A adjacente a outro vértice de A é $n^2/(4m)$. Dessa forma, se eliminamos um vértice de cada aresta, o que sobra é um conjunto independente. Conclua (1.19).

Exercício 44. Do exercício anterior temos $\alpha(G^n) \geq n/(2d(G))$. Deduza do Teorema de Turán que

$$\alpha(G^n) \geq \frac{n}{d(G)+1}.$$

Exercício 45. Dê uma prova probabilística para vale que

$$\alpha(G^n) \geq \frac{n}{d(G)+1}. \tag{1.20}$$

Para tal sorteie uma ordenação $v_1, \ldots, v_n$ de $V(G^n)$. Percorra a lista ordenada e coloque em A cada vértice que não tem vizinhos em A. Prove que a probabilidade de u não ter vizinho em A é $1/(d+1)$ e conclua que o tamanho esperado de A é

$$\sum_{u \in V} \frac{1}{d(u)+1}.$$

Verifique que A é independente e que seu tamanho esperado é pelo menos o lado direitos de (1.20).

1.4 Isomorfismo

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

$$f\colon V(G) \to V(H) \tag{1.21}$$

tal que

$$\{u, v\} \in E(G) \text{ se, e somente se, } \{f(u), f(v)\} \in E(H) \tag{1.22}$$

para todos $u, v \in V(G)$. Uma função f como acima é chamada de isomorfismo e, por abuso de notação, as vezes escrevemos $f\colon G \to H$ para um isomorfismo como definido acima.

Notemos 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. Em grande parte dos problemas em Teoria dos Grafos, senão na maioria, o que importa é como se dão as relações de adjacência e não importa como são nomeados os vértices de um grafo.

Exemplo 7. Há oito grafos distintos sobre um mesmo conjunto de vértices de cardinalidade três, eles estão descritos nas representações da figura 1.7 abaixo. No entanto, há apenas 4 grafos não-isomorfos com três vértices,

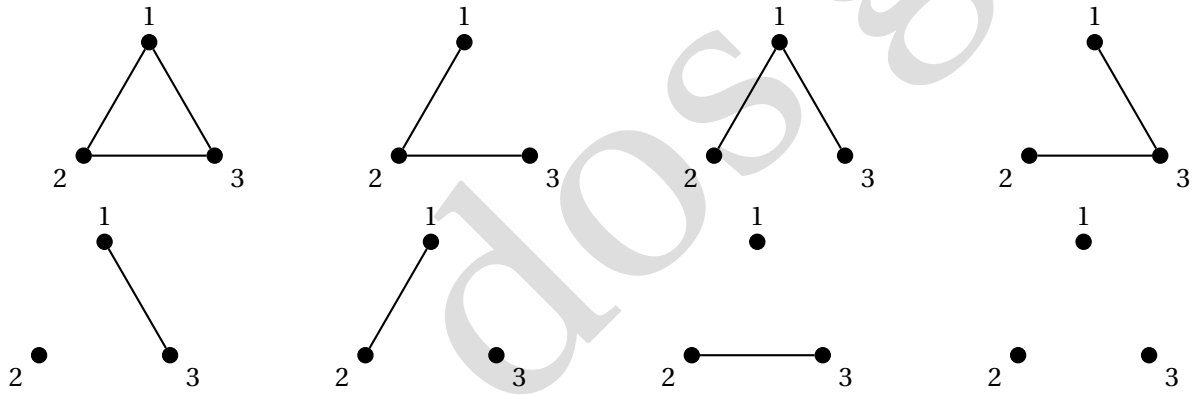


Figura 1.7: grafos distintos de ordem 3.

representados pelos diagramas da figura 1.8.

Figura 1.8: os tipos não-isomorfos de ordem 3.

Exemplo 8 (Grafo de Petersen). Os grafos representados na figura 1.9 são isomorfos pelo isomorfismo na tabela 1.1. Qualquer grafo isomorfo a eles é chamado de grafo de Petersen, é um dos grafos mais conhecidos na Teoria dos Grafos. Isso se deve ao fato dele ser o menor exemplo ou o menor contraexemplo para muitos problemas em grafos. Júlio Petersen (1839–1910) foi um matemático dinamarquês que por volta de 1898 construiu o grafo que leva seu nome como o menor contraexemplo para afirmação: *todo grafo conexo sem ponte admite uma 3-coloração própria das arestas.* Voltaremos a essa afirmação quando tivermos conhecimento dos termos técnicos que ela contém. O grafo de Petersen é retratado nas capas de vários periódicos científicos e livros sobre Teoria dos Grafos e Matemática Discreta.

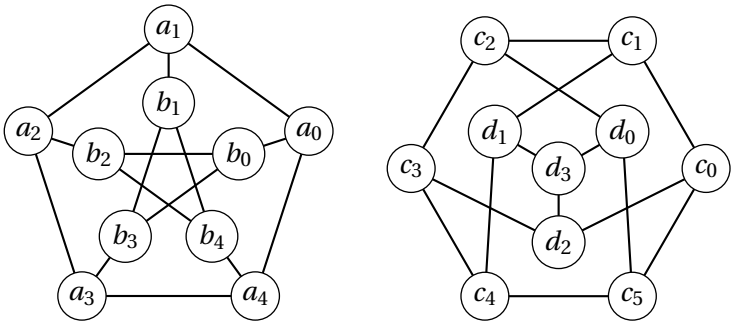


Figura 1.9: grafos isomorfos (grafo de Petersen).

v	$f(v)$	v	$f(v)$
a_0	c_3	b_0	c_4
a_1	d_2	b_1	d_3
a_2	c_0	b_2	c_5
a_3	c_1	b_3	d_1
a_4	c_2	b_4	d_0

Tabela 1.1: isomorfismo entre os grafos da figura 1.9.

Exemplo 9. Nenhum par dentre os grafos G, H e K representados na figura 1.10 são isomorfos. Temos que G não

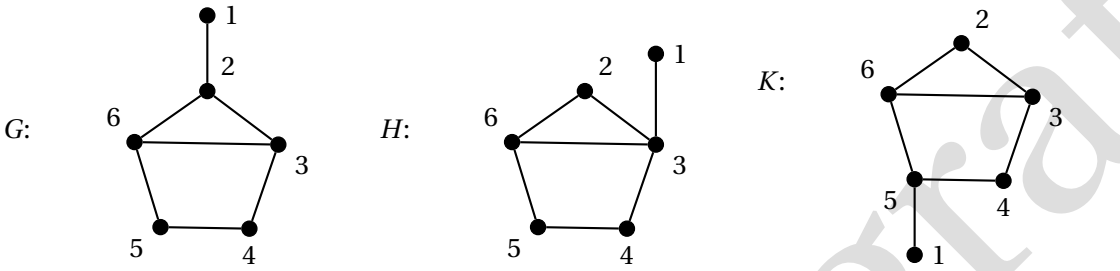


Figura 1.10: grafos não-isomorfos.

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 $f\colon G \to K$ então $f(1) = 4$ e $f(2) = 5$, portanto, a imagem por f do conjunto $\{2,3,6\} \subset V(G)$ é, obrigatoriamente, o conjunto $\{4,5,6\} \subset V(K)$ o que é contradição pois $\{2,3,5\}$ em G induz um triângulo e $f(\{2,3,5\})$ em K não induz triângulo (veja o exercício 50 abaixo).

Nesse exemplo acima 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). *Não existe uma caracterização eficiente de grafos isomorfos.*

Observação 2. Não há algoritmo eficiente que receba dois grafos e decida se eles são isomorfos.

- *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 dos grafos que resolva o problema. Entretanto, não é difícil projetar um algoritmo de tempo polinomial que recebe a terna (G,H,f) onde $f\colon G \to H$ 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, caso seja, um algoritmo eficiente para esse problema implica na existência de algoritmo eficiente para todo problema computacional em NP.

Também 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 no tamanho dos grafos que receba

como entrada uma terna (G, H, P) onde P é uma prova "curta" (i.e., de tamanho polinomial em $|V| + |E| + |E'|$) e que decida se P é uma prova do não-isomorfismo entre G e H. Em linguagem técnica dizemos que *não se sabe se o problema do não-isomorfismo de grafos está na classe* NP *de complexidade computacional.*

Exercícios

Exercício 46. Determine quais pares dentre os grafos abaixo são isomorfos.

(i) G_1 tal que $V(G_1) = \{v_1, u_1, w_1, x_1, y_1, z_1\}$ 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 tal que $V(G_2) = \{v_2, u_2, w_2, x_2, y_2, z_2\}$ 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 tal que $V(G_3) = \{v_3, u_3, w_3, x_3, y_3, z_3\}$ 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 47. Decida se o grafo representado pelo diagrama abaixo é o grafo de Petersen.

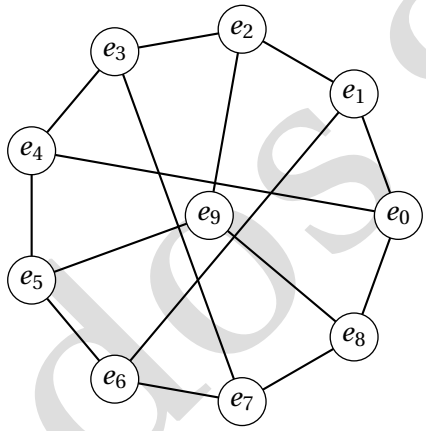


Figura 1.11: esse é o grafo de Petersen?

Exercício 48. Decida se os dois grafos 4-regular representados pelos dois diagramas abaixo são isomorfos.

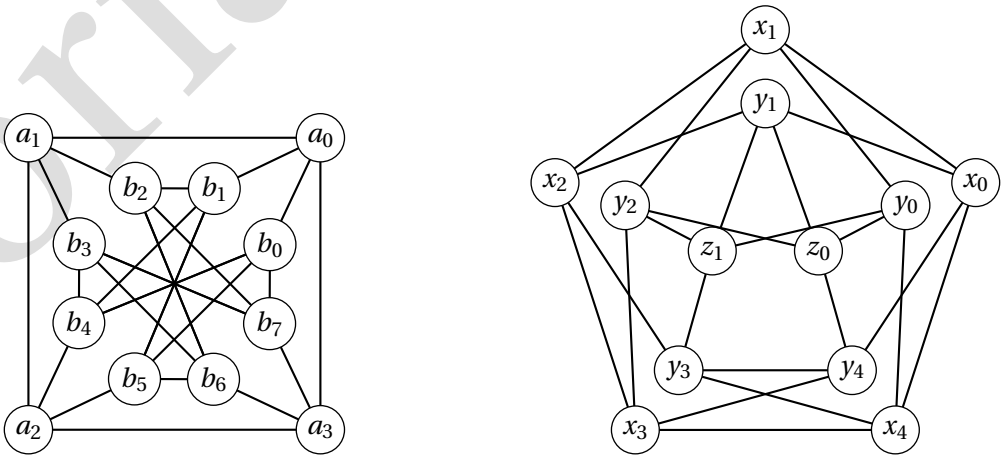


Figura 1.12: esses grafos são isomorfos?

Exercício 49. Mostre que existem 11 grafos não-isomorfos com 4 vértices.

Exercício 50. Sejam G e H grafos isomorfos e $f\colon G \to H$ um isomorfismo. É verdade que $G[U]$ é isomorfo a $H[f(U)]$ para todo $U \subseteq V(G)$? Justifique.

Exercício 51. Mostre que o grafo de Petersen é isomorfo ao complemento do grafo linha do K^5.

Exercício 52. Um automorfismo de um grafo é um isomorfismo do grafo sobre ele mesmo. Quantos automorfismos tem um grafo completo?

Exercício 53. Mostre que o conjunto de automorfismos de um grafo com a operação de composição de funções definem um grupo.

Exercício 54. Qual o número de grafos distintos sobre um conjunto de vértices V de tamanho n?

Exercício 55. Prove que há pelo menos $\frac{2^{\binom{n}{2}}}{n!}$ grafos não isomorfos sobre um conjunto de vértices V de tamanho n.

Exercício 56. Um grafo G é vértice-transitivo se para quaisquer $u, v \in V(G)$ existe um automorfismo f de G com $f(v) = u$. Analogamente, G é aresta-transitivo se para quaisquer arestas $\{x, y\}, \{z, w\} \in E(G)$ 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.

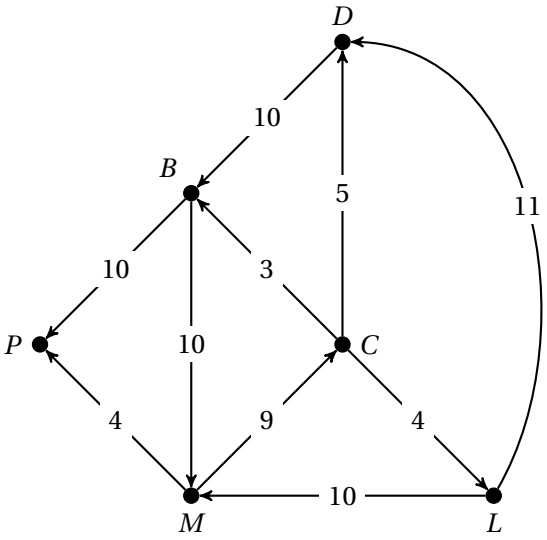
1.5 Outras noções de grafos

Em algumas situações precisamos de 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 as 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 contramão) 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

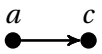
- grafo com pesos nas arestas é um tripla (V, E, ρ) de modo que (V, E) é um grafo e $\rho\colon E \to R$ é uma função que assume valores em $R \subseteq \mathbb{R}$;
- grafo orientado têm orientação nas arestas (são pares ordenados) de modo que para vértices u e v, se (u, v) é uma aresta então (v, u) não é aresta;
- grafo dirigido ou dígrafo 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.

Um exemplo de diagrama de grafo orientado com pesos nas arestas:



Usamos



para representar a aresta orientada (a, c).

Figura 1.13: exemplo de grafo orientado

Exercícios

Exercício 57. Defina isomorfismo para grafos orientados.

Exercício 58. Formule uma versão do teorema 1 para grafos orientados.

1.6 Representação computacional

Nesta seção começamos uma abordagem algorítmica da teoria dos grafos. O primeiro passo é definir representações computacionais de grafos; as representações que mostraremos a seguir assumem que os vértices são um segmento inicial dos inteiros positivos de modo que precisamos de um mapeamento do conjunto de vértices do grafo G no subconjunto $V' = \{1, 2, \ldots, |V(G)|\}$. Isso é feito para facilitar o acesso às informações usando estruturas de dados indexadas disponíveis nas linguagens de programação estruturadas.

Dado um grafo G, o primeiro problema é então construir um grafo isomorfo a G com vértices no conjunto V' de modo que se G é uma entrada para um problema computacional, então G é "traduzido" para sua cópia isomorfa e a partir daí o problema é tratado. Computada uma solução do problema sobre a cópia de G, devemos traduzir a solução obtida para uma solução em 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. Não trataremos dos detalhes aqui.

Em resumo, 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 $n \in \mathbb{N}$.

1.6.1 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. A representação de uma grafo por listas de adjacências não é única, pois qualquer permutação dos nós em $N(i)$ define uma lista válida.

Para o G grafo representado na figura 1.10, uma lista de adjacências é representada na figura 1.14.

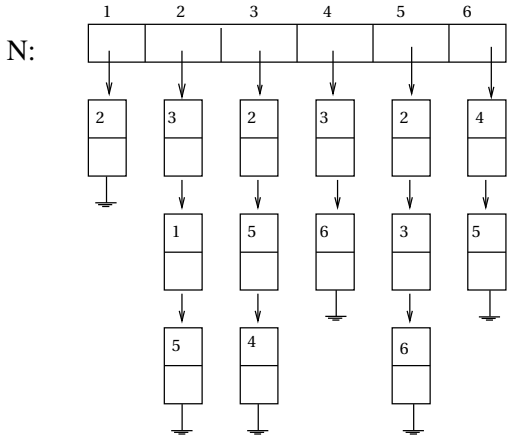


Figura 1.14: lista de adjacências do grafo G na figura 1.10.

1.6.2 Matriz de adjacências de G

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

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

Para o grafo G representado na figura 1.10, a matriz de adjacências é

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

1.6.3 Complexidade de algoritmos em grafos

Usualmente, a expressão que estabelece a complexidade de um algoritmo é escrita em função do tamanho n da representação de uma entrada para o algoritmo e em notação assintótica. No nosso caso, supondo que um algoritmo recebe um grafo como entrada, e só, n é o *tamanho da representação* do grafo, que no caso de matriz de adjacências é razoável admitirmos $n = |V|^2$ e no caso de listas de adjacências $n = |V| + |E|$, ou ainda nesse caso, usamos duas variáveis $(|V|, |E|)$. Neste texto entendemos por complexidade do algoritmo a

complexidade de tempo no pior caso

expressa em função do tamanho da representação de uma entrada para o algoritmo, ou seja, fixado um algoritmo, sua complexidade de pior caso é

$$T(n) = \max\{T(I) : I \text{ tem representação de tamanho } n\}$$

onde $T(I)$ denota o número de instruções executadas pelo algoritmo dado com entrada I.

Exemplo 10. Sejam A e B dois algoritmos distintos que resolvem o mesmo problema sobre um grafo $G = (V, E)$. O algoritmo A usa, no pior caso, exatos $|V|^2$ passos para executar a tarefa e o algoritmo B usa exatos $(|V| + |E|)\lfloor \log |E| \rfloor$ 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 $|V| \leq 16.644$).

Sejam f e g duas funções definidas nos naturais e que assumem valores naturais (ou, reais positivos), então

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

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

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

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 3. 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. É importante notar que "ter complexidade polinomial" é independente da representação do grafo ser por matriz ou lista de adjacências pois essas representações são *polinomialmente relacionadas* (isto é, uma pode ser obtida da outra por algoritmo de tempo polinomial). Ademais, entendemos por *algoritmo eficiente* um algoritmo com complexidade polinomial.

Exercícios

Exercício 59. Mostre que, para n suficientemente grande, se $f(n) = O(g(n))$ então $O(f(n)) + O(g(n)) = O(g(n))$.

Exercício 60. É verdade que para todo G^n vale que $|E(G^n)| = O(n)$ para n suficientemente grande?

Exercício 61. É verdade que para todo G^n vale que $\log |E(G^n)| = O(\log n)$ para n suficientemente grande?

Exercício 62. 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 63. 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 64. Em 1990, Coppersmith e Winograd [6] 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 de modo mais eficiente que $O(n^3)$?

Exercício 65. O traço de uma matriz A, denotado por $\text{tr}(A)$, é a soma dos elementos na diagonal da matriz. Quanto vale o traço de uma matriz de adjacências? Quanto vale $\text{tr}(A^2)$ em função de $E(G)$?

Exercício 66. 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 67. 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 68. 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 69. 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 70 ([7]). 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.

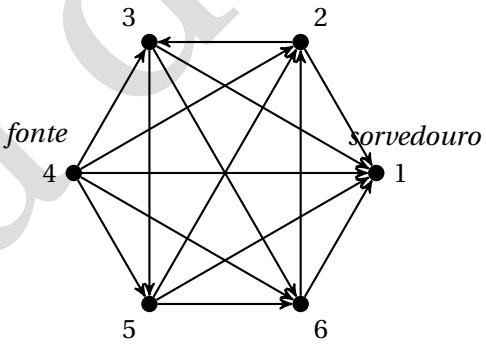


Figura 1.15: exemplo de um grafo orientado com fonte no vértice 4 e sorvedouro no vértice 1.

Notemos que esse resultado significa que não é preciso ler toda a entrada para resolver o problema.

2 | Caminhos, circuitos e percursos em grafos

Neste capítulo estudamos uma classe de grafos, ditos caminhos, e as noções e parâmetros que permeiam essa classe como por exemplo o comprimento de caminhos, a distância entre seus vértices e o diâmetro do grafo, depois apresentamos a classe dos circuitos. Em seguida apresentamos um algoritmo fundamental por ser a base sobre a qual muitos outros algoritmos são construídos, tal algoritmo percorre um grafo visitando seus vértices e arestas, ou seja, o único objetivo de um percurso é sistematicamente visitar todos os vértices e todas as arestas do grafo. Definindo políticas de gerenciamento desse percurso temos os conhecidos algoritmos de busca em largura e em profundidade. A última seção trata do problema algorítmico de determinar distâncias entre pares de vértices num grafo em que pesos positivos nas arestas representam comprimentos dessas 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 Passeios, caminhos e circuitos

Uma sequência W de vértices de um grafo G

$$W = v_1, v_2, \ldots, v_k$$

é chamada de passeio em G se v_i e v_{i+1} são adjacentes, para todo $i \in \{1,2,\ldots,m-1\}$, ou seja, um passeio em G é *qualquer* sequência de vértices de G de modo que vértices consecutivos são adjacentes. Por exemplo, no grafo do exemplo 1 a sequência $1,2,5,2,1,5$ define um passeio, também a sequência $1,2,5,2,1$ define um passeio que nesse caso chamamos de fechado. Um passeio como W é um passeio fechado se $v_1 = v_k$.

Dado um vértice v em $V(G)$, dizemos que v alcança o vértice u em G se existe um passeio

$$v = v_1, v_2, \ldots, v_m = u$$

em G. Convencionamos que v alcança v em G, para todo $v \in V(G)$. Notemos que se v alcança u em G então u alcança v em G e que, também vale, que se v alcança u e u alcança w então v alcança w. Dito isso, podemos concluir que *alcança* é uma relação de equivalência sobre o conjunto $V(G)$. As classes de equivalência dessa relação são os subconjuntos dos vértices alcançáveis entre si, chamadas de componentes conexas do grafo. Um grafo com uma única classe de equivalência é dito conexo.

Um caminho é um passeio em que vértices não são repetidos, ademais caminho também designa uma classe de grafos. Formalmente, um caminho é qualquer grafo ou subgrafo isomorfo a (V,E) dado por

$$\begin{aligned} V &= \{1,2,\ldots,k\} \\ E &= \big\{\{i,i+1\} : 1 \leq i < k\big\}, \end{aligned} \tag{2.1}$$

para algum $k \in \mathbb{N}$. Um caminho com quatro vértices é representado pelo diagrama da figura 2.1. O caminho P definido em (2.1) também é denotado pela sequência

Figura 2.1: um caminho P^4.

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

de seus vértices, de modo que vértices consecutivos na sequência são adjacentes. Seguindo notação estabelecida na seção 1.1, denotamos por P^k um caminho com k vértices.

Num caminho $v_0, \ldots, v_k$, dizemos que os vértices v_0 e v_k são os seus extremos e dizemos que os vértices v_i, $0 < i < k$, são os seus vértices internos. 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 $\text{dist}_G(u, v)$, é definida como o comprimento do menor caminho com extremos u e v, isto é

$$\text{dist}_G(u, v) = \min\{|E(P)| : u, v \text{ são extremos de um caminho } P \subseteq G\}, \tag{2.2}$$

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

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

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

Por exemplo, consideremos o grafo G representado no diagrama da figura 2.2. O passeio $P^4 = a, b, f, i$ é um caminho no grafo definido pelo diagrama, ainda, esse mesmo caminho é definido pelo subgrafo induzido $G[\{a, b, f, i\}]$. Também o subgrafo definido pelo par de subconjuntos $\big(\{a, d, c, h, i\}, \{\{a, d\}, \{d, c\}, \{c, h\}, \{h, i\}\}\big)$ é um caminho nesse grafo G. O subgrafo induzido pelo conjunto de vértices $\{m\}$ é um caminho P^1 e o subgrafo induzido pela aresta $\{j, l\}$ é um caminho P^2. Nesse grafo da figura 2.2 temos $\text{dist}_G(a, j) = \infty$, $\text{dist}_G(a, b) = 1$, $\text{dist}_G(a, i) = 3$, $\text{dist}_G(m, j) = \infty$. Observemos que um caminho de comprimento mínimo pode não ser único.

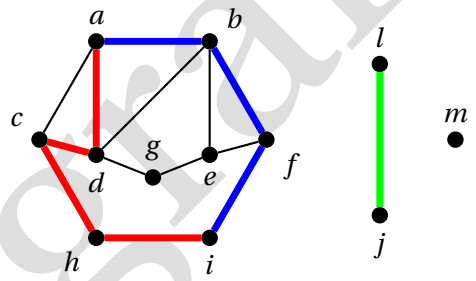


Figura 2.2: exemplos de caminhos em um grafo.

A função $\text{dist}_G : V(G) \times V(G) \to \mathbb{N}$ é uma *métrica* em $V(G)$ pois satisfaz

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

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

3) a *desigualdade triangular*

$$\text{dist}(u, w) \leq \text{dist}(u, v) + \text{dist}(v, w) \tag{2.4}$$

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

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

$$\text{diam}(G) = \max_{u,v \in V(G)} \text{dist}_G(u, v). \tag{2.5}$$

Naturalmente, $\text{diam}(G) = \infty$ se existirem dois vértices no grafo G que não são extremos de um caminho. O grafo de Petersen, por exemplo, tem diâmetro 2, o grafo da figura 2.2 tem diâmetro ∞.

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

$$\begin{aligned} V &= \{1, \ldots, k\} \\ E &= \big\{\{i, i+1\} : 1 \leq i \leq k-1\big\} \cup \big\{\{k, 1\}\big\}, \end{aligned} \tag{2.6}$$

Figura 2.3: um circuito C^4.

para algum $k \geq 3$. Um circuito com quatro vértices é representado pelo diagrama da figura 2.3. Como no caso do caminho, para simplificar, o circuito definido em (2.6) também é denotado pela sequência de seus vértices

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

de modo que vértices consecutivos são adjacentes. Por exemplo, o grafo G definido a partir da figura 2.2; nele temos que $C^4 = G[\{b,d,e,g\}]$ é um circuito, bem como o subgrafo $C^5 = (V,E)$ dado por $V = \{a,b,d,e,g\}$ e $E = \{\{a,b\},\{b,e\},\{e,g\},\{g,d\},\{d,a\}\}$. Também é um circuito o subgrafo induzido pelas arestas cuja cor seja ou azul ou vermelha.

O número de arestas num circuito é o comprimento desse circuito e denotamos um circuito de comprimento k por C^k. Definimos a cintura do grafo G como o comprimento do menor circuito contido em G, ou seja

$$\operatorname{cin}(G) = \min\{|E(C)| : \text{ existe um circuito } C \subseteq G\}, \tag{2.7}$$

quando existe um circuito, caso contrário convencionamos que $\operatorname{cin}(G) = \infty$. Por exemplo, o grafo de Petersen tem cintura 5, o grafo da figura 2.2 tem cintura 3 e um caminho tem cintura ∞.

Proposição 4. *Todo grafo G contém um caminho de comprimento pelo menos $\delta(G)$ e, se $\delta(G) \geq 2$, contém 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 nos casos em que $\delta(G) \geq 2$. Seja $P = x_0, x_1, \ldots, x_k$ um caminho de comprimento máximo em G. De P ter comprimento máximo em G temos que $N_G(x_0) \subsetneq V(P)$, caso contrário, haveria caminho mais longo

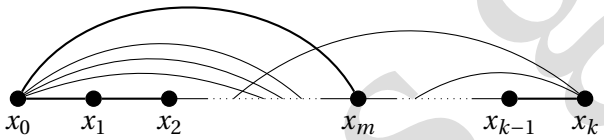


Figura 2.4: se $P = x_0, \ldots, x_k$ é de comprimento máximo em G então seus extremos devem ter a vizinhança em $V(P)$.

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 o 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

$$m = \max\{j : 1 < j \leq k \text{ e } x_j \in N_G(x_0)\}, \tag{2.8}$$

que é o maior índice em $\{2,\ldots,k\}$ de um vértice vizinho de x_0 em P. Basta notarmos 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

$$m = |\{x_1, x_2, \ldots, x_m\}| \geq |N_G(x_0)| \geq \delta(G).$$

Portanto, C tem comprimento pelo menos $\delta(G)+1$. □

Circuito ímpar é como chamamos um circuito de comprimento ímpar. Circuitos ímpares caracterizam grafos bipartidos no sentido de que classe dos grafos livres de circuitos de comprimento ímpar e a classe dos grafos bipartidos coincidem.

Teorema 5. *Um grafo G é bipartido se e somente se G não contém circuitos de comprimento ímpar.*

Demonstração. Começamos por notar que um circuito ímpar não é um grafo bipartido e como subgrafo de grafo bipartido também é bipartido (exercício 31), concluímos que grafos bipartidos não contêm circuito ímpar.

Agora, seja G um grafo sem circuito ímpar e vamos mostrar que G é bipartido. Se G não contém circuito ímpar, então nenhuma de suas componentes conexas contém circuito ímpar. Seja $W = \{w_1, w_2, \ldots, w_m\}$ um subconjunto

de $V(G)$ composto por um representante de cada componente conexa de G. Para cada i, seja H_{w_i} o subgrafo de G induzido pelos vértices que w_i alcança em G, isto é

$$H_{w_i} = G\big[\{v \in V(G) \colon \text{dist}_G(w_i, v) < \infty\}\big].$$

Vamos mostrar que H_{w_i} é bipartido. Definimos os conjuntos

$$A_{w_i} = \{v \in V(H_{w_i}) \colon \text{dist}_G(w_i, v) = 2j \text{ para algum } j \in \mathbb{N}\} \tag{2.9}$$

$$B_{w_i} = \{v \in V(H_{w_i}) \colon \text{dist}_G(w_i, v) = 2j+1 \text{ para algum } j \in \mathbb{N}\} \tag{2.10}$$

com $A_{w_i} \cup B_{w_i} = V(H_{w_i})$; esses conjuntos são, respectivamente, formados pelos vértices que estão a distância par de w_i e pelos os que estão a distância ímpar de w_i. Vamos mostrar que esses conjuntos são independentes. Podemos assumi-los não vazios (por quê?).

Sejam $u, v \in A_{w_i}$ distintos, P o caminho $w_i = x_1, \ldots, x_{2a} = u$ que realiza a distância $\text{dist}_G(w_i, u)$ e Q o caminho $w_i = y_1, \ldots, y_{2b} = v$ que realiza a distância $\text{dist}_G(w_i, v)$. Se os vértices internos de P e Q são disjuntos então a existência da aresta $\{u, v\} \in E(G)$ 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 vale a igualdade $p = q$ (verifique esse fato) 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, logo tais vértices não são adjacentes. Com isso temos que H_{w_i} é um subgrafo bipartido de G.

Os subgrafos $H_{w_1}, \ldots, H_{w_m}$ definidos como acima são bipartidos e disjuntos, isto é

$$V(H_{w_i}) \cap V(H_{w_j}) = \varnothing, \text{ para todos } i \neq j \tag{2.11}$$

e

$$G = \bigcup_{i=1}^{m} H_{w_i} \tag{2.12}$$

logo G é bipartido, pois o resultado da união de grafos bipartidos disjuntos é um grafo bipartido (verifique). □

Exercícios

Exercício 71. 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 72. Escreva um algoritmo que recebe uma grafo G e decide se G é um caminho. Determine a complexidade do algoritmo.

Exercício 73. Sejam $P = u_1, u_2, \ldots, u_k$ e $Q = v_1, v_2, \ldots, v_\ell$ duas sequências de vértices de um grafo. Definimos a concatenação dessas sequências como a sequência 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 sequência P,Q é caminho?

Exercício 74. 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 subsequência $u_1, u_{i_1}, u_{i_2}, \ldots, u_{i_t}, u_k$ é um caminho em G. Também, prove que se $u_1, v_2, \ldots, v_{k-1}, u_k$ é um passeio distinto do anterior, então G contém circuito.

Exercício 75. 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 76. Seja $C \subseteq G$ um circuito no grafo G. Prove que para qualquer $U \subseteq V(G)$ vale que $|E(C) \cap \nabla(U)|$ não pode ser ímpar.

Exercício 77. Prove a desigualdade triangular (equação (2.4)).

Exercício 78. Prove as afirmações feitas nas equações (2.11) e (2.12).

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 conexo. 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 k um número natural. O k-cubo é o grafo cujo conjunto de vértices são as sequências 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.

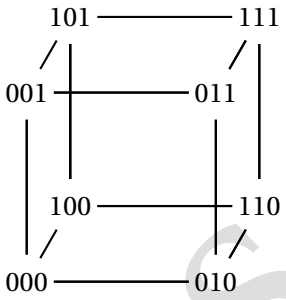


Figura 2.5: um diagrama do 3-cubo.

Exercício 83. Seja k um número natural. O grafo de Fibonacci F_k é o grafo cujo conjunto de vértices são as sequências binárias de tamanho k sem ocorrências de 1 consecutivos; 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 F_k. Determine os parâmetros α, ω e χ do grafo F_k.

Exercício 84. Seja p um número primo e tome $V = \{0,1,\ldots,p-1\}$. Defina o grafo bipartido 3-regular $G = (A \cup B, E)$ com $A = V \times \{a\}$ e $B = V \times \{b\}$, ou seja, o que queremos é que A e B sejam duas cópias disjuntas de V. Cada vértice $(x,a) \in A$ é adjacente aos vértices (x,b), $(x+1,b)$ e $(x+y,b)$ de B onde $y \neq 0,1$ é fixo e a soma é feita módulo p. Prove que qualquer subgrafo induzido de G de ordem $p+1$ contém um caminho de comprimento 2 quando $y = 2, (p+1)/2, p-1$. O mesmo vale para qualquer $y \neq 0,1$ de V?

2.2 Percurso genérico

Seja G um grafo conexo. Um percurso genérico é um algoritmo que visita os vértices e arestas de G. Em qualquer instante durante a execução de um percurso em G valem as seguintes considerações:

- cada vértice $u \in V(G)$ está em um de dois estados possíveis: *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*.

De início todos os vértices de G estão no estado *não-visitado* e todas as arestas no estado *não-visitada* a partir de ambos os extremos. O algoritmo que descreveremos a seguir, que chamaremos de Visite(G,v), recebe um grafo G e um vértice inicial $v \in V(G)$; visita uma única vez todos os vértices a partir de v e visita duas vezes todas as

arestas que incidem nesses 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.

```
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;
```

Algoritmo 1: Visite(G, v).

Análise do algoritmo.

No que segue faremos uma análise do algoritmo Visite(G, v) para determinar sua correção e sua complexidade. 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).

Proposição 6. *Cada vértice de G entra em L no máximo uma vez.*

Demonstração. Um vértice w é inserido em L na execução das linhas 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 e 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 7. *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 e como consequência 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 8. *O algoritmo* Visite(G, v) *termina.*

Demonstração. Por definição de grafo o conjunto $|V(G)|$ é finito. Pela proposição 6 cada vértice entra em L no máximo uma vez. Pela proposição 7 após no máximo

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

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

Lema 9. *Numa execução de* Visite(G, v) *todo vértice que v alcança é visitado.*

Demonstração. Seja $w \in V(G)$ um vértice que v alcança. Por definição existe um passeio

$$v = v_1, v_2, \ldots, v_{m-1}, v_m = w \tag{2.13}$$

em G. Vamos provar por indução em m que w é visitado.

Para a base da indução, suponha $m = 1$. Como v alcança v e é visitado (linha 2), a base está provada. Suponhamos (hipótese da indução) que todo vértice que v alcança via um passeio de $m-1$ vértices ($m > 1$) é visitado. Seja w um vértice que v alcança por um passeio com m vértices, como em (2.13). Façamos $e_m = \{v_{m-1}, v_m\}$. Como v_{m-1} é alcançável por um passeio com $m-1$ vértices, por hipótese v_{m-1} foi visitado, portanto, entrou em L. Pelo corolário8 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 um passeio com m vértices é visitado, para todo inteiro $m \geq 1$. □

Corolário 10. *Cada aresta de G, cujos extremos são alcançáveis, é visitada duas vezes.*

Demonstração. Seja $\{x, y\}$ uma aresta de G. Como $x \in V(G)$ 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 $y \in V(G)$ 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. □

Teorema 11. *Após* Visite(G, v) *cada vértice foi visitado uma vez e cada aresta de 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 para a complexidade 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; para esses casos basta manter um ponteiro em cada lista de adjacências $N(i)$ indicando o primeiro vértice não visitado dela, os detalhes são deixados a cargo do leitor.

Teorema 12. *Para todo* $G = (V, E)$ *a complexidade de* Visite(G, v) *é* $O(|V| + |E|)$. □

2.3 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 1 visto acima, cada caso é obtido quando definimos uma política de gerenciamento para a lista L.

Nos algoritmos a seguir consideraremos, explicitamente, apenas as visitas aos vértice do grafo.

2.3.1 Busca em Largura

A lista L é gerenciada como uma fila, inserção é feita no fim da fila e remoção é feita no começo da fila.

```
1 insira v no fim da fila L;
2 marque v visitado;
3 enquanto L ≠ ∅ faça
4 |  tome u o primeiro da fila L ;
5 |  para cada w ∈ N_G(u) faça
6 |  |  se w é não-visitado então
7 |  |  |  insira w no fim da fila L;
8 |  |  |  marque w visitado;
9 |  remova u de L.
```

Algoritmo 2: Visite_em_Largura(G, v).

Como na análise do algoritmo 1 concluímos o seguinte.

Teorema 13. *Dado um grafo conexo $G = (V, E)$, uma execução de Visite_em_Largura(G, v) visita uma vez cada vértice do grafo em tempo $O(|V| + |E|)$.* □

2.3.2 Busca em Profundidade

Numa busca em profundidade o algoritmo 1 insere elementos no topo da pilha L, a remoção é feita no topo da pilha L e a linha 4 devolve (sem remover) o topo da pilha L.

De modo análogo aos casos anteriores concluímos o seguinte.

Teorema 14. *Dado um grafo conexo G, uma execução de busca em profundidade em $G = (V, E)$ visita uma vez cada vértice do grafo em tempo $O(|V| + |E|)$.* □

A seguinte versão recursiva para a busca em profundidade, que chamamos de *busca em profundidade rotulada*, será usada adiante nesse texto; no início temos $cont = sai(v) = chega(v) = 0$ e $pai(v) = nil$, para todo vértice v de G.

```
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.
```

Algoritmo 3: BP(G, w).

A ideia é manter uma variável contadora das operações de inserção e remoção na pilha L. Esse contador é usado para rotular cada vértice com dois valores inteiros: o valor do contador quando o vértice v entrou na pilha, $chega(v)$, e o valor do contador quando o vértice v saiu da pilha, $sai(v)$. 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.

Proposição 15. *Após uma execução de* 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) chega$(u) < sai(u) < chega(v) < sai(v)$;

(a') chega$(v) < sai(v) < chega(u) < sai(u)$;

(b) $chega(u) < chega(v) < sai(v) < sai(u)$;
(c) $chega(v) < chega(u) < sai(u) < sai(v)$.

Demonstração. A prova é deixada como exercício. □

Exercícios

Exercício 85. Escreva uma versão não recursiva da busca em profundidade rotulada e analise sua complexidade.

Exercício 86. Prove a proposição 15.

Exercício 87. Considere uma execução da busca rotulada BP(G, w) e defina, para todo $t \in \mathbb{N}$, a t-ésima iteração do vetor pai, denotada por $pai^{(t)}$, da seguinte maneira

$$pai^{(t)}(v) = \begin{cases} v & \text{se } t = 0 \\ pai^{(t-1)}\big(pai(v)\big) & \text{se } t > 0. \end{cases} \tag{2.14}$$

Prove as seguintes equivalências: para quaisquer $u, v \in V(G)$

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

Exercício 88. Consideremos o seguinte algoritmo sobre um grafo conexo 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.
```

Algoritmo 4: Visite_vértices(G).

(a) O algoritmo visita todos os vértices de G? (Pode-se assumir que $E(L, \overline{L}) \neq \varnothing$ se, e só 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. Pode-se assumir que as operação em L e em M são feitas em tempo constante.

Exercício 89. Seja G um grafo nas condições do exercício 88 e seja M o conjunto construído por Visite_vértices(G), algoritmo 4. Prove que $G[M]$ não contém circuito.

Exercício 90. Prove que o seguinte algoritmo (que está baseado numa busca em largura) funciona corretamente. Determine a complexidade do algoritmo.

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

```
1  para cada v ∈ V(G) faça dist(w, v) ← ∞;
2  insira w no fim da fila L;
3  dist(w, w) ← 0;
4  enquanto L ≠ ∅ faça
5      tome u o primeiro elemento da fila L;
6      para cada v ∈ N_G(u) faça
7          se dist(w, v) = ∞ então
8              insira v no fim da fila L;
9              dist(w, v) ← dist(w, u) + 1;
10     remova u de L;
```

Algoritmo 5: Distância(G, w).

2.4 Caminhos mínimos em grafos com pesos nas arestas

Nesta seção consideramos grafos com pesos positivos 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, positivo no nosso caso, a cada aresta de E.

A ideia é que esses pesos representem comprimento (ou o custo de travessia) da aresta, com isso o comprimento de um caminho $P = x_0, x_1, \ldots, x_k$ em (V, E, ρ) é a soma dos pesos 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,

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

quando existe algum caminho, caso contrário convencionamos que $\text{dist}_G(u, v) = \infty$ tal que vale (2.3) para todo $n \in \mathbb{R}^+$. Um caminho com extremos u e v de comprimento $\text{dist}_G(u, v) < \infty$ é chamado de caminho mínimo.

O diagrama da figura 2.6 representa um grafo com pesos nas arestas e uma lista de adjacências para esse grafo é representada na figura 2.7. Nesse exemplo $\text{dist}(a, e) = 10$ e um caminho mínimo é a, c, b, d, f, e.

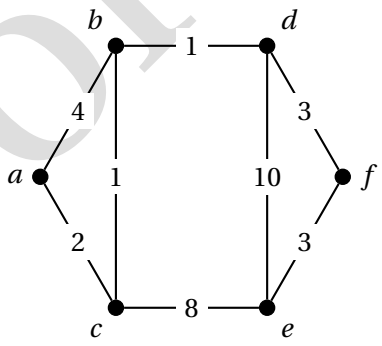


Figura 2.6: diagrama de um grafo com pesos nas arestas.

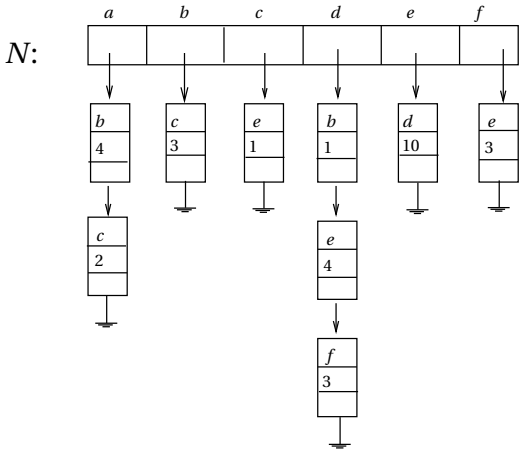


Figura 2.7: lista de adjacências do grafo da figura 2.6.

Para um grafo G qualquer com pesos positivos nas arestas e s um vértice de G as seguintes propriedades são satisfeitas:

desigualdade triangular: vale a seguinte variante da desigualdade triangular (2.4): para todo $\{v,u\} \in E(G)$

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

subestrutura ótima: Se $P = v_1, v_2, \ldots, v_k$ é um caminho mínimo num grafo G então $P_{i,j} = v_i, \ldots, v_j$ é um caminho mínimo com extremos v_i e v_j em G para quaisquer i, j com $1 \leq i < j \leq k$.

Exercícios

Exercício 91. Prove as propriedades *desigualdade triangular* e *subestrutura ótima* enunciadas acima.

Exercício 92. 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.

2.4.1 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 $v \in V$. Usaremos um vetor $d(\,)$ indexado pelos vértices e um subconjunto $S \subseteq V$ da seguinte forma

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

$d(\,)$: em qualquer instante da execução do algoritmo $d(v)$ armazena um valor que significa

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

(b) ou, caso $v \in \overline{S}$, o comprimento de algum caminho não necessariamente mínimo com extremos s e v;

(c) ou $d(v) = \infty$, caso v não tenha sido visitado.

O algoritmo que descrevemos a seguir recebe G e s como acima, determina $\text{dist}_G(s,x)$, para todo $x \in V$ e funciona da seguinte forma. De início $d(s) = 0$, pois s está a distância 0 dele mesmo, logo $S = \{v\}$; para qualquer vértice $v \neq s$ do grafo temos $d(v) = \infty$.

Num momento qualquer da execução em que valha $\overline{S} \neq \varnothing$ o conjunto S contém os vértices cujas distâncias já estão determinadas. Retiramos de $\overline{S}$ o vértice u que tem menor valor de $d(\,)$. Esse 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 colocá-lo em S. Em seguida, visitamos cada vizinho t de u. Pela desigualdade triangular, equação (2.15), $\text{dist}(s,t) \leq d(u) + \rho(\{u,t\})$ e testamos se um caminho de menor comprimento foi encontrado e se for o caso, então atualizamos $d(\,)$, essa etapa é chamada de Relaxação

Relaxação(u,t):

se $d(t) > d(u) + \rho(\{u,t\})$ **então** $d(t) \leftarrow d(u) + \rho(\{u,t\})$;

notemos que a condição na Relaxação é sempre falsa para $t \in S$. O algoritmo prossegue até que $S = V$. Esse algoritmo é conhecido como algoritmo de Dijkstra.

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 para cada v ∈ V(G) faça d(v) ← ∞ ;
2 d(s) ← 0;
3 S ← ∅;
4 enquanto S̄ ≠ ∅ faça
5 |   seja u tal que d(u) = min{d(v) : v ∈ S̄};
6 |   insira u em S;
7 |   para cada t ∈ N(u) faça
8 |   |   se d(t) > d(u) + ρ({u, t}) então d(t) ← d(u) + ρ({u, t});
9 devolva d.
```

Algoritmo 6: Dijkstra(G, s).

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

Análise do algoritmo de Dijkstra.

Vamos mostrar que o algoritmo funciona corretamente.

Proposição 16. *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 se* $d(v) = \text{dist}(s, v)$ *o valor de* $d(v)$ *não muda.*
(b) *Se* $\text{dist}(s, v) = \infty$ *então* $d(v) = \infty$.

Demonstração. Para provar (a) vamos supor o contrário e assumir que $w \in V$ é o primeiro vértice para o qual ocorreu que

$$d(w) < \text{dist}(s, w) \tag{2.16}$$

durante uma execução e seja $u \in V$ o vértice que causou a mudança no valor de $d(w)$ numa relaxação, assim a relaxação faz

$$d(w) = d(u) + \rho(\{u, w\}) \tag{2.17}$$

e nesse momento da execução

$$d(u) \geq \text{dist}(s, u)$$

pois w foi a primeira ocorrência do fato suposto na equação (2.16), logo

$$d(u) + \rho(\{u, w\}) \geq \text{dist}(s, u) + \rho(\{u, w\}). \tag{2.18}$$

Das três equações (2.16), (2.17) e (2.18) acima deduzimos

$$\begin{aligned} \text{dist}(s, w) &> d(u) + \rho(\{u, w\}) \\ &\geq \text{dist}(s, u) + \rho(\{u, w\}) \end{aligned}$$

contradizendo a desigualdade triangular, equação (2.15). Ademais uma Relaxação$(\cdot, v)$ nunca aumenta o valor $d(v)$, logo uma vez que $d(v) = \text{dist}(s, v)$ esse valor não muda.

Para provar (b) notemos que, por (a), $d(v) \geq \text{dist}(s, v) = \infty$ portanto nunca será atribuído um valor real para $d(v)$. □

Proposição 17. *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

$$d(v) \leq d(x_k) + \rho(\{x_k, v\})$$

e por hipótese

$$d(x_k) + \rho(\{x_k, v\} = \text{dist}(s, x_k) + \rho(\{x_k, v\})$$

logo

$$d(v) \leq \text{dist}(s, x_k) + \rho(\{x_k, v\}).$$

Pela propriedade da *subestrutura ótima* (página 35)

$$\text{dist}(s, x_k) + \rho(\{x_k, v\}) = \text{dist}(s, v)$$

portanto, $d(v) \leq \text{dist}(s, v)$. Pela proposição 16(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 18. *Antes de cada iteração do* **enquanto** *na linha 4 vale que*

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

Demonstração. A prova é por contradição. Suponha que

$$\left\{t \in \mathbb{N} : \text{antes da } t+1\text{-ésima rodada existe } u \in S \text{ tal que } \text{dist}(s, u) \neq d(u)\right\} \neq \varnothing.$$

Pelo Princípio da Boa-Ordenação, esse conjunto tem um mínimo k. Na primeira iteração, temos $S = \{s\}$ e $d(s) = \text{dist}(s, s)$, portanto $k \geq 1$. Seja u o primeiro vértice inserido em S e com $d(u) \neq \text{dist}(s, u)$, ou seja, u é o vértice inserido na k-ésima iteração do laço. Congelemos a execução no momento em que u é escolhido na linha 5.

Existe $P = s, x_1, \ldots, x_k, u$ um caminho mínimo com extremos s e u em G, caso contrário $d(u) = \infty$, pelo proposição 16, logo teríamos $d(u) = \text{dist}(s, u)$. Nesse momento da execução $s \in S$ e $u \in \overline{S}$, portanto $s \neq u$.

Tomemos i como o menor inteiro $1 \leq i \leq k$ tal que $\{x_i, x_{i+1}\} \in E(P) \cap E(S, \overline{S})$, isto é, $\{x_i, x_{i+1}\}$ é a primeira aresta de P no sentido de s para u com $x_i \in S$ e $x_{i+1} \in \overline{S}$. Notemos que é possível termos $x_i = s$ e $x_{i+1} = u$.

Agora, quando x_i entrou em S temos $d(x_i) = \text{dist}(x_i, s)$ e ocorreu Relaxação(x_i, x_{i+1}) o que fez com que $d(x_{i+1}) = \text{dist}(s, x_{i+1})$ pela proposição 17. Além disso, $\text{dist}(s, x_{i+1}) \leq \text{dist}(s, u)$ pois os pesos são não-negativos e P é mínimo, de modo que

$$d(x_{i+1}) = \text{dist}(s, x_{i+1}) \leq \text{dist}(s, u) \leq d(u)$$

onde a última desigualdade vem da proposição 16(a). Mas, $d(u) \leq d(x_{i+1})$ pois ambos estão em $\overline{S}$ e u foi escolhido logo vale a igualdade, assim na equação acima $d(u) = \text{dist}(s, u)$ e temos uma contradição. □

A correção do algoritmo de Dijkstra segue facilmente do teorema acima; cada iteração remove um vértice de $\overline{S}$, logo após $|V|$ iterações $\overline{S} = \varnothing$, ou seja $S = V(G)$, e a afirmação do teorema é que $d(v)$ está correto para todo $v \in V(G)$.

Agora, vamos analisar a complexidade do algoritmo de Dijkstra. As trê primeiras linhas tomam tempo $O(|V|)$; cada Relaxação(u, t) toma tempo constante. A complexidade de tempo do laço do algoritmo de Dijkstra depende

de como é determinado o mínimo na linha 5. O laço da linha 4 será executado uma vez para cada $u \in V(G)$; a contribuição do laço da linha 4 para a complexidade do algoritmo é

$$\sum_{u \in V} T_5(u) + T_6(u) + T_{7,8}(u)$$

onde $T_i(u)$ é o tempo gasto no linha i para u dado na linha 3. Se o mínimo é determinado fazendo comparações então em cada rodada o custo é $|\overline{S}| - 1$, portanto,

$$\sum_{u \in V} T_5(u) = \sum_{i=1}^{|V|-1} (i-1) = O(|V|^2).$$

Ademais $\sum_{u \in V} T_6(u) = O(|V|)$ e

$$\sum_{u \in V} T_{7,8}(u) = \sum_{u \in V} \sum_{v \in N(u)} O(1) = O(|E|).$$

Logo $\sum_{u \in V} T_5(u) + T_6(u) + T_{7,8}(u) = O(|V|^2 + |E|)$ que é a complexidade de tempo do algoritmo de Dijkstra levando em conta a complexidade da inicialização.

Outra possibilidade é usar fila de prioridades para determinar o mínimo. Nesse caso, podemos provar que o algoritmo de Dijkstra sobre $G = (V, E)$ usando uma fila de prioridades implementada por uma heap binária tem complexidade $O\big((|V| + |E|) \log |V|\big)$ (veja apêndice A.2.1, página 98, para os detalhes).

Exercícios

Exercício 93. Construa um grafo com pesos que podem ser negativos e no qual algoritmo de Dijkstra devolve resposta errada.

Exercício 94. Considere um grafo D com pesos $\rho\colon E(D) \to [0,1]$ nas arestas que representam a probabilidade de uma aresta não falhar. Escreva um algoritmo para descobrir o caminho mais seguro entre dois vértices.

Exercício 95. Seja $D = (V, E, \rho)$ um grafo com peso $\rho\colon E \to \{0, 1, \ldots, m\}$ nas arestas, $m \in \mathbb{N}$ fixo. Descreva uma fila de prioridades para o algoritmo de Dijkstra de modo que o complexidade para computar as distâncias seja $O(m|V| + |E|)$.

Exercício 96. Dados um grafo G com pesos positivos nas arestas e um vértice s considere um algoritmo que inicia com $d(v) \leftarrow \infty$ para todo vértice v e $d(s) \leftarrow 0$; em seguida o algoritmo repete $|V|$ vezes a seguinte instrução

para cada $\{u, t\} \in E(G)$ **faça** Relaxação(u, t).

Esse algoritmo determina dist(s, v) para todo v corretamente?

Exercício 97. O seguinte algoritmo recebe um grafo com pesos nas arestas e um par de vértices w, v desse grafo e devolve uma sequencia de vértices $v_1, v_2, \ldots, v_k$, com $v_1 = w$ e $v_k = v$.

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)) ;
```

Algoritmo 7: Imprime_caminho_mínimo(G, w, v).

onde Dijkstra'(G, w), na linha 1, é a seguinte variante do algoritmo de Dijkstra

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  para cada v ∈ V(G) faça
2  |  d(v) ← ∞;
3  |  p(v) ← nil;
4  d(s) ← 0;
5  S ← ∅;
6  enquanto S̄ ≠ ∅ faça
7  |  seja u tal que d(u) = min{d(v): v ∈ S̄} ;
8  |  para cada t ∈ N(u) faça
9  |  |  se d(t) > d(u) + ρ({u,t}) então
10 |  |  |  d(t) ← d(u) + ρ({u,t});
11 |  |  |  p(t) ← u;
12 devolva d e p.
```

Algoritmo 8: Dijkstra'(G, s).

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

2.4.2 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 cada $v \in V$.

Suponha que G seja representado pela matriz

$$a_{i,j} = \begin{cases} 0 & \text{se } i = j \\ \rho(\{i, j\}) & \text{se } \{i, j\} \in E \\ \infty & \text{caso contrário.} \end{cases} \tag{2.19}$$

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 $\text{dist}_0(i, j) = a_{i,j}$.

Por exemplo, o diagrama da figura 2.8 ao lado 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.

Também, temos as distâncias $d_0(3,4) = 5$, $d_1(3,4) = 4$ e $d_2(3,4) = d_3(3,4) = d_4(3,4) = 3$, assim como, $d_0(1,2) = d_1(1,2) = d_2(1,2) = 10$, $d_3(1,2) = 4$ e $d_4(1,2) = 3$.

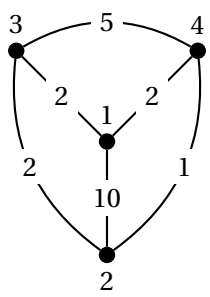


Figura 2.8: grafo com peso nas arestas.

Notemos que $d_0(i,j)$ é dado por $a_{i,j}$ definido em (2.19).

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,

$$\text{dist}_{k+1}(i,j) = \min\left\{\text{dist}_k(i,j), \text{dist}_k(i,k+1) + \text{dist}_k(k+1,j)\right\}. \tag{2.20}$$

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:

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** $(i,j) \in V^2$ **faça**
2 | $\text{dist}_0(i,j) \leftarrow a_{i,j}$, onde $a_{i,j}$ é como em (2.19);
3 **para cada** k *de* 1 *até* $|V|$ **faça**
4 | **para cada** $(i,j) \in V^2$ **faça**
5 | | $\text{dist}_k(i,j) \leftarrow \min\{\text{dist}_{k-1}(i,j), \text{dist}_{k-1}(i,k) + \text{dist}_{k-1}(k,j)\}$.

Algoritmo 9: Floyd–Warshall(G)

Análise do algoritmo de Floyd–Warshall.

A correção segue imediatamente da validade da equação (2.20).

Teorema 19. *A complexidade do algoritmo de* Floyd–Warshall *com entrada* $G = (V,E)$ *é* $O(|V|^3)$.

Exercícios

Exercício 98. Determine $\text{dist}_i(j,k)$ para toda terna $(i,j,k) \in V^3$ no grafo da figura 2.8.

Exercício 99. A seguinte versão do algoritmo de Floyd–Warshall funciona corretamente? Justifique.

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** $(i,j) \in V^2$ **faça**
2 | $\text{dist}(i,j) \leftarrow a_{i,j}$;
3 **para cada** k *de* 1 *até* $|V|$ **faça**
4 | **para cada** $(i,j) \in V^2$ **faça**
5 | | $\text{dist}(i,j) \leftarrow \min\{\text{dist}(i,j), \text{dist}(i,k) + \text{dist}(k,j)\}$.

Algoritmo 10: Floyd–Warshall(G)

Exercício 100. Escreva um algoritmo $O(|V|^3)$ que determina a cintura de um grafo $G = (V,E)$.

3 | Conexidade

Neste capítulo, estudamos a classe dos grafos conexos. Também, apresentamos uma medida de conexidade de grafos, abordamos um problema algorítmico e mostramos um exemplo de como construir grafos de alta conexidade e com o número mínimo de arestas possível. Grafos conexos e com o menor número possível de arestas são estudados na seção 3.2 onde também estudamos o problema da *árvore geradora de custo mínimo*: dado um grafo conexo com pesos nas arestas, determinar um subgrafo gerador conexo minimal e, ainda mais, de custo total mínimo. Tal problema é bastante conhecido e estudado há bastante tempo; apresentaremos dois algoritmos para o problema da árvore geradora de custo mínimo: o algoritmo de Kruskal, proposto em 1956 pelo matemático americano Joseph Kruskal, e o algoritmo de Prim, de 1957, devido ao também americano Robert Prim (o mesmo algoritmo foi inventado em 1930 pelo matemático tcheco Vojtěch Jarník e redescoberto por Dijkstra em 1959). O primeiro algoritmo para esse problema de que se tem notícia, de 1926, é o algoritmo de Otakar Borůvka, outro matemático tcheco que se ocupou de desenvolver uma rede eficiente para a companhia elétrica da Moldávia.

3.1 Grafos conexos

Um grafo G é conexo se é *não-vazio* e quaisquer dois vértices de G pertencem a um mesmo passeio em G, caso contrário dizemos que G é desconexo. A relação binária *alcança* definida sobre $V(G)$ dada no capítulo anterior é uma relação de equivalência, como já havíamos dito, e as classes de equivalência dessa relação são os vértices dos componentes conexos do grafo G. Uma definição equivalente de componente conexo é a seguinte. Dizemos que H é um *subgrafo conexo maximal* de G se para todo F tal que $H \subsetneq F \subseteq G$ implica que F é desconexo. Os componentes conexos do grafo são os subgrafos conexo maximais desse grafo. Por exemplo, no grafo do exemplo 1 temos quatro componentes conexos, a saber, induzidos pelos conjuntos de vértices $\{1,2,5\}$, $\{3,4\}$, $\{6,7\}$ e $\{8\}$.

Num grafo *desconexo* e não-vazio G temos um corte vazio entre subconjuntos não triviais de vértices. De fato, há pelo menos dois vértices, digamos u e w, que não estão num mesmo passeio em G, então os subconjuntos de vértices U e W dos vértices alcançáveis por u e por w, respectivamente, são disjuntos e tais que $E(U,W) = \varnothing$. A recíproca dessa afirmação vale no caso de cortes associados a subconjuntos não triviais de vértices.

Proposição 20. *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 admite um subconjunto próprio e não-vazio $U \subset V(G)$, então para qualquer $u \in U$ e qualquer $w \in \overline{U}$ existe em G um caminho $u = x_0, x_1, \ldots, x_k = w$. Definamos $m = \max\{j : x_j \in U\}$ e temos $0 \leq m < k$ pela escolha de u e de w, logo $\{x_m, x_{m+1}\} \in E(U,\overline{U})$. A recíproca está dada acima, no parágrafo anterior ao teorema. □

Se $G = (V,E)$ é um grafo e e uma aresta de G, então definimos o grafo obtido pela remoção de e por

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

Proposição 21. *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 e uma aresta de algum circuito $C \subseteq G$. Consideremos dois vértices quaisquer u, $v \in V(G-e)$ e P um caminho $u = u_1, u_2, \ldots, u_m = v$ em G com extremos u e v.

Se $e \notin E(P)$ então P é um caminho em $G - e$. Se $e \in E(P)$, então $e = \{u_i, u_{i+1}\}$ para algum i e a sequência $u_1, \ldots, u_{i-1}, C - e, u_{i+2}, \ldots, u_m$ é um passeio em $G - e$ que contém os vértices u e v. Logo $G - e$ é conexo. □

3.1.1 Articulações

Se $G = (V, E)$ é um grafo e v um vértice de G, então o grafo obtido pela remoção de v por é

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

e quando $G - v$ tem mais componentes conexos que G então v é dito articulação ou vértice de corte. Um grafo conexo sem articulações é chamado de biconexo. Os subgrafos biconexos maximais de um grafo G qualquer são chamados de componentes biconexos de G.

Exemplo 11. No grafo do diagrama na figura 3.1 são articulações os vértices $1, 3, 7, 9$. Os componentes biconexos

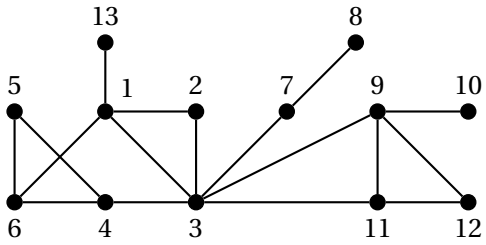


Figura 3.1: $1, 3, 7, 9$ são articulações.

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

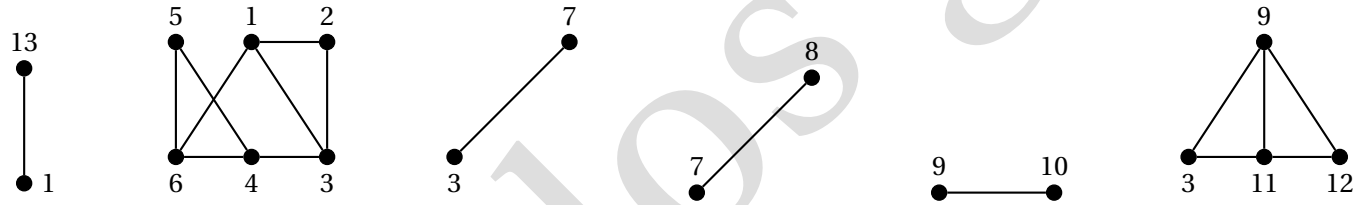


Figura 3.2: os seis componentes biconexos do grafo exibido na figura 3.1.

Na discussão que segue tratamos do seguinte problema computacional: dado G conexo, determinar as articulações de G. Notemos que como o grafo é conexo, numa execução de $\mathrm{BP}(G, w)$ todos os vértices serão visitados. A solução fácil para esse problema é remover cada vértice e testar se o grafo resultante é desconexo, o que resulta num algoritmo de tempo $O(|V|(|V| + |E|))$ pois uma busca, cuja complexidade de tempo é $O(|V| + |E|)$, permite decidir se o grafo é desconexo.

Usaremos a versão rotulada da busca em profundidade, algoritmo 3 na página 32, como base para um algoritmo mais eficiente que determina as articulações. Baseados na equação (2.14), página 33, estabelecemos a seguinte nomenclatura. 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. Notemos que v é um ancestral dele mesmo, um ancestral próprio do vértice v é um vértice ancestral diferente de v. Vamos chamar de

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

Tabela 3.1: do exercício 87 podemos estabelecer as equivalências acima e sabemos (proposição 15) que para quaisquer dois vértices distintos $u, v \in V(G)$ exatamente uma das possibilidades acima vale após a execução do algoritmo.

patriarca o vértice inicial de um percurso em profundidade, isso é, o único vértice $v \in V$ que tem $pai(v) = \texttt{nil}$.

Uma execução do algoritmo 3 sobre G classifica as arestas de $E(G)$ em dois tipos: as arestas pai-filho que são as arestas da forma $\{v, pai(v)\}$ e as outras arestas que são chamadas arestas de retorno. Pondo de outro modo, nas linhas 3 e 4 do algoritmo 3 testamos para cada $u \in N(w)$ se $chega(u) = 0$, ou seja se u não foi visitado, e se a condição é *falsa*, então dizemos que a aresta $\{w, u\}$ é uma aresta de retorno; nesse sentido, uma aresta de retorno foi visitada a partir de w e o outro extremo u é um ancestral de w mais velho que $pai(w)$.

A figura 3.3 a seguir ilustra as definições que acabamos de ver com uma busca em profundidade no grafo do exemplo 11 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.

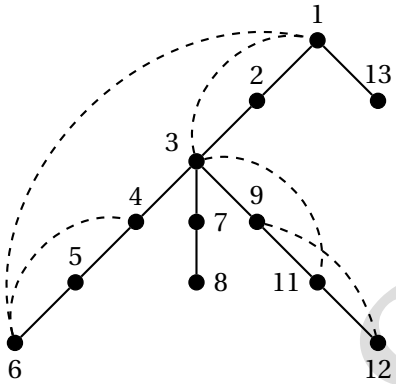


Figura 3.3: visão esquemática de uma busca em profundidade no grafo do exemplo 11. O vértice 1 é o *patriarca* da busca, as arestas tracejadas são arestas de *retorno*, 2 é *ancestral* de 11 pois $2 = pai^{(3)}(11) = pai\big(pai\big(pai(11)\big)\big)$ e, reciprocamente, 11 é *descendente* de 2.

Consideremos a execução de uma busca em profundidade rotulada $\mathrm{BP}(G, w)$ com G conexo. Vamos assumir também que G tem pelo menos três vértices; notemos que se o grafo é trivial então o vértice é articulação e se o grafo tem ordem 2 então é um P^2, logo não tem articulação.

Proposição 22. *Seja G um grafo conexo com pelo menos três vértices e consideremos a execução de uma busca em profundidade rotulada em G. 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 $chega(x) < chega(y)$ 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 3) e nesse momento x ainda está na pilha, pois a aresta $\{x, y\}$ ainda não foi visitada a partir de x. Portanto $chega(x) < chega(y) < sai(y) < sai(x)$ e pela tabela 3.1 y é descendente de x.

Se $chega(y) < chega(x)$ então deduzimos de maneira análoga que x é descendente de y. □

Segue dessa proposição que se v_1 e v_2 são filhos de u então $\{v_1, v_2\} \notin E(G)$. Mais que isso, não há descendente de v_1 adjacente a descendente de v_2.

Proposição 23. *Se v_1 e v_2 são filhos de u então não há descendente de v_1 (inclusive) adjacente a descendente de v_2 (inclusive), 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, de outro modo, $\big(chega(v_1), sai(v_1)\big) \cap \big(chega(v_2), sai(v_2)\big) = \varnothing$. Agora, se t é um descendente de v_1 então $chega(v_1) \leq chega(t) < sai(t) \leq sai(v_1)$. Se w é um descendente de v_2 então $chega(v_2) \leq chega(w) < sai(w) \leq sai(v_2)$; logo, pelo resultado anterior, concluímos que $\big(chega(t), sai(t)\big) \cap \big(chega(w), sai(w)\big) = \varnothing$. □

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 24. *Seja G um grafo conexo. 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 aresta de retorno dos descendentes de v tem como outro extremo um ancestral próprio de u.

Antes de provar o teorema, examinemos a representação gráfica da figura 3.3 acima. 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)).

Demonstração do Teorema 24. 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 $G - u$ tem pelo menos dois componentes conexos, assim u é articulação.

Suponha agora que u não é patriarca e seja v um filho de u com descendentes $\{v_1, v_2, \ldots, v_k\}$ ($k \geq 1$ pois u é descendente dele mesmo). Pela proposição 23 qualquer caminho com extremos v_i e $pai(u)$ ou tem u como vértice interno ou tem uma aresta de retorno $\{v_j, x\}$, para algum $j \in \{0, 1, \ldots, k\}$ e algum ancestral próprio x de u, que existe porque u não é patriarca. Se tais arestas de retorno não existem então todo caminho com extremos $pai(u)$ e um descendente de v passa por u assim em $G - u$ tem pelo menos dois componentes conexos, um que contém $pai(u)$ e outro que contém v.

Para provar a recíproca, vamos supor que u é uma articulação de um grafo G. Sejam x e y dois vértices de G em componentes conexos distintos em $G-u$, logo, todo caminho em G com extremos x e y são da forma x, P_1, u, P_2, y.

Consideremos uma execução de BP(G, w). A prova segue em 3 casos.

Suponhamos que x é descendente de y. Então o único caminho com extremos x, y e arestas pai-filho é da forma y, P_1, u, P_2, x com u descendente de y e ancestral de x. Se houver uma aresta de retorno entre um descendente próprio de u, digamos a, e um ancestral próprio de u, digamos b, então x, P, a, b, Q, y é um passeio em $G-u$, o que é uma contradição.

Suponhamos que y é descendente de x. Uma análise análoga a do parágrafo anterior mostra que se houver uma aresta de retorno entre um descendente próprio de u e um ancestral próprio de u, então derivamos uma contradição.

Resta o caso em que x não é descendente de y e y não é descendente 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 $chega(\,)$, o ancestral mais novo. 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 $x = z_0, z_1, z_2, \ldots, z_k, u, z_{k+1}, \ldots, z_t = y$ com todas as arestas pai-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. Portanto vale (2). □

Usaremos o teorema 24 e a busca em profundidade rotulada para escrever uma algoritmo que determina as articulações de um grafo conexo da seguinte maneira

1. Percorra o grafo em profundidade, recursivamente, a partir de um vértice w computando o tempo de $chega(w)$ para cada vértice;

2. se w é patriarca e tem mais de 2 filhos então w é articulação;

3. se w não é patriarca, então para cada filho u de w, determine a aresta de retorno $\{x, y\}$ entre um descendente x de u e um antecessor y de w tal que $chega(y)$ seja *mínimo*; se para todo filho u de w tal *mínimo* for menor que $chega(w)$ então w *não* é articulação.

Tal algoritmo pode ser descrito da seguinte forma

Dado : Grafo conexo G e um vértice w.
Devolve: as articulações de G.

```
1  cont ← cont + 1;
2  chega(w) ← cont;
3  min(w) ← chega(w);
4  para cada u ∈ N(w) faça
5      se chega(u) = 0 então
6          pai(u) ← w;
7          Articulação(G, u);
8          se pai(w) = nil então
9              se u é o segundo filho de w então escreva("w é articulação");
10         senão
11             min(w) ← mínimo{min(w), min(u)};
12             se min(u) ≥ chega(w) então escreva("w é articulação");
13     senão se pai(w) ≠ u e chega(u) < min(w) então
14         min(w) ← chega(u);
15 cont ← cont + 1;
16 sai(w) ← cont.
```

Algoritmo 11: Articulação(G, w)

Análise do algoritmo BP-Art.

O algoritmo é, essencialmente, uma busca em profundidade, portanto, sua complexidade de tempo é a mesma de tal busca.

Teorema 25. *A complexidade para determinar as articulações de* $G = (V, E)$ *é* $O(|V| + |E|)$. □

Para provar que o algoritmo funciona corretamente, precisamos provar que $min(\cdot)$ é calculado corretamente pois, se esse é o caso, então a corretude segue do teorema 24. Notemos que na linha 9 a resposta esta sempre correta pela condição (1) do teorema. Na linha 12 $min(u) \geq chega(w)$ significa que o vértice mais velho (menor valor de $chega$) que um descendente de u encontra via uma aresta de retorno é no máximo tão velho quanto w, portanto w é articulação e a resposta está correta, pela condição (2), desde que $min(u)$ tenha o valor correto: o valor de $chega$ do ancestral (próprio) mais velho adjacente a um descendente de u.

Consideremos um vértice w que não é pai de nenhum outro vértice de G após uma execução do algoritmo. Em cada rodada da linha 4 as arestas $\{u, w\}$ para as quais vale a condição na linha 13 são arestas de retorno, caso existam. A atribuição na linha 14 termina faz com que

$$min(w) = \text{mínimo}\{chega(u) : u \text{ é ancestral adjacente a } w\}.$$

A prova segue por indução, suponha que $min(u)$ é corretamente calculada nos filhos de w e vamos provar que $min(w)$ fica corretamente calculado ao término do laço da linha 4. Para cada aresta $\{w, u\}$, se u é um vértice novo, isto é, se a aresta é pai-filho, então Articulação(G, u) da linha 7 calcula corretamente o valor $min(u)$ de modo que

na linha 11 $min(w)$ será atualizado para o menor $chega$ de um ancestral de w adjacente a um descendente de u (caso exista); ao final de percorrer todos os filhos a linha 11 será responsável por armazenar

$$\text{mínimo}\{chega(w), min(u_1), \ldots, min(u_m) : u_i \text{ é filho de } w\}.$$

Agora, se u não é um vértice novo, isto é $\{u, w\}$ é uma aresta de retorno, então a linha 14 atualiza $min(w)$ caso tenha se encontrado um ancestral mais velho (menor $chega$) do que o mais velho até agora encontrado. Portanto, sempre que w não é patriarca $min(w)$ é o valor de $chega$ do ancestral mais velho adjacente a um descendente de w.

3.1.2 Grafos 2-conexo

Um grafo biconexo de ordem pelo menos 3 é dito 2-conexo. Apesar do aparente pedantismo dessa definição ela é bastante útil quando generalizamos essa definição para k-conexidade com $k \geq 3$, de modo que o caminho P^2 é o único grafo que é biconexo mas não é 2-conexo.

O seguinte resultado é um caso particular do famoso Teorema de Menger, cuja versão completa é dada adiante neste capítulo.

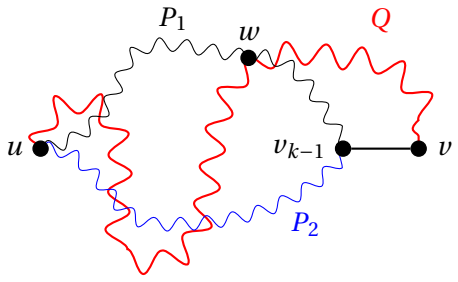
Teorema 26 (Teorema de Menger, 1927)**.** *Um grafo é 2-conexo se, e somente se, quaisquer dois vértices desse grafo pertencem a um circuito.*

Demonstração. A condição é suficiente pois se quaisquer vértices u e v estão num circuito então há sempre dois caminhos disjuntos nos vértices internos e com extremos u e v, portanto, a remoção de um vértice qualquer diferente deles nunca os deixam em componentes conexas distintas.

Seja G um grafo 2-conexo. Vamos provar por indução em k que se $\text{dist}(u, v) = k$ então u e v pertencem a um circuito de G. Se u e v são tais que $\text{dist}(u, v) = 1$; tomemos $e = \{u, v\}$. Notemos que $G - e$ é conexo (justifique) logo deve haver, em $G - e$, um caminho com extremos u e v. Esse caminho define com a aresta e um circuito que contém u e v.

Suponhamos, para a hipótese da indução, que todo par de vértices que distam $k > 1$ pertencem a um circuito de G. Consideremos u e v, vértices de G, que distam $k + 1$ e $u, v_1, \ldots, v_k, v$ $(k > 1)$ um caminho mínimo.

Pela hipótese da indução existe um circuito C que contém os vértices u e v_k de modo que podemos tomar P_1 e P_2 dois caminhos com extremos u e v_k e disjuntos nos vértices internos. Como G é 2-conexo, em $G - v_k$ existe um caminho Q com extremos u e v. Seja w o primeiro vértice de Q a encontrar P_1 ou P_2 quando Q é percorrido no sentido de v para u. Sem perda de generalidade, assumamos que w seja de P_1. Notemos que esse subcaminho de Q com extremos w e v é disjunto de P_2.



Então, o caminho formado pelo segmento de P_1 entre u e w seguido dos vértices de Q entre w e v é um caminho internamente disjunto a P_2. Tais caminhos definem um circuito que contém os vértices u e v. □

Grafos 2-conexo podem ser construídos da seguinte forma: começamos com um circuito C e um caminho P, disjuntos, em seguida identificamos os extremos de P com dois vértices distintos do circuito C; repetimos esse

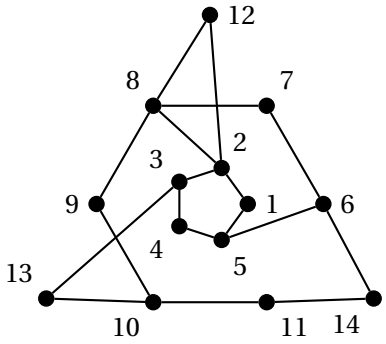


Figura 3.4: exemplo da construção de um grafo 2-conexo.

processo de colar caminhos no grafo resultante das etapas anteriores. No diagrama da figura 3.4, por exemplo, começamos com o circuito $1,2,3,4,5$, em seguida colamos o caminho $2,8,7,6,5$, em seguida $8,9,10,11,14,6$, depois $10,13,3$ e, finalmente, $8,12,2$.

Pelo Teorema de Menger acima o grafo obtido em cada passo é 2-conexo. Mais interessante é o fato dessa construção caracterizar os grafos 2-conexo, isto é, se um grafo é 2-conexo então ele pode ser obtido pela construção acima. De fato, seja G um grafo 2-conexo. Seja $G_0 \subset G$ um circuito qualquer e definiremos uma sequência $G_1, G_2, \ldots, G_i$ de subgrafos de G de modo que G_{j+1} é obtido de G_j pelo processo descrito acima.

Suponha que G_i é maximal com relação ao processo de construção. Se $G \neq G_i$ então temos uma aresta $e = \{u, v\} \in E(G) \setminus E(G_i)$ com $e \cap V(G_i) = \{v\}$, a qual existe pela conexidade de G. Se $u \in V_i$ então fazemos $G_{i+1} = G_i + e$, contrariando a maximalidade de G_i (portanto, G_i é induzido). Como G é 2-conexo temos que $G - v$ é conexo, portanto existe uma caminho P com extremo u e o outro extremo em V_i. Assim, v, u, P é um caminho que podemos colar em G_i contrariando sua maximalidade.

Teorema 27. *Um grafo é 2-conexo se, e só se, ele pode ser construído pelo processo descrito acima.* □

Exercícios

Exercício 101. Seja G um grafo. Uma ponte em G é uma aresta e tal que $G - e$ tem mais componentes conexos que G. Prove que se todos os vértices de G têm grau par então G não tem pontes.

Exercício 102. Escreva um algoritmo inspirado em 11 para determinar as pontes de um grafo conexo.

Exercício 103. Mostre que se $\Delta(G) \leq 2$ então os componentes conexos de G ou são caminhos ou são circuitos.

Exercício 104. Seja H um subgrafo gerador de G. Mostre que se H é conexo então G é conexo.

Exercício 105. 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 106. 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, $|u - v| \equiv k \mod n$.

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

Exercício 108. Prove que num grafo 2-conexo quaisquer duas arestas estão num mesmo circuito.

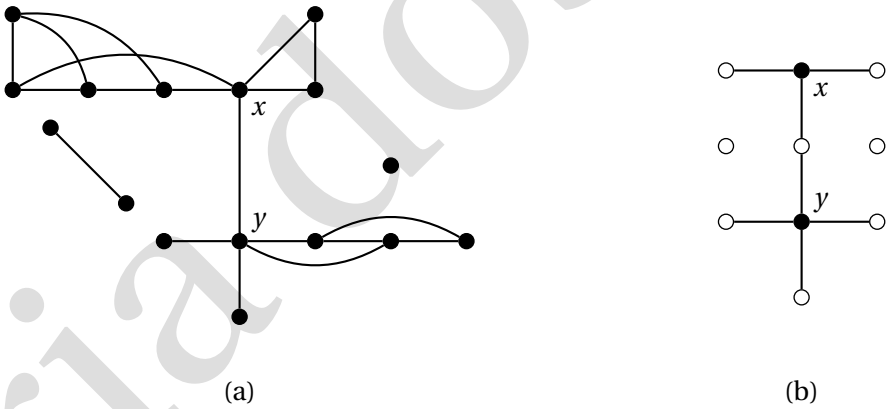
Exercício 109. Determine se a seguinte função, que é uma busca em profundidade modificada, identifica as articulações de um grafo conexo.

```
1  cont ← cont + 1;
2  chega(w) ← cont;
3  min ← cont;
4  para cada u ∈ N(w) faça
5  |  se chega(u) = 0 então
6  |  |  m ←BP-Art(G, 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.
```

Função BP-Art(G, w).

Exercício 110. Seja G um grafo. Lembremos que um vértice $v \in V(G)$ é dito *vértice de corte* ou *articulação* se $G - v$ tem mais componentes conexos que G e que uma aresta $e \in E(G)$ é uma *ponte* se $G - e$ tem mais componentes conexos que G. Dado um grafo G qualquer, não necessariamente conexo, um bloco do grafo G é um componente biconexo ou um vértice isolado no grafos

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 $\{v, b\}$, onde v corresponde a uma articulação de G e b um bloco de G, de forma que "$v \in b$". A figura 3.5 mostra um grafo e o grafo de blocos correspondente.



(a) (b)

Figura 3.5: (a) um grafo com vértices de corte x e y e (b) seu grafo de blocos.

Construa o grafo de blocos do grafo da figura 3.1.

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 111 (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 Árvores e Florestas

Chamamos de acíclico qualquer grafo que não contenha circuito. Uma floresta é um grafo acíclico e os componentes conexos de uma floresta são árvores, ou seja, uma árvore é um grafo conexo e acíclico.

Notemos que todo grafo acíclico G com pelo menos dois vértices tem pelo menos dois vértices de grau 1, basta tomarmos os extremos de um caminho de comprimento máximo (veja a figura 2.4, pág. 27). Chamamos de folha todo vértice de grau 1 numa árvore

Nos grafos conexos, a quantidade de arestas caracteriza árvores. Por um lado temos o seguinte resultado.

Proposição 28. *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. Para $n \geq 2$, 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 $n \geq 1$. □

Por outro lado, grafos conexos com uma aresta a menos que o número de vértices são árvores. Antes de provar a recíproca do lema, introduziremos um novo conceito: 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 28 do seguinte modo: 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 $T = G$, portanto G é uma árvore.

Teorema 29. *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 $M = \{\{v, pai(v)\} : v \in V(G) \text{ e } pai(v) \neq \text{nil}\}$, onde o vetor $pai(\,)$ é determinado pela execução do algoritmo 3(veja os exercícios 88 e 89).

Além do número de arestas, outras propriedades caracterizam árvores. Sejam G um grafo e $x, y \in V(G)$ vértices distintos de G. Definimos o grafo $G + xy$ por

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

Teorema 30. *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 como pode-se verificar facilmente. Se G tem ordem 2, então $G = K^2$ e a verificação também é fácil, deixa-mo-lá para o leitor. Vamos supor que G tem pelo menos 3 vértices e mostraremos (1)⇒(2)⇒(3)⇒(4)⇒(1).

(1) ⇒ (2): Seja G uma árvore. Como G é conexo, existe um caminho $P = x, x_1, x_2, \ldots, x_n, y$ para quaisquer vértices x e y. Suponha que exista outro caminho $Q = x, y_1, y_2, \ldots, y_m, y$, com $Q \neq P$. Definamos, para facilitar a escrita, $x_0 = y_0 = x$, $x_{n+1} = y_{m+1} = y$ e definamos os índices

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

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.

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

(3) ⇒ (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 21), 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 $x, v_1, \ldots, v_k, y, x$.

(4) ⇒ (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. □

Como consequência do teorema 30 temos o seguinte corolário.

Corolário 31. *Uma árvore com n vértices é um grafo conexo com o menor número possível de arestas dentre todos os grafos conexos com n vértices.* □

Exercícios

Exercício 112. Determine o número de arestas de uma floresta com n vértices e com k componentes conexos, em função de n e k.

Exercício 113. 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 114. Mostre que toda árvore T tem pelo menos $\Delta(T)$ folhas.

Exercício 115. Prove que o conjunto de arestas $\{pai(v), v\}$ definido por uma busca em profundidade rotulada induz uma árvore.

Exercício 116. 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 117. 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 118. Seja T uma árvore, u e v dois vértices não adjacentes em T e $a \in E(T)$ uma aresta do circuito em $T + uv$. Mostre que

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

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

Exercício 119. 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 sequência de operações elementares.

Exercício 120. Prove que se uma aresta é ponte num grafo G então ela pertence a todas as árvores geradoras de G.

Exercício 121. 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 122. Mostre que se G é um grafo conexo então o grafo dos blocos de G definido no exercício 110 é uma árvore.

Exercício 123. 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 de $E(G) \setminus E(T)$; para cada tal aresta, o grafo $T + e_i$ tem 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. Tomemos o conjunto $\{0,1\}$ dos restos módulo 2 com as operações de soma e multiplicação módulo 2. É sabido que essa estrutura é um corpo e definiremos, sobre esse corpo, um espaço vetorial pondo

$$0 \cdot C_i = \varnothing \qquad \text{e} \qquad 1 \cdot C_i = E(C_i).$$

Prove que para todo circuito $C \subset G$ existe uma sequência $(b_1, b_2, \ldots, b_{m-n+1}) \in \{0,1\}^{m-n+1}$ tal que $E(C)$ é dado por

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

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

Exercício 124 (*Shannon switching game*). O seguinte jogo foi proposto pelo matemático Claude Shannon para ser jogado por dois jogadores num grafo conhecido como grade retangular; a descrição do jogo num grafo qualquer é a seguinte.

O jogo é jogado em um grafo G com dois vértices especiais, S e C, por dois jogadores, o Positivo e o Negativo, que jogam alternadamente. Cada aresta do grafo pode ser *aberta* ou *fechada*, uma vez aberta ou fechada ela fica assim até o fim do jogo. Na sua vez o Positivo abre uma aresta e, na sua vez, o Negativo fecha uma aresta. O objetivo do Positivo é criar um caminho de arestas abertas entre S e C e o objetivo do Negativo é evitar que tal caminho seja construído.

O jogo (G,S,C) é dito *jogo positivo* se o jogador Positivo tem uma estratégia vencedora, não importando quem começa o jogo. O jogo (G,S,C) é dito *jogo negativo* se o jogador Negativo tem uma estratégia vencedora, não importando quem começa o jogo. O jogo (G,S,C) é dito *jogo neutro* se o jogador vendedor depende de quem começa o jogo.

Prove que se G tem um subgrafo H que contém os vértices S e C e admite duas árvores geradoras disjuntas nas arestas, então o jogo (G,S,C) é positivo. É possível provar a recíporca, que é bem mais difícil (veja [19]). Caracterizar os jogos neutros e os jogos negativos está em aberto.

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

Dado um grafo conexo com pesos $\rho\colon E \to \mathbb{R}$ nas arestas, $G = (V, E, \rho)$, 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 Dijkstra–Jarník–Prim e Kruskal para o problema de determinar uma árvore geradora mínima; esses algoritmos são algoritmos gulosos, que é uma técnica de projeto de algoritmos para resolver problemas de otimização baseada na escolha que parece ser a melhor no momento (ótimo local) e termina com a solução ótima (ótimo global).

3.3.1 Algoritmo de Dijkstra–Jarník–Prim para árvore geradora mínima

O algoritmo de Dijkstra–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 no corte $E(U, \overline{U})$, até que $U = V$. No final, as arestas escolhidas induzem uma árvore geradora mínima.

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 S.
```

Algoritmo 12: Dijkstra–Jarník–Prim(G)

Notemos que esse algoritmo é uma ligeira modificação do algoritmo 4 (pela proposição 20 os laços desses dois algoritmos têm a mesma condição).

Análise e correção do algoritmo de Dijkstra–Jarník–Prim.

Como no correção do algoritmo de Dijkstra, usaremos a técnica do invariante do laço.

Teorema 32. *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. Pelas linhas 6 e 7 toda aresta de S tem seus extremos em U, logo 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 $S_k = S_{k-1} \cup \big\{\{u, w\}\big\}$. Vamos mostrar que $T_k = (U_k, S_k)$ é uma subárvore de alguma árvore geradora mínima de G. Que T_k é uma árvore deixamos para a verificação do leitor. Se $\{u, w\} \in E(T_{\min})$ então $T_{\min}$ contém $T_k = (U_k, S_k)$, logo podemos supor que $\{u, w\} \notin E(T_{\min})$.

Pelo teorema 30 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 Dijkstra–Jarník–Prim e passamos a estudar a complexidade desse algoritmo.

A complexidade de tempo do algoritmo de Dijkstra–Jarník–Prim depende da implementação do passo 5 do algoritmo. Se ordenamos as arestas o custo é $O(|E| \log |E|)$ (veja [7]) 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. Uma implementação usando *heap* binária do algoritmo de Dijkstra–Jarník–Prim tem complexidade $O(|E| \log |V|)$ (seção A.2.1).

3.3.2 Algoritmo de Kruskal para árvore geradora mínima

A ideia 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:

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

```
1 S ← ∅;
2 F ← fila das arestas em ordem crescente 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 S.
```

Algoritmo 13: Kruskal(G)

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 33. *O grafo* $(V(G), S)$ *construído 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 $E(T) = S$.

Vamos provar que $S \subseteq E(T)$. Suponhamos que não é o caso, isto é, $S \setminus E(T) \neq \varnothing$ e definamos $m = \min\{j : e_j \in S \setminus E(T)\}$, onde o mínimo é com relação a sequência $(e_1, e_2, \ldots, e_n)$ em que as arestas foram inseridas no conjunto S pelo algoritmo.

Pelo teorema 30 a árvore $T + e_m$ contém um circuito pois $e_m \notin E(T)$ e nesse circuito deve existir uma aresta f que não está em S porque $G[S]$ é acíclico. O conjunto de arestas

$$R = \begin{cases} \{f\} & \text{se } m = 1 \\ \{e_1, \ldots, e_{m-1}\} \cup \{f\} & \text{se } m > 1 \end{cases}$$

está em T, portanto, R não induz circuito, mas como f não foi escolhida pelo algoritmo de Kruskal tem-se $\rho(f) \geq \rho(e_m)$. Logo $c(T + e_m - f) \leq c(T)$ 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 $S \subseteq E(T)$.

Para concluir, temos que $E(T) \setminus S = \varnothing$ 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*. Essas estruturas mantém dinamicamente uma família de subconjuntos disjuntos com um elemento de cada subconjunto eleito como o *representante* do subconjunto e permitem as operações

$cria(x)$ 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).

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

```
S ← ∅;
F ← fila das arestas em ordem crescente de peso;
para cada v ∈ V(G) faça cria(v) ;
para cada {u, v} ∈ F faça
    se busca(u) ≠ busca(v) então
        insira {u, v} em S;
        união(u, v);
devolva S.
```

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*. Essa representação com as heurísticas mencionadas tem um excelente desempenho assintótico; nesse caso, o número de operações $cria$, $busca$ e $união$ no algoritmos de Kruskal é menor que $2(|E| + |V|)$ e resulta na complexidade de tempo $O(|E| \log |E|)$ para o algoritmo de Kruskal.

Para mais detalhes consulte o apêndice A.2.2.

Exercícios

Exercício 125. 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 126. 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 127. Seja G um grafo conexo com pesos na arestas. Prove que para qualquer $U \subsetneq V(G)$ não-vazio, uma aresta de menor custo em $E(U, \overline{U})$ tem que pertencer a toda árvore geradora de G.

Exercício 128. 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 129. 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 130. 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 131. 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 132. Suponha que todos os pesos das arestas de G são positivos. Mostre que qualquer subconjunto de arestas que induz um subgrafo gerador conexo e tem peso total mínimo é uma árvore. Dê um exemplo com pesos não-positivos onde a conclusão não vale.

Exercício 133. 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 $S \cup \{\{u, v\}\} \subset E(T)$ para alguma árvore geradora mínima T de 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 S.
```

Algoritmo 14: AGM(G)

1. 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.)

2. 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 $e \notin E(T)$.)

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

3.4 Conexidade de grafos

Se $G = (V, E)$ então denotamos por $G - F$ o grafo $(V, E \setminus F)$. Dizemos que o grafo G é k-aresta-conexo se

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

Exemplo 12. O grafo do diagrama abaixo é 0-aresta-conexo, 1-aresta-conexo, 2-aresta-conexo, mas não é 3-aresta-conexo.

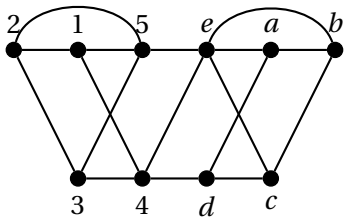


Figura 3.6: a remoção das arestas $\{5, e\}$, $\{4, e\}$ e $\{4, d\}$ resulta num grafo desconexo.

Segue da definição que todo grafo não-vazio e 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.

$$\lambda(G) = \max\{k \in \mathbb{N}\colon G \text{ é } k\text{-aresta-conexo}\}. \tag{3.3}$$

Como a remoção das arestas que incidem num vértice o tornam isolado temos que $\lambda(G) \leq \delta(G)$. No exemplo 12 vale $\lambda(G) = 2$ e $\delta(G) = 3$. Usando que $\lambda(G) \leq \delta(G)$ e o teorema 1, página 8, podemos deduzir que o número de arestas num grafo com pelo menos 2 vértices é

$$|E(G)| \geq \frac{1}{2}|V(G)|\lambda(G). \tag{3.4}$$

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-vértice-conexo ou, simplesmente, k-conexo se

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

O grafo do exemplo 12 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.

Claramente, todo grafo não-vazio é 0-conexo, todo grafo não-trivial conexo é 1-conexo e G^n é $n-1$-conexo se for o grafo completo. Além disso, segue imediatamente da definição que todo grafo k-conexo também é ℓ-conexo para todo natural $\ell < k$.

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

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

No exemplo 12 temos $\kappa(G) = 2$. Se G é desconexo então $\kappa(G) = 0$ e $\kappa(K^n) = n-1$ para todo $n > 1$. Segue da definição de conexidade que vale (verifique)

$$|V(G)| = \kappa(G) + 1 \text{ ou há } U \subseteq V(G) \text{ com } |U| = \kappa(G) \text{ e } G - U \text{ desconexo}$$

ou seja, $\kappa(G) = |V(G)| - 1$ (caso G seja completo) ou $\kappa(G)$ é o tamanho do menor subconjunto de $V(G)$ cuja remoção separa dois vértices de G.

Teorema 34 (Whitney, 1932)**.** *Para todo G com pelo menos 2 vértices*

$$\kappa(G) \leq \lambda(G). \tag{3.6}$$

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:

$$\textit{se } \lambda(G) = k \textit{ então } \kappa(G) \leq k. \tag{3.7}$$

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 $\kappa(G) \leq k - 1$. Seja G um grafo qualquer com $\lambda(G) = k$ e vamos mostrar 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, se no grafo $G - e$ houver um subconjunto L de arestas de cardinalidade $k - 2$ tal que $(G - e) - L$ seja desconexo, então G teria um subconjunto de arestas de cardinalidade menor que k, a saber $L \cup \{e\}$ cuja remoção o tornaria desconexo, portanto $\lambda(G - e) = k - 1$. Pela hipótese indutiva temos que $\kappa(G - e) \leq k - 1$.

Pela definição de conexidade ou $|V(G - e)| = \kappa(G - e) + 1$ ou existe $U \subset V(G - e)$ com $\kappa(G - e)$ vértices tal que $(G - e) - U$ é desconexo. No primeiro caso temos

$$|V(G)| = |V(G - e)| = \kappa(G - e) + 1 \leq k,$$

portanto $\kappa(G) < k$, ou seja, a sentença (3.7) acima vale.

No segundo caso $|V(G)| \geq \kappa(G - e) + 2$ e vale que

(i) ou $G - U$ é desconexo ,

(ii) ou $G - U$ é conexo e a remoção de $\{e\}$ de $G - U$ torna-o desconexo.

No caso (i) temos $\kappa(G) \leq |U|$ e $|U| = \kappa(G - e) \leq k - 1$.

No caso (ii), se $e = \{x, y\}$ então $(G - U) - x$ é desconexo a menos que $G - U = K^2$, entretanto, se tal fato ocorre temos

$$\kappa(G) \leq |V(G)| - 1 = |U| + 1 = \kappa(G - e) + 1 \leq k$$

e se $(G - U) - x$ é desconexo então $\kappa(G) \leq |U| + 1 \leq k$ o que prova (3.7). Pelo Princípio do Indução Finita a sentença (3.7) vale para todo $k \in \mathbb{N}$ e isso prova o teorema. □

Usando (3.4), o número mínimo de arestas num grafo k-aresta-conexo é

$$|E(G)| \geq \frac{k}{2}|V(G)|. \tag{3.8}$$

Da equação 3.4 e do teorema que acabamos de provar tiramos que para todo G com pelo menos dois vértices

$$|E(G)| \geq \frac{1}{2}|V(G)|\kappa(G). \tag{3.9}$$

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

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

Seja G um grafo, $A,\ B,\ S \subseteq V(G)$. Chamamos um caminho P em G de AB-caminho se P tem um extremo em A e o outro em B. Notemos que se $v \in A \cap B$ então o vértice é um AB caminho trivial. Dizemos que S separa A e B se em $G - S$ não há AB-caminho.

No resultado a seguir usaremos uma operação em grafo chamada contração de aresta: se $e = \{x, y\} \in E(G)$, a contração de e resulta num grafo sem a aresta e e com os vértices x e y identificados

$$G/e = \Big(V(G) \setminus \{x,y\} \cup \{v_e\}, E(G) \setminus (E_G(x) \cup E_G(y)) \cup \big\{\{v_e, w\} \colon w \in N_G(x) \cup N_G(y),\ w \neq x,\ y\big\}\Big) \tag{3.10}$$

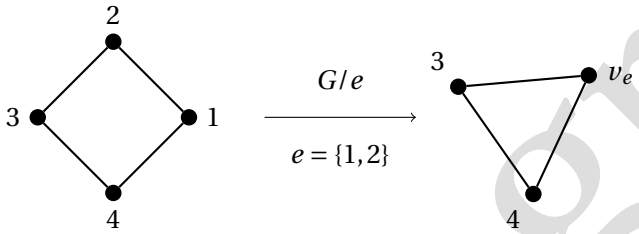


Figura 3.7: exemplo de contração de aresta.

Convencionamos que quando contraímos uma aresta que incide num vértice de $A \subset V(G)$, o vértice resultante da contração está em A.

Teorema 35 (Teorema de Menger)**.** *Sejam G um grafo e A e B subconjuntos de vértices. A cardinalidade do menor conjunto que separa A e B é igual ao número máximo de caminhos disjuntos entre A e B em G.*

Demonstração. Seja k a menor cardinalidade de um conjunto que separa A e B. Como todos os caminhos entre A e B passam por tal conjunto, não deve haver mais que k desses caminhos, logo precisamos mostrar que existem k caminhos disjuntos entre A e B. A prova é por contradição.

Suponha que G é um contraexemplo à afirmação do teorema com o menor número de arestas dentre todos os contraexemplos. Notemos que $|E(G)| > 0$ pois, caso contrário, se $|E(G)| = 0$ então a hipótese de haver conjunto que separa implica em $|A \cap B| = k$ e assim temos os k caminhos triviais $A \cap B$ entre A e B.

Seja $e = \{x, y\}$ uma aresta de G. Se G não tem k caminhos disjuntos entre A e B então também não tem G/e. Pela hipótese de minimalidade G/e tem um conjunto S que separa A e B com menos que k vértices, entre eles o vértice resultado da contração v_e, senão S separaria A e B em G. Desse modo $S' = (S \setminus \{v_e\}) \cup \{x, y\}$ separa A e B com k vértices em G.

Pra finalizar, consideremos $G - e$. Um conjunto que separa A e S' em $G - e$, bem como um que separa S' e B, também separa A e B em G, portanto tem pelo menos k vértices. Pela minimalidade assumida, existem k AS' caminhos disjuntos e k $S'B$ caminhos disjuntos em $G - e$, tais caminhos não se encontram fora de S', portanto podem ser combinados de modo a formarem k AB-caminhos disjuntos. □

Algoritmos eficientes para determinar a conexidade de um grafo são baseados no teorema de Menger (veja exercício 143 para uma formulação equivalente) e técnicas de fluxo em redes (apresentada na seção 6.4).

Exercícios

Exercício 134. Determine $\kappa(G)$ e $\lambda(G)$ nos casos P^k, C^k, K^k, k-cubo (definido no exercício 82), e $K^{m,n}$.

Exercício 135. Dê uma cota inferior para o grau médio de um grafo k-conexo com n vértices.

Exercício 136. 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 137. Prove que se G é 3-regular então $\kappa(G) = \lambda(G)$.

Exercício 138. 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 139. Prove que todo grafo 2-conexo minimal contém um vértice de grau 2.

Exercício 140. Prove que se $\delta(G) \geq |V(G)| - 2$ então $\kappa(G) = \delta(G)$.

Exercício 141. Prove que se $\delta(G) \geq |V(G)|/2$ então $\lambda(G) = \delta(G)$.

Exercício 142. Prove ou dê um contraexemplo: 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 143 (Teorema de Menger, 1927). Prove que G é k-conexo ($k \geq 1$) se e somente se para quaisquer $u, v \in V(G)$ existem k caminhos disjuntos nas arestas com extremos u e v.

Exercício 144. Vale a versão do Teorema de Menger para k-aresta-conexidade?

3.4.1 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,

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

   (2.2) Se n é ímpar,

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

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 36. *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 sequências 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 subsequência $P = i_0, \ldots, i_t$ 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 sequência 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 $i+(\ell-1)r < i_\ell \leq i+\ell r$.

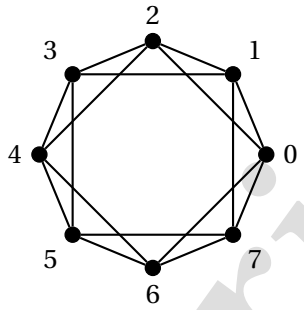
Tal sequência P é um caminho com extremos i e j, o que contradiz o fato deles estarem em componentes distintas.

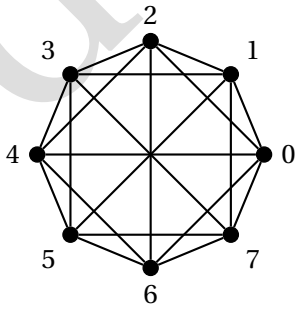
Agora, suponha $k = 2r+1$ com n par e que exista $U \subset V(H_{n,k})$ com $|U| \leq 2r$ e $H_{n,k} - U$ desconexo. Tome dois vértices i e j em componentes distintas de $H_{n,k} - U$ e considere as sequências de vértices S e T como acima. Se $|U \cap S| < r$ ou $|U \cap T| < r$ então temos um caminho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

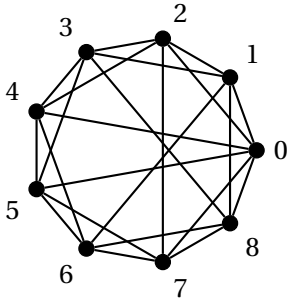
Pra não haver uma sequência como a P definida acima, é preciso que os r vértices de U em S sejam vértices consecutivos $i+s+1, i+s+2, \ldots, i+s+r$. Analogamente, os r vértices de U em T devem ser consecutivos, digamos $j+t+1, j+t+2, \ldots, j+t+r$. Nesse caso, i é um vértice do caminho $Q = j+t+r+1, j+t+r+2, \ldots, i+s$ 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 13. Abaixo temos, respectivamente, os diagramas dos grafos $H_{8,4}$, $H_{8,5}$ e $H_{9,5}$.







4 | Grafos eulerianos e hamiltonianos

4.1 Trilhas e grafos eulerianos

Em 1796 Euler resolveu o problema conhecido como problema das *sete pontes de Königsberg*, o que hoje é considerado como o primeiro resultado publicado na Teoria dos Grafos. O problema é decidir se é possível atravessar cada uma das sete pontes da cidade uma única vez e retornar ao ponto de partida (veja a figura na página 3). Esse problema é modelado pelo grafo da figura 4.1 e queremos saber se existe um passeio fechado onde toda aresta ocorre exatamente uma vez. Grafos que admitem tal passeio são ditos eulerianos.

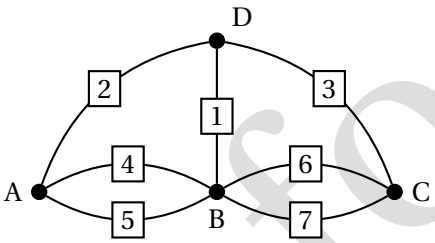


Figura 4.1: as 7 pontes de Königsberg.

Nessa seção admitiremos que os grafos tenham laços e arestas múltiplas, a que chamamos multigrafo. Formalmente, um multigrafo é dado por uma terna (V, E, ϕ) de modo que V e E são conjuntos disjuntos, respectivamente vértices e arestas do multigrafo, e ϕ é uma função que associa a cada aresta um par de vértices, não necessariamente distintos. No grafo da figura 4.1 temos $V = \{A, B, C, D\}$, $E = \{1, 2, 3, 4, 5, 6, 7\}$ e ϕ é dada por

$$\phi(1) = \{B, D\}, \quad \phi(2) = \{A, D\}, \quad \phi(3) = \{C, D\}$$
$$\phi(4) = \{A, B\}, \quad \phi(5) = \{A, B\}, \quad \phi(6) = \{B, C\} \quad \text{e} \quad \phi(7) = \{B, C\}.$$

Dizemos que num multigrafo $G = (V, E, \phi)$ uma sequência alternada vértice–aresta–vértice que não repete arestas

$$v_0, e_1, v_1, \ldots, v_{k-1}, e_k, v_k, \tag{4.1}$$

com $\phi(e_i) = \{v_{i-1}, v_i\}$, para todo $i \in \{1, 2, \ldots, k\}$, e $e_i \neq e_j$ para todo $i \neq j$, é chamada de trilha. Se em (4.1) 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. Por exemplo, no grafo do diagrama na figura 4.2 abaixo $1, a, 2, b, 3, c, 4, d, 2$ é uma trilha, assim como $4, d, 2, b, 3, c, 4, e, 5, f, 3$ é trilha.

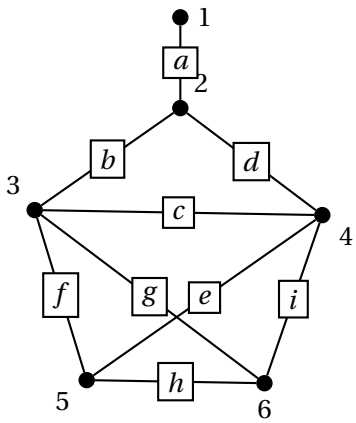


Figura 4.2: exemplos de trilhas.

Observemos que num grafo a sequência dada em (4.1) pode ser representado por $v_0, v_1, \ldots, v_{k-1}, v_k$ com $\{v_{i-1}, v_i\} \in E$. No entanto, nesta seção permitimos multigrafos e, por isso, v_i, e_i, v_{i+1} é diferente de v_i, f_i, v_{i+1} para e_i e f_i arestas distintas com extremos v_i e v_{i+1}.

Teorema 37 (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 $T = v_0, e_1, v_1, \ldots, v_{\ell-1}, e_\ell, v_\ell$ uma trilha com o maior número possível de arestas, com isso a trilha contém todas as arestas cujos extremos são os vértices v_0 e v_ℓ. Como todos os vértices têm grau par $v_0 = v_\ell$ (de fato, se forem diferentes e v_ℓ aparece k vezes na sequência T então $d(v_\ell) = 2(k-1)+1$.

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 $u, e, v_i, e_{i+1}, \ldots, e_\ell, v_\ell, e_1, v_1, \ldots, e_{i-1}v_i$ é uma trilha com mais arestas que T. Portanto, T passa por todas as arestas do grafo. □

Exercícios

Exercício 145. Prove que se G é conexo e tem no máximo dois vértices de grau ímpar então G contem uma trilha euleriana.

Exercício 146. Prove que se G é euleriano então LG (veja a definição no exercício 18) é euleriano.

Exercício 147. Prove que um grafo conexo é euleriano se pode ser particionado em circuitos aresta-disjuntos.

Exercício 148. Um grafo G é dito *aleatoriamente euleriano a partir do vértice x* se qualquer trilha maximal a partir de x é uma trilha euleriana fechada (uma trilha é maximal se, ao remover de G as arestas da trilha, os vértices pelo qual a trilha passa ficam isolados). Prove que G não vazio é aleatoriamente euleriano a partir do vértice x se G tem uma trilha euleriana fechada e x pertence a todo circuito de G.

Exercício 149. Seja T uma árvore. Acrescente a T um novo vértice x e acrescente as arestas entre x e todo vértice de grau ímpar de T. Prove que o grafo obtido é aleatoriamente euleriano a partir do vértice x.

Exercício 150. Um grafo $G = (V, E)$ euleriano pode ter uma ponte?

Exercício 151. O algoritmo de Fleury, de 1883, descobre uma trilha euleriana fechada se o grafo dado for euleriano.

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, que não seja ponte ;
6 |   u ← w;
7 |   G ← G − e.
8 devolva S.
```

Algoritmo 15: Fleury(G)

Prove que a sequência $S = (u_1, u_2, \ldots, u_\ell)$ construída pelo algoritmo é define uma trilha euleriana fechada. Determine a complexidade do algoritmo.

4.2 Grafos hamiltonianos

O matemático Sir William Rowan Hamilton inventou um jogo conhecido como quebra-cabeça de Hamilton que envolve encontrar um circuito hamiltoniano no grafo borda do dodecaedro (figura 4.3 ao lado). Um circuito C em um grafo conexo G é um circuito hamiltoniano em G se $V(C) = V(G)$. Um grafo que contém circuito hamiltoniano é chamado grafo hamiltoniano. Um caminho hamiltoniano em G é um caminho que passa por todos os vértices de G. O dodecaedro é um exemplo de grafo hamiltoniano e o grafo de Petersen é um exemplo de grafo que não é hamiltoniano, porém o grafo de Petersen contém um caminho hamiltoniano (verifique).

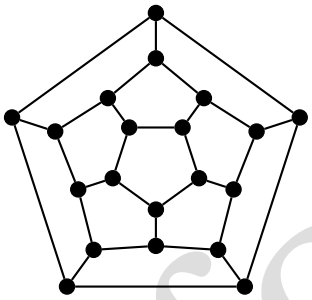


Figura 4.3: grafo dodecaedro.

Ao contrário dos grafos eulerianos, os 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.

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

Proposição 38. *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 $C \subseteq G$. □

Um contraexemplo para a recíproca do resultado acima é o grafo de Petersen pois pode-se verificar (por força bruta) 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 $|S|$, embora, como já foi dito, tal grafo não é hamiltoniano.

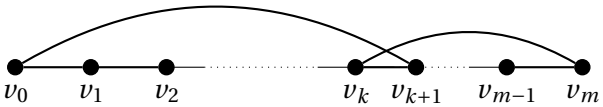
Uma condição suficiente para um grafo ser hamiltoniano é dada no seguinte resultado.

Teorema 39 (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 G um grafo como no enunciado. Tomemos $P = v_0, \ldots, v_m$ o maior caminho em G e defina os conjuntos

$$A = \{v_i \in V(P) : \{v_0, v_{i+1}\} \in E\} \text{ e } B = \{v_j \in V(P) : \{v_0, v_j\} \in E\}. \tag{4.2}$$

Observamos que A e B são subconjuntos de $\{v_0, v_1, \ldots v_{m-1}\}$ com pelos 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 $u \in V(G) \setminus V(C)$ 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 contraexemplo para a recíproca do resultado anterior basta considerar qualquer circuito com pelo menos 5 vértices, que é hamiltoniano e 2-regular.

O método probabilístico: Um torneio com n vértices é obtido quando orientamos cada aresta de um grafo completo com n vértices. Szele em 1943 provou que existe torneio com pelo menos $n!2^{1-n}$ caminhos hamiltonianos dirigidos e sua prova é considerada a primeira aplicação do método probabilístico em Combinatória.

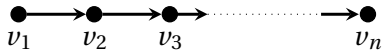


Figura 4.4: caminho dirigido.

Comecemos com o grafo completo de ordem n e em cada aresta sorteamos uma das orientações com probabilidade $1/2$ de ocorrer cada uma. Para cada permutação σ de $V(K^n)$ seja X_σ a variável aleatória que vale 1 se σ corresponde a um caminho hamiltoniano dirigido e vale 0 caso contrário. O valor esperado dessa variável aleatória é

$$0 \cdot \mathbb{P}\big(X_\sigma = 0\big) + 1 \cdot \mathbb{P}\big(X_\sigma = 1\big) = \left(\frac{1}{2}\right)^{n-1}$$

e a soma sobre todas as permutações $\sum_\sigma X_\sigma$ é o número de caminhos hamiltonianos do torneio, cujo valor esperado é

$$\sum_\sigma \left(\frac{1}{2}\right)^{n-1} = n! \left(\frac{1}{2}\right)^{n-1}$$

portanto, deve existir um modo de orientar as arestas do grafo completo de ordem n de modo que tenha pelo menos $n!2^{1-n}$ caminhos hamiltonianos dirigidos.

O problema do caxeiro viajante: Suponha que um caixeiro viajante tenha de visitar n cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. Suponha, também, que não importa a ordem com que as cidades são visitadas. O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total.

Esse problema pode ser modelado como um grafo com peso positivo nas arestas de modo que as cidades são vértices, as estradas entre cidades as arestas do grafo e a distância entre duas cidades é o comprimento da aresta. Podemos considerar o grafo sendo completo, se não existir um caminho entre duas cidades, acrescentamos uma aresta arbitrariamente longa sem afetar o resultado ótimo que é o circuito hamiltoniano de menor custo.

Exercícios

Exercício 152. Prove que se G não é 2-conexo então G não é hamiltoniano.

Exercício 153. Prove que se G é bipartido com bipartição $V(G) = A \cup B$ onde $|A| \neq |B|$, então G não é hamiltoniano.

Exercício 154. Prove que o n-cubo é hamiltoniano para todo $n \geq 2$.

Exercício 155. Prove 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 156. 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 157. Prove que se $d(u) + d(v) \geq |V(G)|$ para todo par u, v de vértices não adjacentes, então G é hamiltoniano se e somente se $G + uv$ é hamiltoniano.

Exercício 158. Prove que se G é um grafo euleriano então LG é hamiltoniano. Dê um exemplo onde a recíproca não vale.

Exercício 159. Dê um exemplo de grafo com n vértices, grau mínimo $\lfloor n/2 \rfloor$ e não-hamiltoniano, para todo $n \geq 3$.

Exercício 160. Verifique que o grafo de Petersen não é hamiltoniano.

5 | Emparelhamento

Emparelhamento é um dos tópicos mais estudados da Teoria dos Grafos devido a ampla variedade de aplicações; uma referência que aborda profundamente esse tema é [20]. Os primeiros estudos conhecidos são de G. Monge de 1784 e o caso de grafos bipartidos teve seus fundamentos estabelecidos por Frobenius e por Kőnig, por volta de 1915. 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 bipartidos.

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. Quando um vértice $v \in V(G)$ pertence a alguma aresta $e \in M$, dizemos que v é saturado por M e, também, que M satura v. Dessa forma, pela definição de emparelhamento, quando v é saturado por M existe uma única aresta e que incide no vértice v e pertence ao emparelhamento, ou seja, $E(v) \cap M = \{e\}$. Se todo elemento de $V(G)$ é saturado por alguma aresta de M, então M é chamado de emparelhamento perfeito em G.

No grafo dado pelo diagrama da figura 5.1 ao lado o conjunto $M_a = \{\{2,4\},\{5,6\}\}$ é um emparelhamento; note que não há arestas do emparelhamento que incidem no vértice 1 nem no vértice 3. Também é um emparelhamento no mesmo grafo $M_v = \{\{1,2\},\{3,6\},\{4,5\}\}$ cujas arestas, ao contrário de M, incidem em todos os vértices do grafo. O emparelhamento M_a não é um emparelhamento perfeito pois não satura os vértices 1 e 3, enquanto que o emparelhamento M_v é um emparelhamento perfeito. Notemos que emparelhamento perfeito não é único, nesse exemplo da figura 5.1 há um outro emparelhamento perfeito (verifique).

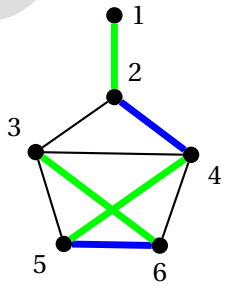


Figura 5.1: emparelhamentos.

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 admitem emparelhamento perfeito.

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

$$\mu(G) = \max\left\{|M| \colon M \text{ é emparelhamento em } G\right\} \tag{5.1}$$

arestas. 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 e um dos extremos não é saturado pelo emparelhamento. Dizemos que $P = x_1, x_2, \ldots, x_k$, para $k \geq 1$, caminho é M-alternante em G se

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

para todo $i \in \{1,2,\ldots,k-1\}$ e um dos extremos não é saturado por M. Ademais, se os dois extremos de um caminho M-alternante *não* são saturados 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. Por exemplo, na figura 5.1 temos $M = \{\{2,4\},\{5,6\}\}$ (o emparelhamento azul) e o caminho M-aumentante $P = 1,2,4,5,6,3$ com $E(P) = \{\{1,2\},\{2,4\},\{4,5\},\{5,6\},\{6,3\}\}$. A diferença simétrica $M \triangle E(P)$ desses conjuntos é

$$\begin{aligned}&\big(\{\{2,4\},\{5,6\}\} \cup \{\{1,2\},\{2,4\},\{4,5\},\{5,6\},\{6,3\}\}\big) \setminus \big(\{\{2,4\},\{5,6\}\} \cap \{\{1,2\},\{2,4\},\{4,5\},\{5,6\},\{6,3\}\}\big)\\ &\quad = \{\{1,2\},\{4,5\},\{6,3\}\}\end{aligned}$$

que é um emparelhamento com uma aresta a mais que M (que é o emparelhamento em verde no figura 5.1).

Proposição 40. *Sejam G um grafo, M um emparelhamento em G e P um caminho M-aumentante em G. 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 ser M-aumentante tiramos que $|E(P)| - |M \cap E(P)| = |M \cap E(P)| + 1$ e usamos essa igualdade para concluir

$$\begin{aligned}|M \triangle E(P)| &= |(M \cup E(P)) \setminus (M \cap E(P))|\\ &= \big|\big(M \setminus (M \cap E(P))\big) \cup \big(E(P) \setminus (M \cap E(P))\big)\big|\\ &= |M \setminus (M \cap E(P))| + |E(P) \setminus (M \cap E(P))|\\ &= |M| - |M \cap E(P)| + |E(P)| - |M \cap E(P)|\\ &= |M| - |M \cap E(P)| + |M \cap E(P)| + 1\\ &= |M| + 1.\end{aligned}$$

Resta provar que $M \triangle E(P)$ é emparelhamento. Suponha que não, então existem duas arestas distintas em $M \triangle E(P)$ adjacentes, digamos que $e, f \in M \triangle E(P)$ com $e \cap f = x$ e, sem perda de generalidade, $e \in M \setminus E(P)$ e $f \in E(P) \setminus M$. Do caminho ser aumentante temos que x deve ser um vértice interno em P logo x deve estar saturado por uma aresta $g \in M \cap E(P)$, logo $x \in e \cap g$ contrariando o fato de M ser um emparelhamento. □

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 41 (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 pela diferença simétrica dos dois emparelhamentos $H = G[M \triangle M^*]$. Como $\Delta(H) \leq 2$ os componentes conexos de H são circuitos e caminhos (exercício 103).

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 saturados por M. Esse caminho é M-aumentante em G. □

Exercícios

Exercício 161. Prove que uma árvore qualquer tem no máximo um emparelhamento perfeito.

Exercício 162. Considere um grafo G de n vértices. Mostre que um emparelhamento em G tem no máximo $n/2$ arestas.

Exercício 163. Mostre que o k-cubo admite emparelhamento perfeito, para todo $k \geq 1$.

Exercício 164. 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 165. Mostre que $\mu(G) = \alpha(\mathrm{L}G)$ (veja a definição de LG no exercício 18).

Exercício 166. Prove que se M é emparelhamento em G então existe um emparelhamento máximo que satura todos os vértices saturados por M.

Exercício 167 (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.

Exercício 168 (Teorema de Petersen, 1891). Use o exercício anterior para mostrar o seguinte resultado. Todo grafo 3-regular e sem ponte admite emparelhamento perfeito.

5.2 Emparelhamento e cobertura 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 não-saturado 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. Uma cobertura mínima é uma cobertura com o menor número possível de vértices

$$\nu(G) = \min\Big\{|U| \colon U \text{ é cobertura por vértices em } G\Big\}. \tag{5.2}$$

Notemos que para qualquer cobertura U e qualquer emparelhamento M vale que toda aresta de M tem que ter pelo menos um extremo em U, portanto $|M| \leq |U|$ e, em particular,

$$\mu(G) \leq \nu(G). \tag{5.3}$$

Por exemplo, em circuitos de tamanho n ímpar a cobertura mínima tem $(n+1)/2$ vértices enquanto que se n é par a cobertura mínima tem $n/2$ vértices. Dessa forma, a desigualdade em (5.3) é estrita em circuitos ímpares e vale a igualdade em circuitos pares. Na figura 5.2 ao lado temos o diagrama de um grafo bipartido, portanto sem circuito ímpar, e o subconjunto de vértices $\{a_1, a_2, b_3, b_4\}$ é uma cobertura com a mesma cardinalidade de um emparelhamento máximo, dado pelas arestas em destaque; o seguinte resultado garante que essa cobertura é mínima.

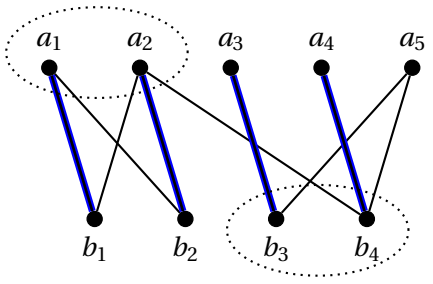


Figura 5.2: um emparelhamento e uma cobertura.

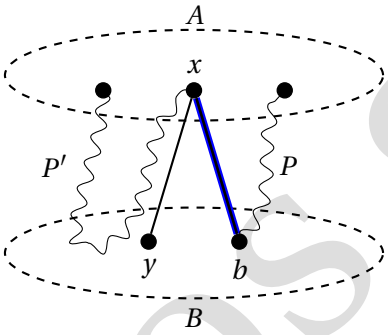
Teorema 42 (Teorema de Kőnig, 1931)**.** *Num grafo bipartido $G = (A \cup B, E)$ o tamanho de um emparelhamento máximo é igual ao tamanho de uma cobertura mínima, ou seja*

$$\mu(G) = \nu(G). \tag{5.4}$$

Demonstração. Pela equação (5.3) só precisamos provar $\nu(G) \leq \mu(G)$. Consideremos M um emparelhamento máximo em G e vamos construir uma cobertura $U \subseteq V(G)$ da seguinte maneira: para cada aresta $\{a, b\} \in M$, com $a \in A$ e $b \in B$, escolhemos b para U se b é extremo de algum caminho M-alternante, caso contrário, escolhemos a. Note que dessa forma temos $|U| = |M| = \mu(G)$ pois em U temos um vértice para cada aresta e porque M é máximo.

Vamos mostrar que U é uma cobertura. Considere uma aresta qualquer $\{x, y\} \in E(G)$, onde $x \in A$ e $y \in B$, e vamos mostrar que essa aresta encontra U. Se $\{x, y\} \in M$ então a aresta encontra U por definição de U.

Consideremos o caso $\{x, y\} \notin M$. Como M é máximo, ou x ou y ou ambos são saturados por M, caso contrário acrescentaríamos $\{x, y\}$ a M e obteríamos um emparelhamento maior. Se M não satura x então y pertence ao caminho M-alternante $P = x, y$, logo $y \in U$. Resta-nos verificar o caso $\{x, y\} \notin M$ com x saturado por M por causa de uma aresta $\{x, b\} \in M$ para algum $b \in B$, $b \neq y$.



Se $x \in U$ então não há mais o que provar. Agora, se $x \notin U$ então $b \in U$, mas isso significa que b está no extremo de algum caminho M-alternante que denotamos por P. Se y está em P então y é extremo de um caminho M-alternante e com isso $y \in U$. Caso contrário, também haverá um caminho M-alternante com extremo y: ou P, x, y no caso $x \notin V(P)$, ou P', b caso $x \in V(P)$, onde P' denota um subcaminho alternante de P com um extremo em x. Em ambos os casos teremos $y \in U$ pois, como M é máximo, não pode haver caminho M-aumentante em G. Em todos os casos a aresta $\{x, y\}$ encontra U, portanto, U é cobertura. Logo $|U| \geq \nu(G)$ e

$$\mu(G) \geq \nu(G). \tag{5.5}$$

Das equações (5.3) e (5.5) concluímos que $\nu(G) = \mu(G)$. □

Observamos que computar a cobertura mínima de um grafo qualquer, isto é, se levamos em conta grafos não-bipartidos, é um problema NP-difícil (exercício 179) enquanto que μ pode ser computado em tempo polinomial (veja [11]), portanto, não vale a igualdade no caso geral e ν pode ser computado em tempo polinomial no caso bipartido.

O seguinte resultado, também é bastante conhecido, dá uma condição necessária e suficiente para que exista um emparelhamento que satura umas das partes de um grafo bipartido. Dizemos que M satura o subconjunto de vértices U se todo vértice de U é saturado por alguma aresta de M.

Teorema 43 (Teorema de Hall, 1935)**.** *Em todo grafo bipartido $G = (A \cup B, E)$ existe um emparelhamento que satura 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 satura 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. Tomemos os conjuntos $A' = A \cap U$, $A'' = A \setminus A'$ e $B' = B \cap U$, $B'' = B \setminus B'$.

Se não existe um emparelhamento que satura A então do teorema 42 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 Kőnig. 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 satura A.

Se $|A| = 1$ então $|N(A)| \geq 1$, portanto, existe um emparelhamento que satura A. Seja $G = (A \cup B, E)$ um grafo bipartido com $|A| > 1$ e que satisfaz a condiçao de Hall (5.6), em particular, $|B| \geq |A|$.

Suponhamos que todo grafo bipartido $(A' \cup B', E)$ com $|A'| < |A|$ que satisfaz a condição de Hall (5.6) tem um emparelhamento que satura 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 os vértices $A' = A \setminus \{a\}$ e $B' = B \setminus \{b\}$. Nesse caso, para cada $S \subseteq A' \subset A$ vale que $|N_{G'}(S)| \geq |S|$ (justifique) e pela hipótese indutiva concluímos que existe M que satura A'. Portanto $M \cup \{\{a, b\}\}$ satura 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 satura 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 satura $\overline{S}$. Para concluir a demonstração é suficiente observar que $M \cup M'$ é um emparelhamento em G que satura A. □

Corolário 44 (Forma defectiva do teorema de Hall)**.** *Em todo grafo bipartido $G = (A \cup B, E)$ existe um emparelhamento que satura A a menos de d vértices se, e somente se,*

$$|N(S)| \geq |S| - d \text{ para todo } S \subseteq A. \tag{5.8}$$

5.2.1 Método húngaro

O algoritmo abaixo recebe um grafo bipartido $G = (A \cup B, E)$ e devolve um emparelhamento que satura A ou um subconjunto $S \subseteq A$ tal que $|N(S)| < |S|$ cuja existência é garantida pelo teorema de Hall. A ideia do algoritmo é construir caminhos alternantes a partir de um vértice não-saturado em A e caso ache um caminho aumentante, então computa um emparelhamento maior e recomeça, senão os caminhos alternantes determinam um conjunto que viola a condição de Hall.

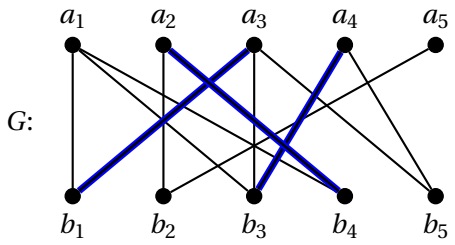


Figura 5.3: ilustação do método húngaro.

Por exemplo, suponha que já temos o emparelhamento dado pelas arestas em destaque no grafo da figura 5.3 abaixo.

Começamos pelo vértice a_1 não-saturado por M. E escolhemos um vizinho de a_1. Se existir algum vizinho não saturado, 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 (figura 5.4(b)). O próximo passo é continuar a busca a partir de um vizinho dos vértices da forma a_i já escolhidos (figura 5.4 (c) e (d)). Notemos que basta buscar tais vizinhos

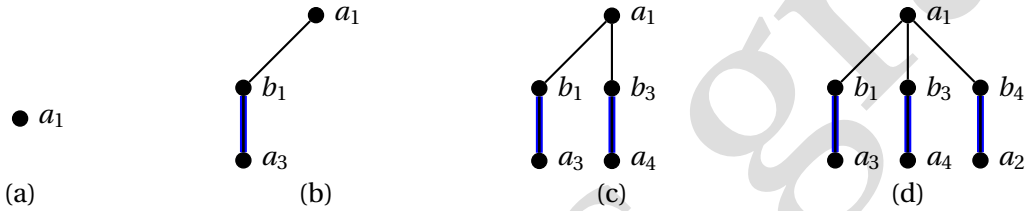


Figura 5.4: busca por caminhos aumentantes.

dentre os vértices ainda não escolhidos, no nosso exemplo, após o estágio representado pela figura 5.4(d) não há necessidade de considerar a aresta $\{a_3, b_3\}$ pois o novo caminho alternante definido por essa aresta seria redundante para nossos propósitos. No próximo estágio dado pela figura 5.5 chegamos a um caminho aumentante. Quando o algoritmo descobre um caminho aumentante, usa-o para obter um emparelhamento com mais aresta e recomeça o processo. Caso contrário, teremos construído caminhos alternantes que começam num vértice não-saturado e todos terminam num vértice saturado. Os vértices desses caminhos definem um conjunto que viola a condição de Hall. Por exemplo, na figura abaixo o algoritmo começa pelo vértice não-saturado a_4. Esse vértice tem os vizinhos b_2 e b_3 que estão cobertos pelas arestas $\{a_1, b_2\}$ e $\{a_3, b_3\}$ 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 saturar A.

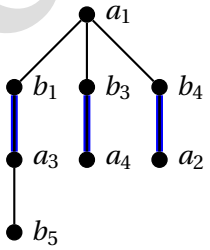


Figura 5.5: caminho aumentante.

Essa ideia 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. Suponhamos que M não satura A e seja $u \in A$ não saturado por

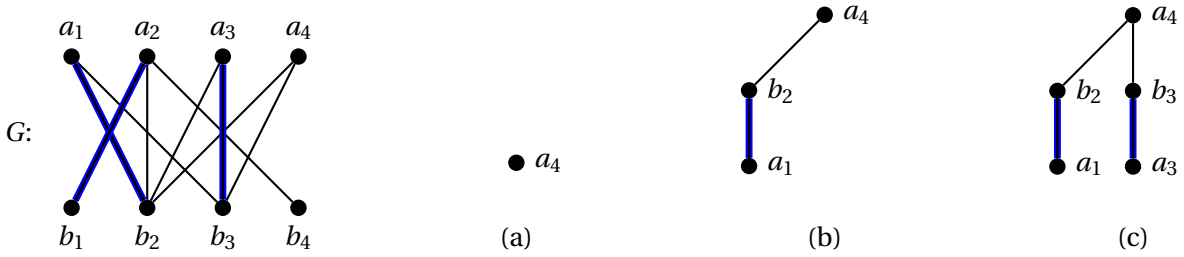


Figura 5.6: $S = \{a_1, a_3, a_4\}$ viola a condição de Hall.

M. Denotemos por C o subconjunto de $A \cup B$ formado por todos os vértices alcançáveis a partir de u por caminho M-alternante. Tomemos $S = C \cap A$ e $T = C \cap B$.

Certamente, $T \subseteq N(S)$. Agora, se existe $x \in S$ com um vizinho fora de T, digamos $b \in B \setminus T$, como $b \notin T$, o vértice b não é saturado por M logo 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 saturados por M (justifique), temos $|T| = |S| - 1$, logo $|N(S)| = |S| - 1 < |S|$ o que contraria a hipótese da condição de Hall vale sobre G, logo M satura A. □

Não é difícil extrair um algoritmo dessa demonstração, esse algoritmo pode ser escrito da seguinte maneira.

Dado : um grafo bipartido $G = (A \cup B, E)$.
Devolve: emparelhamento que satura A ou um obstáculo de Hall $S \subseteq A$.

1 comece com M vazio;
2 **se** *existe* $u \in A$ *não coberto por* M **então** $(S, T) \leftarrow (\{u\}, \varnothing)$;
3 **senão** devolva M ;
4 **se** $N(S) = T$ **então** devolva S ;
5 **senão** escolha $b \in N(S) \setminus T$;
6 **se** *existe* $a \in V \setminus S$ *tal que* $\{a, b\} \in M$ **então**
7 | insira a em S;
8 | insira b em T;
9 | volte para 4;
10 **senão**
11 | seja P o caminho M-aumentante de u até b;
12 | atribua a M o emparelhamento $M \triangle E(P)$;
13 | volte para 2.

Algoritmo 16: Húngaro(A, B, E)

Análise e correção do algoritmo de húngaro.

A correção do algoritmo segue da terceira prova do teorema de Hall. O tempo para a busca de um caminho M-aumentante é $O(|V| + |E|)$ (uma busca) no pior caso. Fazendo uma busca pra cada vértice resulta em $O(|V|(|V| + |E|))$.

5.2.2 Aplicações do teorema de Hall

Transversais: se $S = \{S_1, S_2, \ldots, S_k\}$ é uma família de subconjuntos finitos de um universo U então uma transversal de S é um subconjunto $T \subseteq U$ de cardinalidade k que tem um elemento distinto de cada S_i. Uma transversal de S ou uma obstrução para a sua existência pode ser obtida definindo-se o grafo bipartido com $A = \{1, 2, \ldots, k\}$ e $B = U$ e $\{i, s\} \in E$ se e só se $s \in S_i$.

Pelo teorema de Hall: *S admite uma transversal se, e só se, a união de quaisquer m elementos de S tem cardinalidade pelo menos m.*

Retângulos latinos: uma matriz $m \times n$ cujas entradas são tomadas de $\{1,2,\ldots,n\}$ e as entradas de cada linha e cada coluna são diferentes é um retângulo latino $m \times n$. Quando $m = n$ dizemos quadrado latino.

Todo retângulo latino $m \times n$, $m < n$, pode ser estendido para um quadrado latino pela adição de $n - m$ novas linhas.

O seguinte argumento mostra como obtemos um retângulo latino $m+1 \times n$ de um $m \times n$. Tomemos $S_i \subseteq \{1,2,\ldots,n\}$ o conjunto dos elementos que não ocorrem na linha i do retângulo latino M. Para cada coluna i vale $|S_i| = n - m$, portanto, para qualquer subconjunto I de índices $\sum_{i\in I} |S_i| = |I|(n-m)$. Notemos que $|\bigcup_{i\in I} S_i| \geq |I|$, caso contrário pelo menos um elemento da união estaria em mais que $n-m$ dos conjuntos S_i, portanto, a família de conjuntos $\{S_i\}_i$ tem uma transversal. Essa transversal pode ser usada como uma nova linha do retângulo latino.

Exercícios

Exercício 169. Refaça os exercícios 2 e 22.

Exercício 170. 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	×		×			

Modifique o algoritmo acima para que determine o maior número de professores que a escola pode contratar.

Exercício 171 (Frobenius, 1917). Prove que se G é bipartido e k-regular, $k > 0$, então G tem emparelhamento perfeito.

Exercício 172. Prove que se G é bipartido então existe um emparelhamento que satura todos os vértices de grau $\Delta(G)$.

Exercício 173. 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 174. Prove o corolário 44. (Dica: Adicione d vértices novos a B e faça-os adjacentes a todos os vértices de A.)

Exercício 175. 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 176. Seja $G = (A \cup B, E)$ um grafo bipartido e M um emparelhamento em G. Seja $U \subset V(G)$ o conjunto dos vértices saturados 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 177. Prove que todo emparelhamento máximo em $G = (A \cup B, E)$ tem cardinalidade

$$\min_{U \subseteq A} |A| - |U| + |N(U)|.$$

Exercício 178. 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 179. Como foi dito na observação 1, página 11, 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 180. Modifique o algoritmo Húngaro(A,B,E) para que ele devolva um emparelhamento máximo.

Exercício 181. Faça uma análise detalhada da complexidade do algoritmo Húngaro e deduza que é $O(|V||E|)$.

Exercício 182. Escreva um algoritmo de tempo polinomial para computar $\alpha(G)$ quando G é bipartido.

Exercício 183. Escreva um algoritmo que, dado G, determine um emparelhamento *maximal* em G, isto é, emparelhamento M tal que não exista um emparelhamento M' com $M \subset M'$. Use esse algoritmo para determinar uma cobertura S tal que $|S| \leq 2\nu(G)$.

Exercício 184. 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

$$\mu_\rho(G) = \max\{p(M)\colon M \text{ é emparelhamento em } G\}$$

o peso de um emparelhamento de peso máximo em G. Usando a ideia do Kruskal, escrevemos o seguinte algoritmo

Dado : um grafo G com pesos não-negativos nas arestas.
Devolve: um emparelhamento em G.

```
1 M ← ∅;
2 F ← fila das arestas em ordem decrescente 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.
```

Algoritmo 17: Emparelhamento_quase_máximo(G)

Prove que a resposta do algoritmo satisfaz $p(M) \geq \mu_\rho(G)/2$.

Exercício 185. Na primeira prova que apresentamos do teorema de Hall deduzimos o resultado do teorema de Kőnig. Esses teoremas são, de fato, equivalentes. Demonstre o teorema de Kőnig ($\mu > \nu$) a partir do teorema de Hall.

Exercício 186. Prove que o teorema de Hall é equivalente ao teorema de Menger (teorema 35, página 58).

5.3 Coloração de arestas

Uma k-aresta-coloração de G é uma função $c\colon E(G) \to \{1,2,\ldots,k\}$; os elementos no contra-domínio são as cores e $c(e)$ é a cor da aresta e. A coloração é uma coloração própria se arestas da mesma cor não compartilham vértice, isto é, $c(e) \neq c(f)$ sempre que $e \neq f$ e $e \cap f \neq \varnothing$. De outro modo, c é uma coloração própria se

$$M_i = \{e \in E(G)\colon c(e) = i\}$$

é um emparelhamento em G, para todo $i \in \{1,2,\ldots,k\}$. Na figura 5.7 apresentamos um exemplo de coloração própria das arestas de um grafo usando 5 cores.

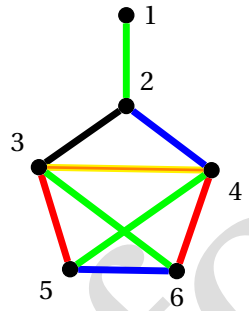


Figura 5.7: coloração própria.

O índice cromático de um grafo G é o parâmetro

$$\chi'(G) = \min\{k\colon G \text{ admite uma } k\text{-aresta-coloração própria}\}.$$

Claramente, vale

$$\Delta(G) \leq \chi'(G) \leq |E(G)|. \tag{5.9}$$

Num grafo estrela, que é o grafo bipartido completo $K^{1,n}$, vale a igualdade de ambos os lados da equação acima. Num circuito ímpar de ordem maior que 3 as desigualdades são estritas.

O índice cromático é monótono pois uma coloração própria das arestas de G também é uma coloração própria das arestas de qualquer subgrafo logo

$$\chi'(H) \leq \chi'(G) \text{ para todo } H \subseteq G. \tag{5.10}$$

Proposição 45. *Se G é bipartido e k-regular então $\chi'(G) = k$.*

Demonstração. Sabemos que $\chi'(G) \geq k$. Vamos provar usando indução em k que há uma k-coloração própria das arestas de G. Para $k = 1$ as arestas do grafo é um emparelhamento perfeito, portanto $\chi'(G) = 1$. Assumamos que $\chi'(H) = \ell$ para todo grafo bipartido ℓ-regular com $\ell < k$.

Seja G um grafo bipartido k-regular com $k > 1$. Pelo exercício 171 o grafo G tem um emparelhamento perfeito M. O subgrafo $G - M$ é $(k-1)$-regular, logo admite uma $(k-1)$-aresta-coloração própria c'. A coloração

$$c(e) = \begin{cases} c'(e) & \text{se } e \notin M \\ k & \text{se } e \in M \end{cases}$$

é uma k-aresta-coloração própria das arestas de G. □

Para grafos bipartidos vale a cota inferior de (5.9).

Teorema 46 (Kőnig, 1916)**.** *Se G é bipartido então $\chi'(G) = \Delta(G)$.*

Demonstração. Seja G um grafo bipartido de grau máximo Δ e tomemos uma coleção $M_1,\ldots,M_\Delta$ de emparelhamentos tal que $|M_1 \cup \cdots \cup M_\Delta|$ é máximo.

Suponhamos que exista $\{x,y\} \in E(G) \setminus (M_1 \cup \cdots \cup M_\Delta)$, assim deve existir i tal que M_i não satura x e deve existir j tal que M_j não satura y.

Se $i = j$ então podemos acrescentar $\{x,y\}$ ao emparelhamento M_i contrariando a maximalidade de $|M_1 \cup \cdots \cup M_\Delta|$.

Se $i \neq j$ então consideremos o subgrafo H induzido por $M_i \cup M_j \cup \{x, y\}$. Dessa forma $\Delta(H) \leq 2$ logo seus componentes conexos são circuitos e caminhos (exercício 103). Se $\{x, y\}$ está num circuito C, então a outra aresta de C que incide em x é de M_j e a outra aresta de C que incide em y é de M_i e assim, alternadamente, donde concluímos que C é ímpar, um absurdo. Portanto $\{x, y\}$ deve estar num caminho P, mas nesse caso podemos rearranjar as arestas de M_i e M_j em P de modo a cobrir todas as arestas, contrariando a maximalidade da união.

Com essa contradição concluímos que não pode existir $\{x, y\} \in E(G) \setminus (M_1 \cup \cdots \cup M_\Delta)$, logo G admite uma Δ-aresta-coloração própria. □

5.3.1 Teorema de Vizing

Vimos que $\chi'(G) \geq \Delta(G)$. Há exemplos em que $\chi'(G) > \Delta(G)$, e o próximo resultado diz que se esse é o caso então $\chi'(G) = \Delta(G) + 1$.

Teorema 47. *Para todo grafo G,*

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

Demonstração. □

Do ponto de vista computacional, decidir se $\chi'(G)$ é $\Delta(G)$ ou $\Delta(G) + 1$ é NP-difícil.

Exercícios

Exercício 187. Estabeleça o índice cromático de caminhos e circuitos.

Exercício 188. Determine o índice cromático do grafo de Petersen.

Exercício 189. Sejam G um grafo e c uma aresta-coloração. Defina o *grau cromatico* de v segundo c por

$$d_c^{\mathrm{cr}}(v) = |\{i : c(e) = i \text{ para alguma aresta } e \text{ com extremo } v\}|.$$

A coloração c pode ser *melhorada* se existe uma aresta-coloração c' tal que

$$\sum_{v \in V(G)} d_{c'}^{\mathrm{cr}}(v) > \sum_{v \in V(G)} d_c^{\mathrm{cr}}(v).$$

Se a coloração c não pode ser melhorada então ela é *ótima*.

(a) Verifique que $d_c^{\mathrm{cr}}(v) \leq d(v)$ para qualquer aresta-coloração c.

(b) Verifique que $d_c^{\mathrm{cr}}(v) = d(v)$ para qualquer aresta-coloração própria c. Conclua que, nesse caso, c é ótima.

(c) Dê um exemplo de um grafo com uma k-aresta-coloração ótima mas que não admite uma k-aresta-coloração própria.

(d) Prove que uma aresta-coloração c de G é própria se, e somente se, $d_c^{\mathrm{cr}}(v) = d(v)$ para todo $v \in V(G)$.

Exercício 190. Deduza o teorema 46 do exercício 172.

6 | Grafos dirigidos

capítulo não revisado

Neste capítulo expomos alguns tópicos sobre grafos dirigidos, para mais sobre o assunto veja [1]. 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. 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 $G = (V,E)$ é o grafo

$$SG = \left(V, \bigcup_{(u,v)\in E} \{\{u,v\}\}\right)$$

6.1 Representação computacional e percurso

Como fizemos antes, 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. No caso de matriz, a matriz de adjacências é dada por $a_{i,j} = 1$ se e só se (i,j) é aresta. Notemos que a matriz agora pode não ser simétrica.

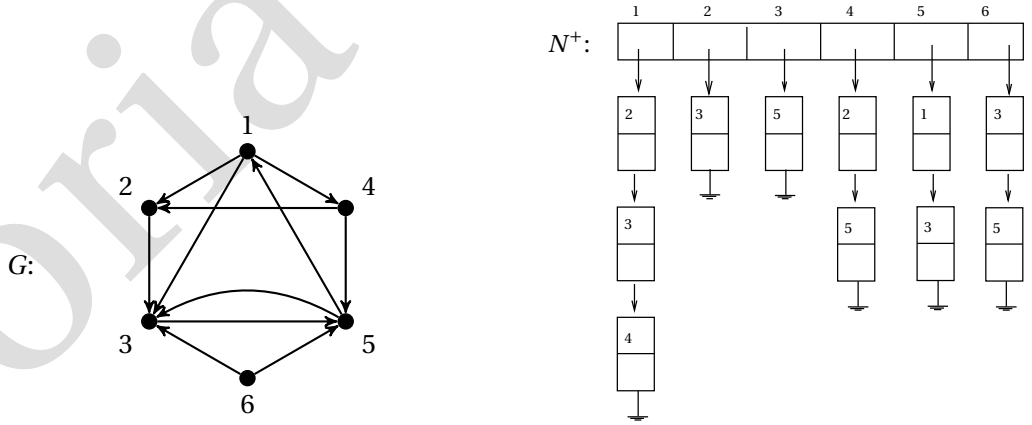


Figura 6.1: Diagrama de um grafo dirigido e uma representação por lista de adjacências.

Uma busca em profundidade rotulada, algoritmo 3 na página 32, executada num grafo dirigido resulta numa classificação das arestas em quatro tipos, ao invés dos dois tipos que ocorrem no caso não-dirigido. Usaremos a terminologia apresentada na 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, 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 $chega(v) < sai(v) < chega(u) < sai(u)$.

A figura abaixo esquematiza uma busca em profundidade rotulada no grafo dirigido do exemplo 6.1.

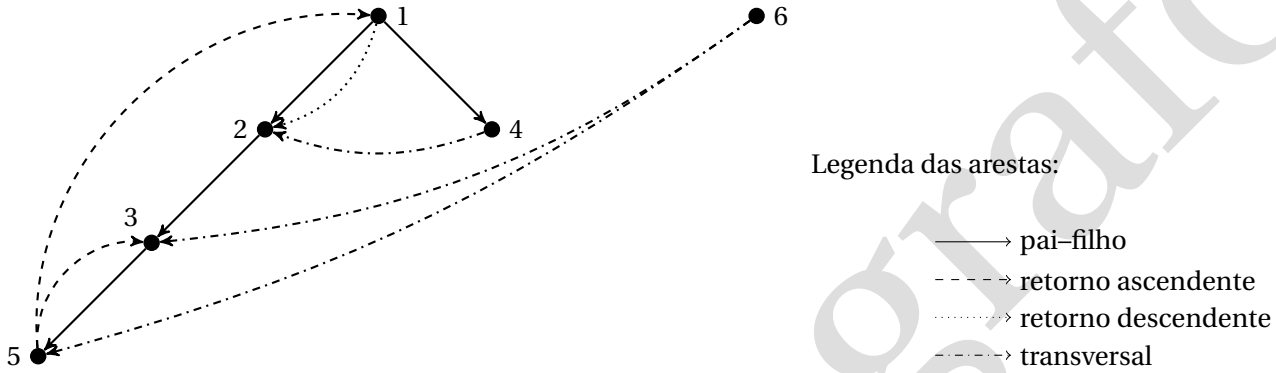


Figura 6.2: Busca em profundidade rotulada.

Um caminho orientado é o grafo dirigido P definido por uma sequência 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 $(x_1 \to x_k)$-caminho. Um circuito orientado é o grafo dirigido C definido por uma 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 $(x_k, x_1) \in E(C)$.

Exercícios

Exercício 191 (Fecho transitivo). Se $D = (V,E)$ é um grafo dirigido então o fecho transitivo de D é o grafo $D^* = (V, E^*)$ onde

$$E^* = \{(i,j) \in V \times V : \text{ existe um } (i \to j)\text{-caminho em } D\}.$$

Escreva um algoritmo de tempo $O(|V|^3)$ para determinar o fecho transitivo de um grafo dirigido.

Exercício 192. Ordenação topológica é o problema: 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.

(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. Prove que o algoritmo está correto.

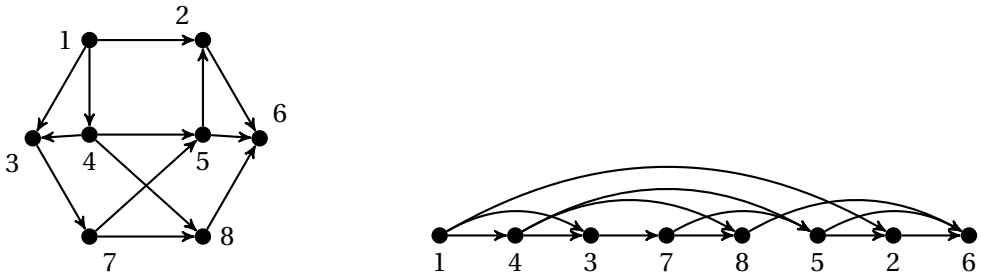


Figura 6.3: Um grafo orientado e uma ordenação topológica de seus vértices: $1 \prec 4 \prec 3 \prec 7 \prec 8 \prec 5 \prec 2 \prec 6$.

6.2 Caminhos mínimos em grafos dirigidos com pesos nas arestas

Além da orientação nas arestas, 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,

$$\operatorname{dist}(u, v) = \min\{c(P)\colon P \text{ é um } (u \to v)\text{-caminho}\}$$

quando existe algum caminho, caso contrário convencionamos que $\operatorname{dist}(u, v) = \infty$. Um $(u \to v)$-caminho de comprimento $\operatorname{dist}(u, v)$ é chamado de **caminho mínimo**.

Exemplo 14. O seguinte diagrama representa um grafo dirigido com pesos nas arestas

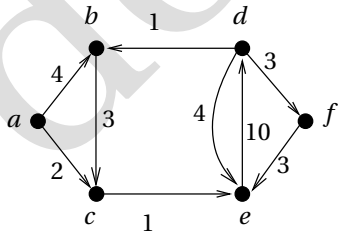


Figura 6.4: Diagrama de um grafo dirigido com peso nas arestas. Note que $\operatorname{dist}(c, e) \neq \operatorname{dist}(e, c)$, $\operatorname{dist}(a, f) = 10$.

Observação 4. Num grafo dirigido G podemos ter

$$\operatorname{dist}(u, v) \neq \operatorname{dist}(v, u)$$

e vale a seguinte forma da desigualdade triangular, para todo $(v, u) \in E(G)$

$$\operatorname{dist}(s, u) \leq \operatorname{dist}(s, v) + \rho(v, u) \tag{6.4}$$

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 36, resolve o seguinte problema computacional: 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 $\operatorname{dist}(s, v)$ 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 93), 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 um laço 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 $\text{dist}(s, v)$ 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 36, para cada aresta

para cada $(u, w) \in E(G)$ **faça** Relaxação(u, w). ;

onde

Algoritmo 18: Relaxação(u, w)

```
1 se d[w] > d[u] + ρ(u, w) então d[w] ← d[u] + ρ(u, w).;
```

assim, na primeira rodada de relaxações em $E(G)$ temos que $\text{dist}_G(s, v_1)$ está determinado; na segunda, $\text{dist}_G(s, v_2)$ está determinado (veja o lema 17, que também vale no caso dirigido). Na k-ésima rodada $\text{dist}_G(s, v)$ está determinado (veja exercício 195 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 19: 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 $\text{dist}_G(s, v)$, 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  ;
3  d[s] ← 0;
4  cont ← 1;
5  enquanto cont ≤ |V| − 1 faça
6      para cada (u, w) ∈ E(G) faça Relaxação(u, w);
7      ;
8      cont ← cont + 1;
9  para cada (u, w) ∈ E(G) faça
10     se d[w] > d[u] + ρ(u, w) então devolva falso;
11     ;
12 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 195.

Lema 48. 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

$$d(w] = \text{dist}_G(s, w) \leq \text{dist}_G(s, u) + \rho(u, w) = d(u) + \rho(u, w),$$

portanto o algoritmo devolve *verdadeiro*. □

Para a complexidade, notemos que o tempo dos laços aninhados nas linhas 5 e 6 predomina e resulta em $O(|V||E|)$.

Exercícios

Exercício 193. 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 194. Prove a desigualdade (6.4).

Exercício 195 (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) $d(v) \geq \text{dist}_G(s, v)$ 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 $\text{dist}_G(s, v) = \infty$ então $d(v) = \infty$;

(d) se $s, \ldots, u, w$ é um $(s \to w)$-caminho mínimo e $d(u) = \text{dist}_G(s, u)$ então Relaxação(u, w) resulta em $d[w] = \text{dist}_G(s, w)$;

(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), $(v_1, v_2), \ldots,$ (v_{k-1}, v_k), nessa ordem e com possíveis outras relaxações intermediárias, então teremos no final da seqüência $d[w] = \text{dist}_G(s, w)$.

Exercício 196. 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.

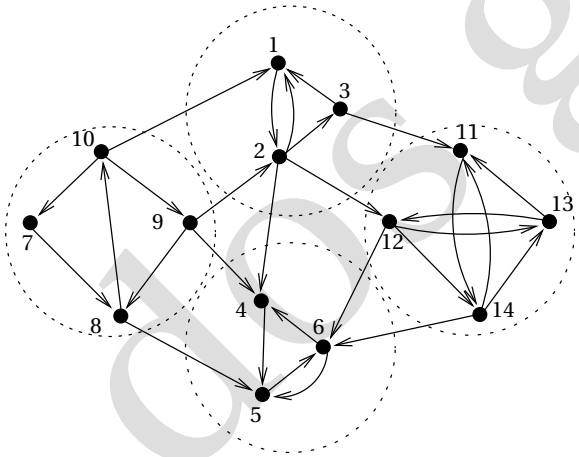
Exercício 197. Suponha que existe um algoritmo $\mathscr{A}(G, u, v)$ que determina em tempo polinomial um $(u \to v)$-caminho mínimo no grafo dirigido com pesos reais nas arestas G. Mostre como usar esse algoritmo para determinar se G tem um circuito (orientado) hamiltoniano em tempo polinomial. (*Dica:* atribua peso -1 a todas as arestas do grafo e compute $\mathscr{A}(G, u, v)$ 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 $\mathscr{A}$ 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 15. O diagrama abaixo mostra um grafo dirigido, cada um dos quatro círculos identifica um componente fortemente conexo do grafo.



Dado um grafo dirigido G, queremos computar 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 $E(G)$: $G^{\mathrm{T}} = \big(V(G), \{(u, v) \in V \times V : (v, u) \in E(G)\}\big)$.

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 6.5(a) abaixo).

Observação 5. 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 6. 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 16. Na figura abaixo mostramos (a) o grafo dos componentes fortemente conexos do grafo dirigido mostrado no exemplo 15 e também (b) uma ordenação topológica dos vértices do grafo das componentes.

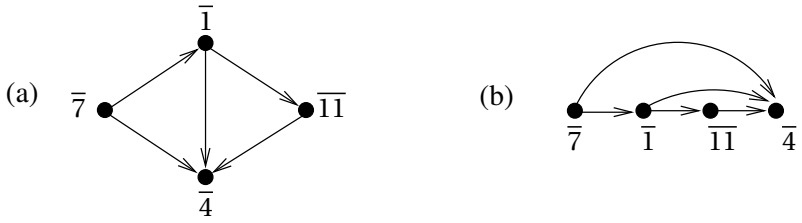


Figura 6.5: (a) Grafo dos componentes fortemente conexos do grafo do exemplo 15, $\overline{1} = \{1,2,3\}$, $\overline{4} = \{4,5,6\}$, $\overline{7} = \{7,8,9,10\}$ e $\overline{11} = \{11,12,13,14\}$. (b) Ordenação topológica do grafo das componentes.

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 20: 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 49. *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*

$$\max_{u \in W} \mathrm{sai}(u) > \max_{v \in W'} \mathrm{sai}(v).$$

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 50. *Sejam W e W' dois componentes fortemente conexos de G. Suponha que existe uma aresta $(u,v) \in E(G^{\mathrm{T}})$ tal que $u \in W$ e $v \in W'$. Então*

$$\max_{u \in W} \mathrm{sai}(u) < \max_{v \in W'} \mathrm{sai}(v).$$

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 50 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 $sai(u) = \max\{sai(v) : v \in V(G) - W\}$ e, novamente pelo corolário 50, 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 21: 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 |   ;
7 sai(v) ← cont;
8 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 22: 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);
6      ;
   /* Ordene os vértices por ordem decrescente de sai[] */
7  L ← lista ordenada de V(G) por ordem decrescente de sai[];
   /* Determine o grafo transposto */
8  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 */
9  cont ← 0;
10 c ← 0;
11 para cada k ∈ V(G) faça
12     sai[k] ← 0;
13     chega[k] ← 0;
14 para cada k ∈ L faça
15     se chega[k] = 0 então
16         c ← c + 1;
17         BP(G^T, k, c);
```

Agora, damos uma prova de que o algoritmo CFC está correto.

Teorema 51. *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 14 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 14 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 $W_i = \{v\colon cfc(v) = i\}$, $1 \le i \le m+1$, são $m+1$ componentes fortemente conexos de G.

Após a atribuição $c = m+1$ na linha 16, temos uma chamada da busca em profundidade BP($G^{\mathrm{T}}, k, m+1$), 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 50, 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 $\mathrm{BP}(G^{\mathrm{T}}, k, m+1)$. Assim, nenhum vértice fora de W_{m+1} será descendente de k no final de $\mathrm{BP}(G^{\mathrm{T}}, k, m+1)$. □

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 $A^{\mathrm{T}}(i,j) = A(j,i)$. 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 9 tem custo $O(1)$, a 11 $O(|V|)$ e o laço na linha 14 tem custo $O(|V|+|E|)$.

Teorema 52. CFC(G) *tem complexidade* $O(|V|\log|V|+|E|)$.

Exercícios

Exercício 198. Demonstre o lema 49.

Exercício 199. Mostre que o algoritmo 20 pode ser implementado com complexidade $O(|V|+|E|)$.

Exercício 200. Se ao invés de usarmos uma busca em profundidade em G^{T} por ordem decrescente de $sai[\,]$ no algoritmo 22 usarmos uma busca em profundidade em G por ordem crescente de $chega[\,]$ o algoritmo determinaria os componentes fortemente conexos do grafo?

Exercício 201. Se ao invés de usarmos uma busca em profundidade em G^{T} por ordem decrescente de $sai[\,]$ no algoritmo 22 usarmos uma busca em profundidade em G por ordem crescente de $sai[\,]$ o algoritmo determinaria os componentes fortemente conexos do grafo?

Exercício 202. 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 203. 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 204. Escreva um algoritmo de complexidade $O(|V|+|E|)$ para computar o grafo G^* dos componentes fortemente conexos de G.

Exercício 205. 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 $G = (V, E)$ é um grafo dirigido, c é uma função $c\colon E \to \mathbb{R}^+$ 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 $0 \le f(e) \le c(e)$;
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 "chega" em t.

Proposição 53. *Se $N = (G, c, s, t)$ é uma rede e f um s-t 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

$$\sum_{e \in E^-(s)} f(e) = \sum_{e \in E^+(t)} f(e)$$

segue da igualdade das equações. □

O valor do somatório na equação (6.6) é chamado valor do fluxo e é denotado por **val**(**f**)

$$\text{val}(f) = \sum_{e \in E^-(s)} f(e). \tag{6.7}$$

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 $\text{val}(f) \geq \text{val}(g)$ para todo s-t fluxo g na rede N.

Para atacar o problema, analisaremos a relação enter fluxos e cortes numa rede; como seria esperado, fluxo e corte em redes são conceitos estreitamente relacionados pois cortes com arestas de baixa capacidade deve suportar pouco fluxo. Em um grafo dirigido G um **corte separa os vértices** s **e** t, ou simplesmente s-t corte, é um conjunto de arestas

$$E(S, \bar{S}) = \{(u, v) \in E(G) : u \in S \text{ e } v \in \bar{S}\} \text{ tal que } s \in S \text{ e } t \in \bar{S}$$

e a capacidade do corte é

$$c(S) = C(S, \bar{S}) = \sum_{e \in E(S, \bar{S})} c(e); \tag{6.8}$$

e o fluxo no corte é

$$f(S) = f(S, \bar{S}) = \sum_{e \in E(S, \bar{S})} f(e) - \sum_{e \in E(\bar{S}, S)} f(e). \tag{6.9}$$

Segue imediatamente da definição de fluxo que $f(S) \leq c(S)$ para todo s-t corte S e, mais que isso, vale a afirmação de que o valor de um s-t fluxo é no máximo a capacidade de qualquer s-t corte.

Proposição 54. *Se $N = (G, c, s, t)$ é uma rede e f um s-t fluxo, então*

$$\text{val}(f) \leq c(S). \tag{6.10}$$

Demonstração. De fato, pela definição e valor e pela restrição 2 na definição de fluxo

$$\text{val}(f) = \sum_{v \in S} \left(\sum_{e \in E^-(v)} f(e) - \sum_{e \in E^+(v)} f(e) \right)$$

mas cada aresta da forma $e = (x, y) \in S \times S$ contribui duas vezes na soma, quando $v = x$ contribui com $f(e)$ e quando $v = y$ contribui com $-f(e)$, logo essas contribuições se cancelam e ficamos com

$$\text{val}(f) = \sum_{v \in S} \left(\sum_{e \in E^-(v) \cap S \times \bar{S}} f(e) - \sum_{e \in E^+(v) \cap \bar{S} \times S} f(e) \right) = f(S). \tag{6.11}$$

□

Proposição 55. *Para todo s-t fluxo f e todo s-t corte S vale* $\text{val}(f) \leq c(S)$, *em particular*

$$\max\{\text{val}(f) : f \text{ é um } s\text{-}t \text{ fluxo}\} \leq \min\{c(S) : S \text{ é um } s\text{-}t \text{ corte}\}. \tag{6.12}$$

Demonstração. Para todo s-t corte S, temos por (6.11) $\text{val}(f) = f(S)$ e pela definição de fluxo (item 1) o fluxo em cada aresta é limitado pela capacidade da aresta, logo $f(S) \leq c(S)$, donde segue a desigualdade. □

Um s-t **corte mínimo** é um s-t corte S de capacidade $c(S) \leq c(S')$ para todo s-t corte S' (lado direito de (6.12)).

Corolário 56. *Para todo s-t fluxo f e todo s-t corte S, se* $\text{val}(f) = c(S)$ *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

$$\text{val}(f) \leq \text{val}(f^*) \leq c(S^*) \leq c(S).$$

De $\text{val}(f) = c(S)$ vale a igualdade nas equações acima. □

Sejam $N = (G, c, s, t)$ uma rede, $P \subset SG$ um caminho com extremos s e t no *grafo subjacente* ao grafo dirigido G e f um s-t fluxo na rede N. Digamos que $P = x_0, x_1, \ldots, x_{m-1}, x_m$ com $x_0 = s$ e $x_m = t$. Dizemos que P é f-aumentante se para cada $\{x_i, x_{i+1}\} \in E(P)$ temos

1. $e = (x_i, x_{i+1}) \in E(G)$ e $f(e) < c(e)$, nesse caso dizemos que e é uma aresta *direta* de P, ou
2. $e = (x_{i+1}, x_i) \in E(G)$ e $f(e) > 0$, nesse caso dizemos que e é uma aresta *reversa* de P.

Se P é f-aumentante e definimos

$$\epsilon_1 = \min\{c(e) - f(e) : e \text{ é uma aresta direta de } P\} \tag{6.13}$$

$$\epsilon_2 = \min\{f(e) : e \text{ é uma aresta reversa de } P\} \tag{6.14}$$

$$\delta = \min\{\epsilon_1, \epsilon_2\}. \tag{6.15}$$

e definimos $g : E \to \mathbb{R}$ por

$$g(e) = \begin{cases} f(e) + \delta & \text{se } e \text{ é direta} \\ f(e) - \delta & \text{se } e \text{ é reversa} \\ f(e) & \text{caso contrário,} \end{cases} \tag{6.16}$$

então g é um fluxo (verifique) com $\text{val}(g) = \text{val}(f) + \delta$, $\delta > 0$, 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 demonstrarmos esse resultado, vejamos um exemplo para ilustrar o papel das arestas reversas num caminho aumentante; nos caminhos com somente arestas diretas a idéia é clara pois podemos aumentar o fluxo em cada uma das arestas no caminho do mesmo montante que ainda temos um fluxo.

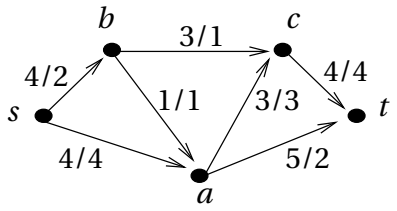


Figura 6.6: Uma rede. Os números representam as capacidades e os fluxos nas arestas na foma c/f.

Exemplo 17. Consideremos a rede representada no diagrama da figura 6.6 abaixo onde os números definem as capacidades das arestas.

O único caminho aumentante nessa rede é o caminho s, b, c, a, t e (c, a) é uma aresta reversa. A aresta (a, c) pode transferir duas unidades de fluxo para (b, c) e podemos aumentar o valor do fluxo. O fluxo obtido através desse caminho está representado na figura 6.7 abaixo.

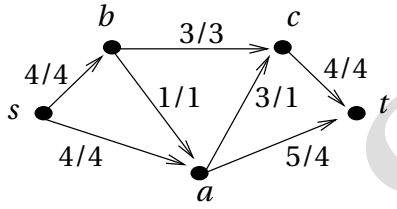


Figura 6.7: Uma rede. O fluxo agora tem maior valor com respeito ao fluxo anterior.

Teorema 57 (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. Já verificamos acima que se existe caminho f-aumentante então f não é máximo. Agora, suponhamos que não exista caminho f-aumentante com extremos s e t.

Definimos S como o conjunto dos vértices v (incluindo s) para os quais há um caminho de f-aumentante com extremos s e v.

Cada aresta e do s-t corte $E(S, \bar{S})$ deve estar saturada, isto é, $f(e) = c(e)$, caso contrário teríamos um caminho aumentante até o extremo de e em $\bar{S}$, e para cada aresta $d \in E(\bar{S}, S)$ vale $f(d) = 0$ (pelo mesmo motivo da saturação), logo $f(S) = c(S)$ e como $\text{val}(f) = f(S)$ (equação (6.11)) pelo corolário 56 f deve ser máximo. □

Teorema 58 (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 $e \in E(G)$. Ademais* $\text{val}(f) = c(S)$ *para algum $S \subset V(G)$.*

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 (6.15) é um inteiro maior que 0. Tome $f_1 = g$, par g definida em (6.16) 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 inteiros e estritamente crescente. Como o valor é limitado pela capacidade de qualquer corte, essa sequência é finita, digamos $f_0, f_1, f_2, \ldots f_n$.

Seja S o conjunto dos vértices v (incluindo s) para os quais há um caminho de f_{n-1}-aumentante com extremos s e v, como descrito na demonstração do teorema 57. Dessa forma $val(f_n) = c(S)$ e pelo corolário 56 $\text{val}(f_n)$ é máximo e $c(S)$ é mínimo. □

Se as capacidade são racionais então podemos recair no teorema acima multiplicando as capacidades pelo seu denominador comum. No caso geral, não demostraremos mas vale o seguinte.

Teorema 59 (Ford e Fulkerson, 1956)**.** *O valor máximo de um fluxo numa rede N é igual a capacidade mínima de um corte em N.* □

Esses resultados dão origem ao seguinte algoritmo.

Algoritmo 23: Ford–Fulkerson(G, c, s, t)

Dado : uma rede (G, c, s, t).
Devolve: Um fluxo de valor máximo.

1 **para cada** $(u, v) \in E(G)$ **faça** $f(u, v) \leftarrow 0$;
2 ;
3 **enquanto** *existe caminho f-aumentante* **faça**
4 escolha um caminho f-aumentante;
5 determine δ como em (6.15);
6 defina g como em (6.16);
7 $f \leftarrow g$;
8 devolva f.

Se as capacidades das arestas são inteiras então esse algoritmo termina 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 [10]) ou ainda não convergir para a solução. Note que o algoritmo não é polinomial no tamanho da entrada (veja a observação 3).

Exemplo 18. Esse é, talvez, o exemplo mais simples da situação em que o algoritmo de Ford–Fulkerson falha, foi descoberto em 1993 por Uri Zwick [24]. 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

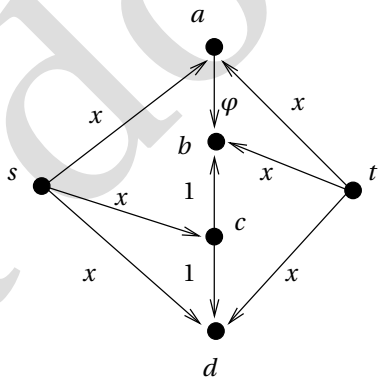


Figura 6.8: Exemplo de uma rede onde o algoritmo não converge, $\varphi = \frac{\sqrt{5}-1}{2}$.

s, c, b, t, as capacidades residuais das arestas $(a, b), (c, b), (cd)$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 4 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 $\text{dist}_{SR_f}(s, t)$ 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 através de um caminho f_i-aumentante de acordo com (6.16), para todo $i \geq 0$.

Proposição 60. *Para todo vértice $v \neq s, t$,*

$$\text{dist}_{SG_{f_{i+1}}}(s, v) \geq \text{dist}_{SG_{f_i}}(s, v)$$

Demonstração. Suponha que a afirmação é falsa e seja k o menor natural para o qual existe um vértice v tal que $\text{dist}_{SG_{f_{k+1}}}(s, v) < \text{dist}_{SG_{f_k}}(s, v)$. Seja $P = s, u_1, \ldots, u_\ell, v$ o caminho mínimo em $SG_{f_{k+1}}$, de modo que

$$\text{dist}_{SG_{f_{k+1}}}(s, v) = \text{dist}_{SG_{f_{k+1}}}(s, u_\ell) + 1. \tag{6.17}$$

Pela escolha de v

$$\text{dist}_{SG_{f_{k+1}}}(s, u_\ell) \geq \text{dist}_{SG_{f_k}}(s, u_\ell), \tag{6.18}$$

portanto, $\text{dist}_{SG_{f_{k+1}}}(s, v) \geq \text{dist}_{SG_{f_k}}(s, u_\ell) + 1$.

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 $\text{dist}_{SG_{f_k}}(s, v) \leq \text{dist}_{SG_{f_k}}(s, u_\ell) + 1$ pela desigualdade triangular; por (6.18) e (6.17) deduzimos que $\text{dist}_{SG_{f_k}}(s, v) \leq \text{dist}_{SG_{f_{k+1}}}(s, v)$, 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, $\text{dist}_{SG_{f_k}}(s, u_\ell) = \text{dist}_{SG_{f_k}}(s, v) + 1$ e como $\text{dist}_{SG_{f_{k+1}}}(s, v) \geq \text{dist}_{SG_{f_k}}(s, u_\ell) + 1$, segue que $\text{dist}_{SG_{f_{k+1}}}(s, v) \geq \text{dist}_{SG_{f_k}}(s, u_\ell) + 2$, uma contradição. □

Teorema 61 (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 (6.15). 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, $\text{dist}_{SG_{f_k}}(s, j) = \text{dist}_{SG_{f_k}}(s, i) + 1$ e $\text{dist}_{SG_{f_\ell}}(s, i) = \text{dist}_{SG_{f_\ell}}(s, j) + 1$.

Como $\text{dist}_{SG_{f_k}}(s, j) \leq \text{dist}_{SG_{f_\ell}}(s, j)$ temos que $\text{dist}_{SG_{f_\ell}}(s, i) = \text{dist}_{SG_{f_\ell}}(s, j) + 1 \geq \text{dist}_{SG_{f_k}}(s, j) + 1 = \text{dist}_{SG_{f_k}}(s, i) + 2$, 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 62. *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 4 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 206. Prove que para quaisquer $S, T \subset V(G)$ vale que $f(S) = f(T)$. Derive desse fato que se $val(f) = c(S)$ então f é máximo e S é mínimo.

Exercício 207. Seja f um fluxo que não é máximo numa rede. Seja g a função dada em (6.16). prove que g é um fluxo na mesma rede.

Exercício 208. 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.

A | Algoritmos e Estruturas de dados

A.1 Algoritmos

A.1.1 Correção de algoritmos

Um algoritmo está correto se a saída está correta para toda entrada. A prova da *correção do algoritmo* tem duas partes: provar que a resposta está correta se o algoritmo terminar, provar que o algoritmo termina.

A seguir, p e q denotam proposições lógicas. Um algoritmo, ou trecho de algoritmo, S está *parcialmente correto* com respeito a **pré-condição** p e a **pós-condição** q se sempre que p for verdadeiro para os valores de entrada de S e S terminar, então q é verdadeiro para os valores de saída de S.

Usamos a notação $p\{S\}q$ para dizer que *S está parcialmente correto com respeito a pré-condição p e a pós-condição q*.

Exemplo Por exemplo, se p é a proposição $x = 1$ e q é a proposição $z = 3$ e S é o trecho
então, $p\{S\}q$. De fato, suponha p. Após a execução de S, $y = 2$ e $z = 1+2$ pois de p temos $x = 1$. Portanto, $z = 3$.

$y \leftarrow 2;$
$z \leftarrow x + y;$

Argumentos válidos para correção parcial de algoritmos.

1. *Composição*

$$p\{S_1\}q \text{ e } q\{S_2\}r \Rightarrow p\{S_1; S_2\}r$$

2. *Condicional*

   (a) No caso {**se** *condição* **então** S} o argumento é

$$\big((p \text{ e } \textit{condição})\{S\}q \text{ e } (p \text{ e não}(\textit{condição})) \Rightarrow q\big)$$
$$\Longrightarrow$$
$$p\{\textbf{se } \textit{condição} \textbf{ então } S\}q$$

   (b) No caso {**se** *condição* **então** S_1 **senão** S_2} o argumento é

$$\big((p \text{ e condição})\{S_1\}q \text{ e } (p \text{ e não(condição)})\{S_2\}q\big)$$
$$\Longrightarrow$$
$$p\{\textbf{se } \textit{condição} \textbf{ então } S_1 \textbf{ senão } S_2\}q$$

3. *Laços* No caso {**enquanto** *condição* **faça** S} o argumento é

$$(p \text{ e condição})\{S\}p$$
$$\Longrightarrow$$
$$p\{\textbf{enquanto } \textit{condição} \textbf{ faça } S\}(\text{não(condição) e } p)$$

   A proposição $(p$ e condição$)\{S\}p$ é o **invariante do laço**. Provar que uma proposição é invariante de laço precisa de indução finita.

Algoritmo 24: multiplica($m, n : inteiros$)

Dado : m e n inteiros
Devolve: $m \cdot n$
/* p: m e n inteiros */
se $n < 0$ **então** $a \leftarrow -n$; ;
senão $a \leftarrow n$;
;
/* q: p e $a = |n|$ */
$k \leftarrow 0$;
$x \leftarrow 0$;
/* r: q e $k = 0$ e $x = 0$ */
enquanto $k < a$ **faça**
    /* (invariante do laço) $x = mk$ e $k \leq a$ */
    $x \leftarrow x + m$;
    $k \leftarrow k + 1$;
/* s: $x = ma$ e $a = |n|$ */
se $n < 0$ **então** $produto \leftarrow -x$; ;
senão $produto \leftarrow x$;
;
/* t: $produto = mn$ */

Exemplo. Seja S o algoritmo 24.

Sejam p e q as proposições (lógicas) descritas no algoritmo e R o trecho de algoritmo

$$R\text{: } \textbf{se } n < 0 \textbf{ então } a \leftarrow -n; \textbf{ senão } a \leftarrow n.$$

Vamos provar

(A.1) $$p\{R\}q.$$

Assuma $m \in \mathbb{Z}$ e $n \in \mathbb{Z}$. Se $m \in \mathbb{Z}$ e $n \in \mathbb{Z}$ e $n < 0$ então após R terminar temos $a = -n$. Se $n < 0$ então $|n| = -n$ portanto $a = |n|$, portanto

(A.2) $$(m \in \mathbb{Z} \text{ e } n \in \mathbb{Z} \text{ e } n < 0)\{R\}(a = |n|).$$

Agora, se $m \in \mathbb{Z}$ e $n \in \mathbb{Z}$ e não$(n < 0)$ então após R terminar temos $a = n$. Se vale não$(n < 0)$ então $|n| = n$ portanto $a = |n|$. Portanto,

(A.3) $$(m \in \mathbb{Z} \text{ e } n \in \mathbb{Z} \text{ e não}(n < 0))\{R\}(a = |n|).$$

De (A.2), (A.3) e do argumento 2(b) temos (A.1), e, portanto, $p\{R\}q$.
Agora, considere o trecho

$$T\text{: } k \leftarrow 0;\ x \leftarrow 0$$

e vamos provar

$$q\{T\}r$$

para q e r descritos no algoritmo. Após $\{k \leftarrow 0;\ x \leftarrow 0\}$ vale $k = 0$ e $x = 0$, portanto

$$q\{k \leftarrow 0;\ x \leftarrow 0\}(q \text{ e } k = 0 \text{ e } x = 0) \tag{A.4}$$

para qualquer proposição verdadeira q, em particular quando

$$q\colon\ m \in \mathbb{Z} \text{ e } n \in \mathbb{Z} \text{ e } a = |n|$$

é verdadeira, logo $q\{T\}r$.

Assumamos r, isto é, que vele q e $k = 0$ e $x = 0$. Se $k < a$ (a condição do laço) então o invariante

$$x = mk \text{ e } k \leq a$$

é verdadeiro antes da primeira rodada do laço pois $0 = m0$ e $0 \leq |n|$. Assuma

$$x = mk \text{ e } k \leq a \text{ e } k < a$$

antes de uma execução do trecho

$$U\colon\ x \leftarrow x + m;\ k \leftarrow k + 1;$$

Considere uma execução de U e denote por x' e k' os valores das variáveis x e k *antes dessa execução.* Então, por hipótese

$$x' = mk' \text{ e } k' \leq a$$

e (condição do laço) $k' < a$; após a execução de U teremos $x = x' + m$ e $k = k' + 1$. Agora,

- Se $x = x' + m$ e $x' = mk'$, então $x = (mk') + m = m(k' + 1)$.
- Se $k = k' + 1$ $x = m(k' + 1)$ então $x = mk$.
- Se $k' < a$ então $k' + 1 \leq a$.

Portanto, após a execução de U o invariante é verdadeiro, ou seja,

$$(x = mk \text{ e } k \leq a \text{ e } k < a)\{U\}(x = mk \text{ e } k \leq a)$$

e do argumento para laços, temos

$$(x = mk \text{ e } k \leq a)\{\textbf{enquanto } k < a \textbf{ faça } U\}(x = mk)$$

ainda, não$(k < a)$ e $x = mk$ e $k \leq a$ equivale a $x = mk$ e $k = a$. Portanto,

$$r\{\textbf{enquanto } k < a \textbf{ faça } U\}(x = ma)$$

logo $r\{\textbf{enquanto } k < a \textbf{ faça } U\}s$, onde $s\colon\ x = ma$ e $a = |n|$.

Finalmente, assumamos a proposição s, denotemos por Q o trecho de algoritmo

$$\textbf{se } n < 0 \textbf{ então } produto \leftarrow -x; \textbf{ senão } produto \leftarrow x$$

e vamos provar t, isto é, que $produto = mn$. Porém,

- Se s, então $x = m|n|$.
- Se $x = m|n|$ e $n < 0$ então após Q, $produto = -x$.
- Se $produto = -x$ e $x = m|n|$ e $n < 0$ então $produto = -m|n| = -m(-n) = mn$.

Portanto t. Agora,

- se $x = ma$ e $a = |n|$, então $x = m|n|$.
- Se $x = m|n|$ e não$(n < 0)$ então após Q, $produto = x$.
- Se $produto = x$ e $x = m|n|$ e não$(n < 0)$ então $produto = m|n| = mn$.

Portanto, pelo argumento 2.(b) para condicionais

$$(x = ma \text{ e } a = |n|)\{Q\}(produto = mn).$$

Resumindo, provamos

1. $p\{$**se** $n < 0$ **então** $a \leftarrow -n$; **senão** $a \leftarrow n\}q$;
2. $q\{k \leftarrow 0; x \leftarrow 0\}r$;
3. $r\{$**enquanto** $k < a$ **faça** $\{c \leftarrow x + m;\ k \leftarrow k + 1;\}\}s$;
4. $s\{$**se** $n < 0$ **então** $produto \leftarrow -x$; **senão** $produto \leftarrow x\}t$;

De 1 e 2 temos

$$p\{\textbf{se } n < 0 \textbf{ então } a \leftarrow -n;\ \textbf{senão } a \leftarrow n; k \leftarrow 0; x \leftarrow 0\}r. \tag{A.5}$$

pela regra da composição. De 3 e 4 temos, por composição novamente

$$\begin{aligned} r \quad \{ \quad & \textbf{enquanto } k < a \textbf{ faça } \{c \leftarrow x + m;\ k \leftarrow k + 1;\}; \\ & \textbf{se } n < 0 \textbf{ então } produto \leftarrow -x; \\ & \textbf{senão } produto \leftarrow x \,\}\, t. \end{aligned}$$

que junto com (A.5) e mais a regra da composição resulta que

$$(m \in \mathbb{Z} \text{ e } n \in \mathbb{Z})\{S\}(produto = mn).$$

Mais Invariante de laço.

Consideremos o seguinte algoritmo:

Algoritmo 25: fat(n)

```
Dado    : n natural
Devolve: n!
i ← 1;
f ← 1;
enquanto i < n faça
    f ← f * (i + 1);
    i ← i + 1;
devolva f.
```

Vamos provar que a seguinte proposição é um invariante do laço na terceira linha do algoritmo

$$i \geq 1 \text{ e } i \leq n \text{ e } f = i! \tag{A.6}$$

Antes, porém, notemos que o invariante e a negação da condição do laço resulta que quando o laço termina temos $i \geq 1$ e $i \leq n$ e $f = i!$ e não$(i < n)$, ou seja, $i = n$ e $f = n!$, provando a correção parcial do algoritmo.

Vamos provar que a proposição (A.6) é verdadeira independente do número de vezes que o laço é executado. A prova é por indução no número de iterações do `enquanto`.

Suponhamos que a condição do laço vale, ou seja, $i < n$. Antes da primeira iteração (A.6) é verdadeiro pois $i = f = 1! = 1$.

Suponhamos que $i < n$ e que $i \geq 1$ e $i \leq n$ e $f = i!$. Após a execução de $f \leftarrow f * (i+1)$ e após a execução de $i \leftarrow i+1$ temos que para o novo valor de i, $i \geq 1$ e $i \leq n$ e $f = i!$.

Exercícios

Exercício 209. Prove que o seguinte programa está parcialmente correto.

Algoritmo 26: Divisão_inteira(a, d)

Dado : a natural e d natural positivo
Devolve: inteiros q e r tais que $a = dq + r$ e $0 \leq r < d$

$r \leftarrow a$;
$q \leftarrow 0$;
enquanto $r \geq d$ **faça**
    $r \leftarrow r - d$;
    $q \leftarrow q + 1$;
devolva q, r.

Exercício 210. Escreva o argumento para outros comandos de laço como o **para cada**.

A.2 Estruturas de Dados

A.2.1 Heap

Uma árvore binária T é ou a árvore vazia, denotada $\varnothing$ (por abuso de notação) e que não possui vértices, ou é uma terna $T = (r, E, D)$ em que r é um vértice chamado raiz de T, e E e D são árvores binárias disjuntas e que não contêm r, chamadas subarvore esquerda e subarvore direita, respectivamente. A raiz de E, dado que $E \neq \varnothing$, é chamada *filho esquerdo* de r e a raiz de D, dado que $D \neq \varnothing$, é chamada *filho direito* de r. Nos casos $(f, \varnothing, \varnothing)$ chamamos f de *folha*, e os vértices não-folha são ditos *vértices internos*.

Heap é uma estrutura de dados com representação em um vetor V de uma árvore binária quase-completa, isto é, uma árvore binária com altura d na qual todas as folhas estão no nível d ou $d-1$, e se um nó n na árvore tem algum descendente direito no nível d, então todos os descendentes esquerdos de n que forem folhas estão também no nível d. Dado um heap V e um índice i, os índices no vetor do pai, filho esquerdo e filho direito são dados por:

- pai$(i) = \lfloor i/2 \rfloor$
- esq$(i) = 2i$
- dir$(i) = 2i + 1$

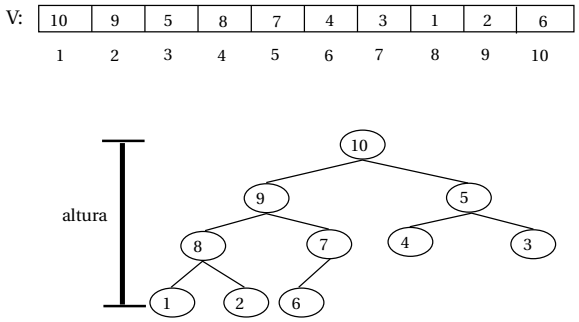


Figura A.1: Exemplo de uma *heap*

Cada nó tem associado um conteúdo que satisfaz apenas um dos seguintes itens, chamado *propriedade da heap*:

- *max-heap*: $V[\text{pai}(i)] \geq V[i]$;
- *min-heap*: $V[\text{pai}(i)] \leq V[i]$.

Exercício 211. Prove que a altura de uma árvore binária quase-completa de n é $\lfloor \log_2(n) \rfloor$.

A seguir daremos alguns algoritmos para *heap* e neles usamos tamHeap(V) para o índice do maior elemento em V que está preenchido com elementos do *heap*; tam(V) é o tamanho do vetor.

Algoritmo 27: FazHeap(V, i)

Dado : dados um vetor V e um índice i de modo que as subarvores esquerda e direita do elemento i já satisfazem a propriedade (max) *heap*

Devolve: V com subarvore de raiz i satisfazendo a propriedade de *heap*

se (esq(i) ≤ tamHeap(V) ***e*** $V[\text{esq}(i)] > V[i]$) **então**
  $maior \leftarrow \text{esq}(i)$;
senão
  $maior \leftarrow i$;
se (dir(i) ≤ tamHeap(V) *e* $V[\text{dir}(i)] > V[maior]$) **então**
  $maior \leftarrow \text{dir}(i)$;
se $maior \neq i$ **então**
  troca o conteúdo de $V[i]$ com $V[maior]$;
FazHeap($V, maior$).

Exercício 212. Mostre que para uma *heap* de tamanho n a complexidade de tempo de FazHeap é $O(\log n)$.

Algoritmo 28: constroiheap(V)

Dado : vetor V

Devolve: V que satisfaz a propriedade de (max) *heap*

tamHeap(V) = tam(V);
para i *de* $\lfloor \text{tamHeap}(V) \rfloor$ *até* 1 **faça**
  FazHeap(V, i).

Observemos que todos os elementos com índice maior que $\lfloor \text{tamHeap}(V) \rfloor$ são folhas, portanto, a iteração só precisa tratar os elementos armazenados nos índices menores que esee.

Exercício 213. Prove que a complexidade de tempo de constroiheap para n elementos é $O(n)$.

Exercício 214. Use o constroiheap para projetoar um algoritmo de ordenação e analise a complexidade desse algoritmo.

Uma das aplicaçõs para o *heap* é a implementação de uma fila de prioridades: uma estrutura para manter um conjunto de elementos S, cada um com um valor associado, chamado de *chave*. Deve prover as seguintes operações:

- insere(S, x): insere o elemento x em S;
- maximo(S): retorna o elemento de S de maior chave;
- extraiMax(S): remove e retorna o elemento de S com maior chave.

Algoritmo 29: max(V)

devolva $V[1]$.

Algoritmo 30: extraiMax(V)

se tamHeap(V) ≥ 1 **então**
    $max \leftarrow V[1]$;
    troca o conteúdo de $V[1]$ como o de $V[\text{tamHeap}(V)]$;
    tamHeap(V) $\leftarrow$ tamHeap(V) $- 1$;
    FazHeap($V, 1$);
    devolva max.

Algoritmo 31: insere(A, k)

tamHeap(V) $\leftarrow$ tamHeap(V) $+ 1$;
$i \leftarrow$ tamHeap(V);
$V[i] \leftarrow k$;
enquanto $i > 1$ ***e*** $V[\text{pai}(i)] < A[i]$ **faça**
    troca o conteúdo de $V[\text{pai}(i)]$ com o de $V[i]$;
    $i \leftarrow$ pai(i);

Observação 7 (Dijkstra). da seguinte forma

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

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

Dijkstra

No caso de fila de prioridades implementada por *heap* binária com elementos de $\overline{S}$ e chave $d(\,)$, o elemento de maior prioridade corresponde ao de menor chave. O procedimento de escolher o vértice com menor valor de $d(\,)$ devolve o elemento de maior prioridade de Q em tempo $O(1)$. O tempo para construir tal *heap* binária é $O(|V|)$, portanto a inicialização do algoritmo toma tempo $O(|V|)$. A remoção na linha 5 tem custo $T_5(u) = O(\log|V|)$ para

todo $u \in V$. Inserção em S pode ser feita em tempo constante, ou seja, $T_6(u) = O(1)$. Agora, $T_{7,8}(u)$ pode ser escrita como

$$T_{7,8}(u) = \sum_{t \in N(u)} T_8(u)$$

e cada vez que a linha 8 de Dijkstra é executada pode ser necessária uma atualização da *heap*, de modo que essa operação toma tempo $O(\log|V|)$. Logo

$$\sum_{u \in V} T_5(u) + T_6(u) + T_{7,8}(u) = \sum_{u \in V} \left(O(\log|V|) + O(1) + \sum_{t \in N(u)} O(\log|V|) \right) = O\big((|V| + |E|)\log|V|\big).$$

Teorema 63. *O algoritmo de Dijkstra sobre $G = (V, E)$ usando uma fila de prioridades implementada por uma heap binária tem complexidade $O\big((|V| + |E|)\log|V|\big)$.*

Exercício 215. Na estimativa de $T_5(u)$ dissemos que $T_5(u) = O(\log(|\overline{S}|))$ e depois superestimamos tomando $T_5(u) = O(\log|V|)$ para todo u. Usando a primeira estimativa, obteríamos

$$\sum_{u \in V} T_5(u) = \sum_{i=1}^{|V|} O(\log i).$$

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

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

Prim

Vejamos como fica usando uma fila de prioridades implementada por uma *heap* binária. O algoritmo usa dois vetores $chave$ e pai, indexados por V e uma fila de prioridades Q:

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

$pai[\,]$ as arestas $\{v, pai(v)\}$ são as arestas escolhidas pelo algoritmo. De início toda posição armazena nil;

Q fila de prioridades indexada por $\overline{U}$ e o vértice de $\overline{U}$ com maior prioridade é o de menor $chave$.

O algoritmo fica assim:

```
1  escolha v ∈ V(G);
2  chave(v) ← 0;
3  U ← ∅;
4  Q ← V;
5  enquanto Ū ≠ ∅ faça
6      u ← extrai mínimo de Ū;
7      para cada t ∈ N(u) faça
8          se t ∈ Ū e ρ({u,t}) < chave[t] então
9              chave[t] ← ρ({u,t});
10             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 $O\big((|V| + |E|)\log|V|\big) = O(|E|\log|V|)$, pois o grafo é conexo.

Teorema 64. *A complexidade do algoritmo* Dijkstra–Jarník–Prim(G) *é* $O(|E(G)|\log|V(G)|)$. □

A.2.2 Estruturas de dados para representar conjuntos disjuntos

Nessa seção veremos estruturas para implementar as operações faz, $busca$ e $união$ 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 faz, $busca$ e $união$; doravante denotamos por m a quantidade de tais operações, e com a hipótese de que *as n primeiras operações são faz*. Isso porque as vezes o tempo de pior caso de uma operação é uma super-estimativa para o custo da maioria das operações.

O problema é representar dinamicamente uma família $\mathscr{S}$ de conjuntos disjuntos $\mathscr{C} = \{C_1, C_2, \ldots, C_k\}$ sobre um universo $\mathscr{U}$ 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 $\mathscr{U} = \{1, 2, \ldots, n\}$.

Essas estruturas devem comportar as operações $cria$, $busca$ e $união$ sobre $\mathscr{C}$:

$cria(x)$ cria o conjunto unitário $C_{k+1} = \{x\}$ em $\mathscr{C}$

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

$união(x, y)$ substitui em $\mathscr{C}$ os conjuntos que contém x e y pela união desses dois conjuntos; se $x \in C_x$ e $y \in C_y$ então $união(x, y)$ remove os conjuntos C_x e C_y de $\mathscr{C}$ e acrescenta $C_x \cup C_y$ a $\mathscr{C}$, além disso escolhe-se um representante para $C_x \cup C_y$.

Representação usando vetor

Usamos um vetor C indexado por $\mathscr{U}$ 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 $C_k = \{x \in \mathscr{U} : C[x] = k\}$. Escolhemos como representante do conjunto o menor elemento. Dessa forma, os algoritmos para as operações $cria$, $busca$ e $união$ ficam da seguinte forma

Algoritmo 32: $cria(x)$

```
1 C[x] ← x.
```

Algoritmo 33: $busca(x)$)

```
1 devolva C[x].
```

Algoritmo 34: $união(x, y)$

```
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;
```

Teorema 65. *O tempo gasto com m operações num universo com n elementos é $O(mn)$.* □

Observamos que o tempo de pior caso não está superestimado como mostra a seguinte seqüência de operações: $cria(1), cria(2) \ldots cria(n)$,

$$\begin{array}{rc} união(2,1) & \text{1 atualização no vetor } C \\ união(3,2) & \text{2 atualizações no vetor } C \\ união(4,3) & \text{3 atualizações no vetor } C \\ união(5,4) & \text{4 atualizações no vetor } C \\ \vdots & \vdots \\ união(n,n-1) & n-1 \text{ atualizações no vetor } C \end{array}$$

cujo custo é $\Theta(nm)$.

Corolário 66. *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.

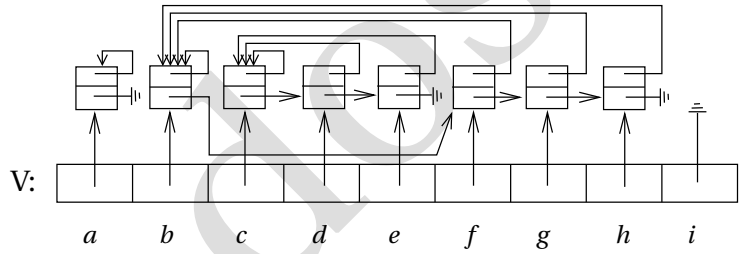


Figura A.2: Representação com listas ligadas dos subconjuntos $\{c,d,e\}$, $\{a\}$ e $\{b,f,g,h\}$ do universo $\{a,b,c,d,e,f,g,h,i\}$.

As funções $cria$ e $busca$ operam ligeiramente diferente da que vimos:

$cria(x)$ 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 67. *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 68. *O representante de um elemento qualquer do universo muda no máximo $\lfloor \lg n \rfloor$ vezes.* □

Teorema 69. *O tempo para m operações num universo com n elementos é $O(m + n \log n)$.*

Demonstração. As operações $cria$ e $busca$ custam $O(m)$ e cada um dos n elementos tem $rep[\,]$ atualizado $O(\log n)$ vezes. □

Corolário 70. *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.3); o pai da raiz é ela mesma. Dessa forma, $cria$ 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.4) e $busca$, que tem tempo dependente da profundidade do elemento na árvore, retorna um ponteiro para uma raiz.

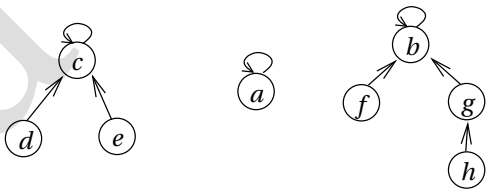


Figura A.3: Uma representação com árvores dos subconjuntos $\{c, d, e\}$, $\{a\}$ e $\{b, f, g, h\}$.

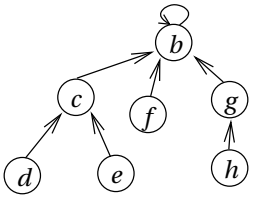


Figura A.4: Resultado de $união(e, f)$.

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.*

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 71. *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 $união(x,y)$ então $t(z) \geq 2^{h(z)}$ para todo z com $busca(z) \notin \{busca(x), busca(y)\}$, 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 72. *O tempo de m operações num universo de tamanho n é $O(m \log n + n)$.*

Demonstração. Cada $cria$ é feito em tempo $O(1)$ e cada $união$ em tempo $O(1)$, resultando em $O(n)$. Cada $busca$ 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 35: $cria(x)$

```
1 pai[x] ← x;
2 t[x] ← 1.
```

Algoritmo 36: $busca(x)$

```
1 enquanto x ≠ pai[x] faça x ← pai[x];
2 ;
3 devolva pai[x].
```

Para descrever a união, usaremos o procedimento $link$ a seguir.

Algoritmo 37: $link(x,y)$

```
1 t[x] ← t[x] + t[y];
2 pai[y] ← x.
```

Algoritmo 38: $união(x,y)$

```
1 se t[busca(x)] ≥ t[busca(y)] então link(busca(x), busca(y));
2 ;
3 senão link(busca(y), busca(x)).;
```

Como o tempo das operações $busca$ são proporcionais a altura parece ser mais natural uma heurística que leva em conta a altura e não o tamanho da árvore.

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 71 continua valendo, e conseqüentemente o corolário. Os algoritmos para $cria$, $busca$ e $união$ são os algoritmos 35, 36 e 38 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 39: $link(x,y)$

```
1 se h[x] = h[y] então h[x] ← h[x] + 1;
2 ;
3 pai[y] ← x.
```

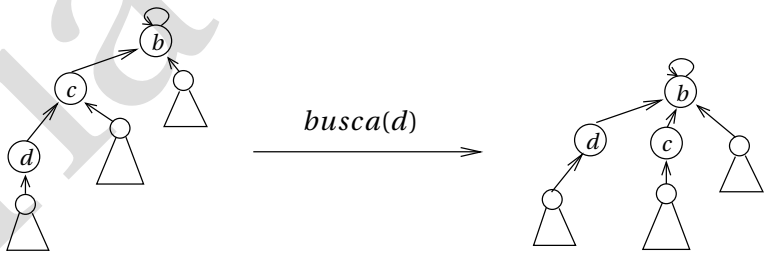
Comparando os tempos obtidos para lista ligada e árvores

$$O(m + n\log n) \times O(m\log n + n)$$

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 19. 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

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 [7]), o resultado que mostraremos a seguir é um pouco menos preciso.

Teorema 73 (Tarjan, 1975)**.** *O tempo gasto com m operações num universo de tamanho n é* $O(m\alpha(m,n))$. □

Algoritmo 40: $busca(x)$

1 **se** $x \neq pai[x]$ **então** $pai[x] \leftarrow busca(pai[x])$;
2 ;
3 devolva $pai[x]$.

Observação 8. 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 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 $cria$. Particionamos os elementos de $\mathscr{U}$ 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) $rank[x] < rank[p[x]]$ para todo $x \in \mathscr{U}$ 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 74. *O tempo gasto com m operações num universo com n elementos é* $O((m+n)\lg^*(n))$.

———————————————————————- 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 4, página 102). Essa representação com as heurísticas mencionadas tem um excelente desempenho assintótico; nesse caso, o número de operações $cria$, $busca$ 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 76) mais $O(|E|\log|E|)$ para a ordenação das arestas; resultando

Corolário 75. *A complexidade de* Kruskal(G) *é* $O(|E|\log|E|)$.

: 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 74 na seção 4.

Teorema 76. *A complexidade de m operações* $cria/busca/união$*, das quais as n primeiras são* $cria$*, onde n é o número de elementos do conjunto universo, é* $O((m+n)\lg^*(n))$. □

———————————————————————-

Exercícios

Exercício 216. 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. [76]

[2] Nicolas Barnier and Pascal Brisset. Graph coloring for air traffic flow management. *Annals of Operations Research*, 130:163–178, 2004. [4]

[3] N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA, 1986. [3]

[4] Béla Bollobás. *Extremal graph theory*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1978. [12]

[5] Preston Briggs, Keith D. Cooper, and Linda Torczon. Improvements to graph coloring register allocation. *ACM Trans. Program. Lang. Syst.*, 16(3):428–455, 1994. [4]

[6] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.*, 9(3):251–280, 1990. [23]

[7] 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. [24, 53, 104]

[8] D. de Werra. An introduction to timetabling. *European Journal of Operational Research*, 19(2):151 – 162, 1985. [4]

[9] I. G. Enting. The combinatorics of algebraic graph theory in theoretical physics. In *Combinatorial mathematics (Proc. Internat. Conf. Combinatorial Theory, Australian Nat. Univ., Canberra, 1977)*, volume 686 of *Lecture Notes in Math.*, pages 148–156. Springer, Berlin, 1978. [4]

[10] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in networks*. Princeton University Press, Princeton, N.J., 1962. [89]

[11] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Comput. Surv.*, 18(1):23–38, 1986. [68]

[12] A. Gamst. Some lower bounds for a class of frequency assignment problems. *Vehicular Technology, IEEE Transactions on*, 35(1):8 – 14, feb. 1986. [4]

[13] 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. [11]

[14] Oded Goldreich, Russell Impagliazzo, Leonid Levin, Ramarathnam Venkatesan, and David Zuckerman. Security preserving amplification of hardness. In *31st Annual Symposium on Foundations of Computer Science, Vol. I, II (St. Louis, MO, 1990)*, pages 318–326. IEEE Comput. Soc. Press, Los Alamitos, CA, 1990. [3]

[15] Rajiv Gupta, Mary Lou Soffa, and Denise Ombres. Efficient register allocation via coloring using clique separators. *ACM Trans. Program. Lang. Syst.*, 16(3):370–386, 1994. [4]

[16] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S.)*, 43(4):439–561 (electronic), 2006. [3]

[17] The Clay Mathematics Institute. Millennium Problems. http://www.claymath.org/millennium, May 2000. Acesso em 03/2008. [5]

[18] N. J. Kalton and James W. Roberts. Uniformly exhaustive submeasures and nearly additive set functions. *Trans. Amer. Math. Soc.*, 278(2):803–816, 1983. [3]

[19] Alfred Lehman. A solution of the shannon switching game. *Journal of the Society for Industrial and Applied Mathematics*, 12(4):687–725, 1964. [51]

[20] 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. [65]

[21] Alexander Lubotzky and Igor Pak. The product replacement algorithm and Kazhdan's property (T). *J. Amer. Math. Soc.*, 14(2):347–363 (electronic), 2001. [3]

[22] Michael Sipser and Daniel A. Spielman. Expander codes. *IEEE Trans. Inform. Theory*, 42(6, part 1):1710–1722, 1996. [3]

[23] Nenad Trinajstić. *Chemical graph theory*. Mathematical Chemistry Series. CRC Press, Boca Raton, FL, second edition, 1992. [4]

[24] 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. [89]

Índice Remissivo

Índice de Símbolos