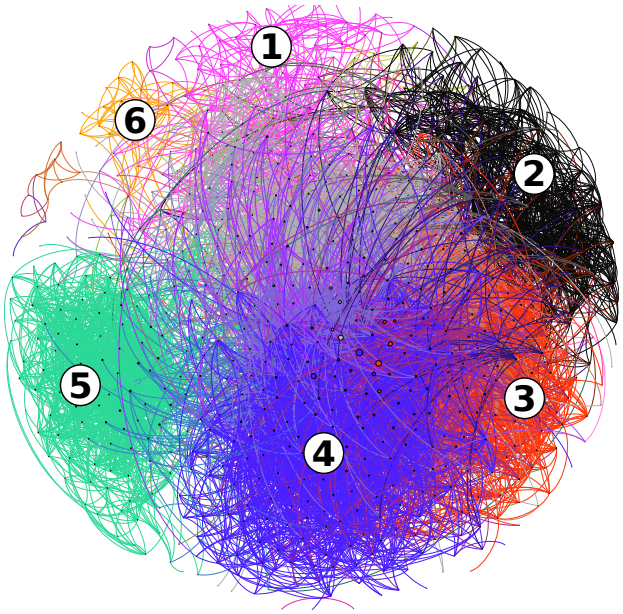
Rafael de Santiago

Anotações para a Disciplina de Grafos

Versão de 28 de março de 2019- 11:33:43



Universidade Federal de Santa Catarina

Sumário

CAPÍTULO 1

Introdução

1.1 Histórico

Uma breve história do passado da Teoria de Grafos (NETTO, 2006):

- 1847: Kirchhoff utilizou modelos de grafos no estudo de circuitos elétricos, criando a teoria de árvores;
- 1857: Cayley usou grafos em química orgânica para enumeração de isômetos dos hidrocarbonetos alifáticos saturados;
- 1859: Hamilton inventou um jogo de buscar um percurso fechado envolvendo todos os vértices de um dodecaedro regular, de tal modo que cada vértice fosse visitado apenas uma vez;
- 1869: Jordan estudou matematicamente as árvores (grafos acíclicos);
- 1878: Sylvester foi o primeiro a utilizar o termo *graph*;
- 1879: Kempe não conseguiu demonstrar a conjectura das 4 cores;
- 1880: Tait falhou ao demonstrar uma prova falsa da conjectura das 4 cores;
- 1890: Haewood provou que a prova de Kempe estava errada e demonstrou uma prova consistente para 5 cores. A de 4 cores só saiu em 1976;

- 1912: Birkhoff definiu os polinômios cromáticos;
- 1926: Menger demonstour um importante teorema sobre o problema de desconexão de itinerários em grafos;
- 1930: Kuratowski encontrou uma condição necessária e suficiente para a planaridade de um grafo;
- 1931: Whitney criou a noção de grafo dual;
- 1936: Primeiro livro sobre grafos foi lançado por König;
- 1941: Brooks enunciou um teorema fornecendo um limite para o número cromático de um grafo;
- 1941: Turán foi o primeiro da teoria extremal dos grafos;
- 1947: Tutte resolveu o problema da existência de uma cobertura minimal em um grafo;
- 1956+: Com as publicações de Ford e Fulkerson, Berge (1957) e Ore (1962), a teoria de grafos passa a receber mais interesse;

1.2 Definições Iniciais

Antes de visitar a representação de grafos, é importante que saibamos o que são vértices e arestas. Vértices geralmente são representados como unidades, elementos ou entidades, enquanto as arestas representam as ligações/conexões entre pares de vértices. Geralmente, chamaremos o conjunto de vértices de V e o conjunto de arestas de E. Define-se que $E \subseteq V \times V$. Também usaremos n e m para denotarem o número de vértices e arestas respectivamente, então $n = |V|$ e $m = |E|$. O número de arestas possível em um grafo é $\frac{n^2-n}{2}$.

Um grafo pode ser representado de duas formas (CORMEN et al., 2012). A primeira forma é chamada de lista de adjacências e tem mais popularidade em artigos científicos.

Nela, o grafo é representado como uma dupla para especificar vértices e arestas. Por exemplo, para um grafo G, pode-se dizer que o mesmo é uma dupla $G = (V, E)$, especificando assim que o grafo G possui um conjunto V de vértices e E de arestas. A segunda forma seria uma através de uma matriz binária, chamada de matriz de adjacência. Normalmente representada pela letra $A(G)$, a matriz é definida por $A(G) = \{0,1\}^{|V|\times|V|}$, a qual seus elementos $a_{u,v} = 1$ se existir uma aresta entre os vértices u e v. Um exemplo das das formas para um mesmo grafo pode ser visualizado no Exemplo 1.2.1.

Example 1.2.1. A Figura 1 exibe um grafo de 4 vértices e 4 arestas. Na representação por listas de adjacências, o grafo pode ser representado da seguinte forma

$$G = (\{1,2,3,4\}, \{\{1,2\},\{1,4\},\{2,4\},\{3,4\}\}). \tag{1.1}$$

A representação por uma matriz de adjacência ficaria assim

$$A(G) = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 1 \\ 2 & 1 & 0 & 0 & 1 \\ 3 & 0 & 0 & 0 & 1 \\ 4 & 1 & 1 & 1 & 0 \end{array}.$$

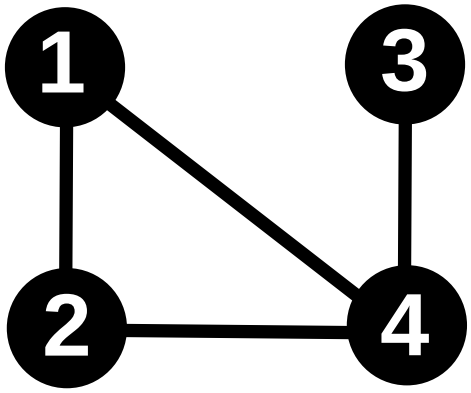


Figura 1 – Exemplo de grafo com 4 vértices e 4 arestas.

1.2.1 Grafos Valorados ou Ponderados

Um grafo é valorado quando um peso ou valor é associado a suas arestas. Na literatura, a definição do grafo passa a ser uma tripla $G = (V, E, w)$, na qual V é o conjunto de vértices, E é o conjunto de arestas e $w : e \in E \rightarrow \mathbb{R}$ é a função que especifica o valor.

Quando não se possui valornas arestas, parte-se de uma relação binária entre existir ou não uma aresta entre dois vértices. Neste caso, se u e v possui uma aresta, geralmente se simboliza essa ligação com o valor 1, e se não existir 0.

Em uma matriz de adjacências para grafos valorados, o valor das arestas aparecem nas células da matriz. Em um par de vértices que não possui valor estabelecido (não há aresta), representa-se com uma lacuna ou com um valor simbólico para o problema que o grafo representa. Por exemplo, se os valores representam as distâncias, geralmente se associa o valor infinito aos pares de vértices que não possuem arestas.

Um exemplo de grafo valorado e suas representações pode ser visualizado no Exemplo 1.2.2.

Example 1.2.2. A Figura 2 exibe um grafo valorado de 4 vértices e 4 arestas. Na representação por listas de adjacências, o grafo pode ser representado da seguinte forma

$$G = (\{1,2,3,4\}, \{\{1,2\},\{1,4\},\{2,4\},\{3,4\}\}, w). \tag{1.2}$$

A função w teria os seguintes valores: $w(\{1,2\}) = 8$, $w(\{1,4\}) = 9$, $w(\{2,4\}) = 5$ e $w(\{3,4\}) = 7$.

A representação por uma matriz de adjacência ficaria assim

$A(G) =$

	1	2	3	4
1	0	8	0	9
2	8	0	0	5
3	0	0	0	7
4	9	5	7	0

ou desta outra forma para o caso de uma aplicação a problemas que envolvam distâncias

$$A(G) = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & \infty & 8 & \infty & 9 \\ 2 & 8 & \infty & \infty & 5 \\ 3 & \infty & \infty & \infty & 7 \\ 4 & 9 & 5 & 7 & \infty \end{array}.$$

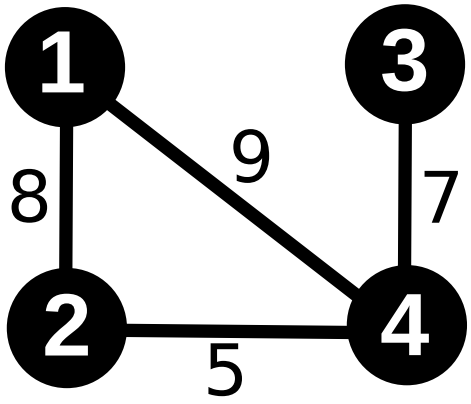


Figura 2 – Exemplo de grafo valorado com 4 vértices e 4 arestas.

Grafos com Sinais

Para representar alguns problemas, utiliza-se valores negativos associados às arestas. Um exemplo disso, seriam grafos que representem relações de amizade e de inimizade. Para amizade, utiliza-se o valor 1 e para inimizade o valor -1. Nesse caso, não dizemos que o grafo é valorado ou ponderado, mas sim um grafo com sinais. Quando os valores negativos e positivos podem ser diferentes de 1 e -1, diz-se que os grafos são valorados e com sinais.

1.2.2 Grafos Orientados

Um grafo orientado é aquele no qual suas arestas possuem direção. Nesse caso, não chamamos mais de arestas e sim de arcos. Um grafo orientado é definido como uma

dupla $G = (V, A)$, a qual V é o conjunto de vértices e A é o conjunto de arcos. O conjunto de arcos é composto por pares ordenados (u, v), os quais $u, v \in V$ e representam um arco saindo de u e incidindo em v. Duas funções importantes devem ser consideradas nesse contexto: a função de arcos saintes $\delta^+(v) = \{(v,u) : (v,u) \in A\}$ e arcos entrantes $\delta^-(v) = \{(u,v) : (u,v) \in A\}$.

O Exemplo 1.2.3 exibe a representação de um grafo orientado.

Example 1.2.3. A Figura 3 exibe um grafo orientado de 4 vértices e 4 arestas. Na representação por listas de adjacências, o grafo pode ser representado da seguinte forma

$$G = (\{1,2,3,4\}, \{(1,4),(2,1),(4,2),(4,3)\}). \tag{1.3}$$

A representação por uma matriz de adjacência ficaria assim

$$A(G) = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{array}.$$

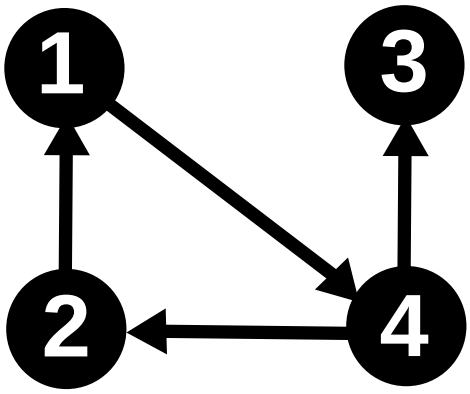


Figura 3 – Exemplo de grafo orientado com 4 vértices e 4 arcos.

1.2.3 Hipergrafo

Um hipergrafo $H = (V, E)$ é um grafo no qual as arestas podem conectar qualquer número de vértices. Cada aresta é chamada de hiperaresta $E \subseteq 2^V \setminus \{\}$.

1.2.4 Multigrafo

Um multigrafo $G = (V, E)$ é um grafo que permite múltiplas arestas para o mesmo par de vértices. Logo, não se tem mais um conjunto de arestas, mas sim uma tupla de arestas. Para o exemplo da Figura 4, têm-se $E = (\{1,2\}, \{1,2\}, \{1,4\}, \{2,4\}, \{3,4\}, \{3,4\})$.

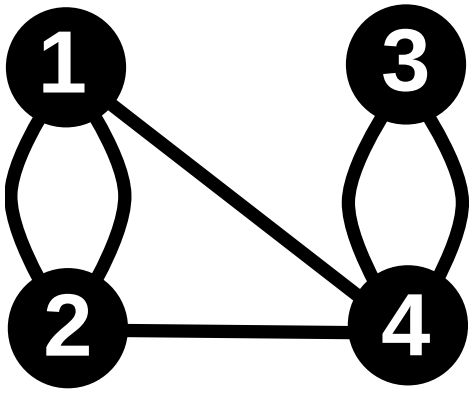


Figura 4 – Exemplo de um multigrafo com 4 vértices e 6 arestas.

1.2.5 Grau de um Vértice

O grau de um vértice é a quantidade de arestas que se conectam a determinado vértice. É denotada por uma função d_v, onde $v \in V$. Em um grafo orientado, o número de arcos saintes para um vértice v é denotado por d_v^+, e o número de arcos entrantes é denotado por d_v^-.

1.2.6 Igualdade e Isomorfismo

Diz-se que dois grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$ são iguais se $V_1 = V_2$ e $E_1 = E_2$. Os dois grafos são considerados isomorfos se existir uma função bijetora (uma-por-uma) para todo $v \in V_1$ e para todo $u \in V_2$ preserve as relações de adjacência (NETTO, 2006).

1.2.7 Partição de Grafos

Uma partição de um grafo é uma divisão disjunta de seu conjunto de vértices. Um grafo $G = (V, E)$ é dito k-partido se existir uma partição $P = \{p_i | i = 1, \ldots, k \wedge \forall j \in \{1, \ldots, k\}, j \neq i (p_i \cap p_j \neq \{\})\}$. Quando $k = 2$, dize que o grafo é bipartido (NETTO, 2006).

1.2.8 Matriz de Incidência

Sobre um grafo orientado $G = (V, E)$, uma matriz de incidência $B(G) = \{+1, -1, \}^{|V| \times |A|}$ mapeia a origem e o destino de cada arco no grafo G. Dado um arco (u, v), $b_{u,(u,v)} = +1$ e $b_{v,(u,v)} = -1$ (NETTO, 2006).

1.2.9 Operações com Grafos

As seguintes operações binárias são descritas em Netto (2006):

- União: Dados os grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$;
- Soma (ou *join*): Dados os grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, $G_1 + G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{\{u, v\} : u \in V_1 \wedge v \in V_2\})$;
- Produto cartesiano: Dados os grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, $G_1 \times G_2 = (V_1 \times V_2, E)$, onde $E = \{\{(v, w), (x, y)\} : (v = x \wedge \{w, y\} \in E_2) \vee (w = y \wedge \{x, y\} \in E_1)\}$. $G_1 \times G_2$ e $G_2 \times G_1$ são isomorfos;
- Composição ou produto lexicográfico: Dados os grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, $G_1 \circ G_2 = (V_1 \times V_2, E)$, onde $E = \{\{(v, w), (x, y)\} : (\{v, x\} \in E_1 \vee v = x) \wedge \{w, y\} \in E_2\}$;
- Soma de arestas: Dados os grafos $G_1 = (V_1, E_1)$ e $G_2 = (V_2, E_2)$, os quais $V_1 = V_2$, $G_1 \oplus G_2 = (V_1, E_1 \cup E_2)$.

A seguinte operação unária é descrita em Netto (2006):

- Contração de dois vértices: Dado um grafo $G = (V, E)$ e dois vértices $u, v \in V$, a operação de contração desses dois vértices em G, gera um grafo $G' = (V', E')$ o qual $V' = V \setminus \{u, v\} \cup \{uv\}$ e $E' = \{\{x, y\} \in E : x \neq u \wedge x \neq v\} \cup \{\{x, uv\} : \{x, u\}, \{x, v\} \in E\}$.

Outras operações sobre grafos são descritas na literatura. Esse texto irá omití-las por enquanto para que sejam utilizados no momento mais oportuno. Sõ elas: inserção e remoção de vértices e arestas, desdobramento de um vértice. Essa última depende do contexto de aplicação.

1.2.10 Vizinhança

A vizinhança de vértices é diferente para grafos não-orientados e orientados. Para um grafo grafo não-orientado $G = (V, E)$, uma função de vizinhança é definida por $N : v \in V \rightarrow \{u \in V : \{v, u\} \in E\}$ e indica o conjunto de todos os vizinhos de um vértice específico. Para o grafo do Exemplo 1.2.1, $N(1) = \{2, 4\}$.

Para um grafo orientado $G = (V, A)$, diz-se que um vértice $u \in V$ é sucessor de $v \in V$ quando $(v, u) \in A$; e $u \in V$ é antecessor de $v \in V$ quando $(u, v) \in A$. As funções de vizinhança para um grafo orientado G são $N^+ : v \in V \rightarrow \{u \in V : (v, u) \in A\}$, $N^- : v \in V \rightarrow \{u \in V : (u, v) \in A\}$, e $N(v) = N^+(v) \cup N^-(v)$.

Diz-se que a vizinhança de v é fechada quando esse mesmo vértice se inclui no conjunto de vizinhos. A função que representa vizinhança fechada v é simbolizada neste texto como $N_*(v) = N(v) \cup \{v\}$.

As funções de vizinhança também podem ser utilizadas para identificar um conjunto de vértices vizinhos de um grupo de vértices em um grafo $G = (V, E)$ (orientado ou não). Nesse contexto, $N(S) = \bigcup_{v \in S} N(v)$, $N^+(S) = \bigcup_{v \in S} N^+(v)$, e $N^-(S) = \bigcup_{v \in S} N^-(v)$.

As noções de sucessor e antecessor podem ser aplicadas iterativamente. As Equações (1.4), (1.5), (1.6) e (1.7) exibem exemplos de fechos transitivos diretos.

$$N^0(v) = \{v\} \tag{1.4}$$

$$N^{+1}(v) = N^{+}(v) \tag{1.5}$$

$$N^{+2}(v) = N^{+}\left(N^{+1}(v)\right) \tag{1.6}$$

$$N^{+n}(v) = N^{+}\left(N^{+(n-1)}(v)\right) \tag{1.7}$$

Chama-se de fecho transitivo direto aqueles que correspondem aos vizinhos sucessivos e os inversos os que correspondem aos vizinhos antecessores. Um fecho transitivo direto de um vértice v de um grafo $G = (V, E)$ são todos os vértices atingíveis a partir v no grafo G; ele é representado pela função $R^{+}(v) = \bigcup_{k=0}^{|V|} N^{+k}(v)$. Um fecho transitivo inverso de v é o conjunto de vértices que atingem v; ele é representado pela função $R^{-}(v) = \bigcup_{k=0}^{|V|} N^{-k}(v)$. Diz-se que w é descendente de v se $w \in R^{+}(v)$. Diz-se que w é ascendente de v se $w \in R^{-}(v)$.

1.2.11 Grafo Regular

Um grafo não-orientado $G = (V, E)$ que tenha $d(v) = k \forall v \in V$ é chamado de grafo k-regular ou de grau k. Um grafo orientado $G_o = (V, A)$ que possui a propriedade $d^{+}(v) = k \forall v \in V$ é chamado de grafo exteriormente regular de semigrau k. Se G_o tiver $d^{-}(v) = k \forall v \in V$ é chamado de grafo interiormente regular de semigrau k.

1.2.12 Grafo Simétrico

Um grafo orientado $G = (V, A)$ é simétrico se $(u, v) \in A \iff (v, u) \in A \forall u, v \in V$.

1.2.13 Grafo Anti-simétrico

Um grafo orientado $G = (V, A)$ é anti-simétrico se $(u, v) \in A \iff (v, u) \notin A \forall u, v \in V$.

1.2.14 Grafo Completo

Um grafo completo $G = (V, E)$ é completo se $E = V \times V$.

Grafos bipartidos completos $G_B = ((X, Y), E)$ possuem $E = X \times Y$.

1.2.15 Grafo Complementar

Para um grafo $G = (V, E)$, um grafo complementar é definido por $G^c = \overline{G} = (V, (V \times V) \setminus E)$.

1.2.16 Percursos em Grafos

"Um percurso, itinerário ou cadeia é uma família de ligações sucessivamente adjacentes, cada uma tendo uma extremidade adjacente a anterior e a outra à subsequente (à execeção da primeira e da última)" (NETTO, 2006). Diz-se que um percurso é aberto quando a última ligação é adjacente a primeira. Têm-se desse modo um ciclo.

Um percurso é considerado simples se não repetir ligações (NETTO, 2006).

Caminhos são cadeias em grafos orientados.

Circuitos são ciclos em grafos orientados.

1.2.17 Cintura e Circunferência

Cintura de um grafo G é comprimendo do menor ciclo existente no grafo. É representada pela função $g(G)$. A circunferência é comprimento do maior ciclo. A circunferência do grafo G é representada pela função $c(G)$.

CAPÍTULO 2

Representações Computacionais

Duas formas de representação computacional de grafos são amplamente utilizadas. São elas "listas de adjacências" e "por matriz de adjacências" (CORMEN et al., 2012). Elas possuem vantagens e desvantagens principamente relacionadas à complexidade computacional (consumo de recursos em tempo e espaço). Detalhes sobre vantagens e desvantagens não aparecerão nesse documento. Um de nossos objetivos do momento será implementar e avaliar as duas formas de representação.

2.1 Lista de Adjacências

A representação de um grafo $G = (V, E)$ por listas de adjacências consiste em um arranjo, chamado aqui de *Adj*. Esse arranjo é composto por $|V|$ listas, e cada posição do arranjo representa as adjacências de um vértice específico (CORMEN et al., 2012). Para cada $\{u, v\} \in E$, têm-se $Adj[u] = (\ldots, v, \ldots)$ e $Adj[v] = (\ldots, u, \ldots)$ quando G for não-dirigido. Quando G for dirigido, para cada $(u, v) \in E$, têm-se $Adj[u] = (\ldots, v, \ldots)$.

Para grafos ponderados, Cormen et al. (2012) sugere o uso da própria estrutura de adjacências para armazenar o peso. Dado um grafo ponderado não-dirigido $G = (V, E, w)$, para cada $\{u, v\} \in E$, têm-se $Adj[u] = \big(\ldots, \big(v, w(\{u, v\})\big), \ldots\big)$ e $Adj[v] = \big(\ldots, \big(u, w(\{u, v\})\big), \ldots\big)$. Quando o grafo for dirigido, para cada $(u, v) \in E$, têm-se $Adj[u] = \big(\ldots, \big(v, w((u, v))\big), \ldots\big)$.

O Algoritmo 1 representa a carga de um grafo dirigido e ponderado $G = (V, A, w)$ em uma lista de adjacências *Adj*.

Algoritmo 1: Criação de uma lista de adjacências para um grafo dirigido e ponderado.

Input : um grafo dirigido e ponderado $G = (V, A, w)$
1 criar arranjo $Adj[|V|]$
2 **foreach** $v \in V$ **do**
3 $\quad Adj[v] \leftarrow$ listaVazia()
4 **foreach** $(u, v) \in A$ **do**
5 $\quad Adj[u] \leftarrow Adj[u] \cup \big(v, w((u, v))\big)$
6 **return** Adj

2.2 Matriz de Adjacências

Uma matriz de adjacência é uma representação de um grafo através de uma matriz A. Para um grafo não-dirigido $G = (V, E)$, $A = \mathbb{B}^{|V|\times|V|}$, na qual cada elemento $a_{u,v} = 1$ e $a_{v,u} = 1$ se $\{u, v\} \in E$; $a_{u,v} = 0$ e $a_{v,u} = 0$ caso $\{u, v\} \notin E$. Para todo grafo não-dirigido G, $a_{u,v} = a_{v,u}$.

Para um grafo dirigido $G = (V, X)$, $A = \mathbb{B}^{|V|\times|V|}$, na qual cada elemento $a_{u,v} = 1$ se $(u, v) \in A$; $a_{u,v} = 0$ e $a_{u,v} = 0$ caso $(u, v) \notin X$.

Para um grafo não-dirigido e ponderado $G = (V, E, w)$, a matriz será formada por células que comportem o tipo de dado representado pelos pesos. Assumindo que os pesos serão números reais, então a matriz de adjacências será $A = \mathbb{R}^{|V|\times|V|}$. Cada elemento $a_{u,v} = w(\{u, v\})$ e $a_{v,u} = w(\{u, v\})$ se $\{u, v\} \in E$; $a_{u,v} = \epsilon$ e $a_{v,u} = \epsilon$ caso $\{u, v\} \notin E$. ϵ é um valor que representa a não conexão, geralmente 0, $+\infty$ ou $-\infty$ dependendo do contexto de aplicação.

O Algoritmo 2 representa a carga de um grafo dirigido e ponderado $G = (V, A, w)$ em uma matriz de adjacências *Adj*.

Algoritmo 2: Criação de uma matriz de adjacências para um grafo dirigido e ponderado.

Input : um grafo dirigido e ponderado $G = (V, A, w : A \to \mathbb{R})$, um símbolo ϵ que representa a não adjacência

```
1 Adj ← ℝ^{|V|×|V|}
2 foreach v ∈ V do
3     foreach u ∈ V do
4         Adj_{u,v} ← ε
5 foreach (u, v) ∈ A do
6     Adj_{u,v} ← w((u, v))
7 return Adj
```

2.3 Exercícios

Implemente as duas bibliotecas para grafos. Preencha a seguinte tabela a partir da análise computacional, de acordo com as operações abaixo determinadas.

	Lista de Adjacências	**Matriz de Adjacências**
Inserção de vértice		
inserção de arestas		
Remoção de vértice		
Remoção de arestas		
Teste se $\{u, v\} \in E$		
Percorrer vizinhos		
Grau de um vértice		

CAPÍTULO 3

Buscas em Grafos

3.1 Busca em Largura

Dado um grafo $G = (V, E)$ e uma origem s, a **busca em largura** (Breadth-First Search - BFS) explora as arestas/arcos de G a partir de s para cada vértice que pode ser atingido a partir de s. É uma exploração por nível. O procedimento descobre as distâncias (número de arestas/arcos) entre s e os demais vértices atingíveis de G. Pode ser aplicado para grafos orientados e não-orientados (CORMEN et al., 2012).

O algoritmo pode produzir uma árvore de busca em largur com raiz s. Nesta árvore, o caminho de s até qualquer outro vértice é um caminho mínimo em número de arestas/arcos (CORMEN et al., 2012).

O Algoritmo 3 descreve as operações realizadas em uma busca em largura. Nele, criam-se três estruturas de dados que serão utilizados para armazenar os resultados da busca. O arranjo C_v é utilizado para determinar se um vértice $v \in V$ foi visitado ou não; D_v determina a distância percorrida até encontrar o vértice $v \in V$; e A_v determina o vértice antecessor ao $v \in V$ em uma busca em largura a partir de s (CORMEN et al., 2012).

Algoritmo 3: Busca em largura.

```
   Input  : um grafo G = (V, E), vértice de origem s ∈ V
   // configurando todos os vértices
1  C_v ← false ∀ v ∈ V
2  D_v ← ∞ ∀ v ∈ V
3  A_v ← null ∀ v ∈ V
   // configurando o vértice de origem
4  C_s ← true
5  D_s ← 0
   // preparando fila de visitas
6  Q ← Fila()
7  Q.enqueue(s)
   // propagação das visitas
8  while Q.empty() = false do
9      u ← Q.dequeue()
10     foreach v ∈ N(u) do
11         if C_v = false then
12             C_v ← true
13             D_v ← D_u + 1
14             A_v ← u
15             Q.enqueue(v)
16 return (D, A)
```

3.1.1 Complexidade da Busca em Largura

O número de operações de enfileiramento e desenfileiramento é limitado a $|V|$ vezes, pois visita-se no máximo $|V|$ vértices. Como as operações de enfileirar e desenfileirar podem ser realizadas em tempo $\Theta(1)$, então para realizar estas operações demanda-se tempo de $O(|V|)$. Deve-se considerar ainda, que muitas arestas/arcos incidem em vértices já visitados, então inclui-se na complexidade de uma BFS a varredura de todas as adjacências, que demandaria $\Theta(|E|)$. Diz-se então, que a complexidade computacional da BFS é $O(|V| + |E|)$.

3.1.2 Propriedades e Provas

3.1.2.1 Caminhos Mínimos

A busca em largura garante a descoberta dos caminhos mínimos em um grafo não-ponderados $G = (V, E)$ de um vértice de origem $s \in V$ para todos os demais atingíveis.

Para demonstrar isso, Cormen et al. (2012) examina algumas propriedades importantes a seguir. Considere a distância de um caminho mínimo $\delta(s,v)$ de s a v como o número mínimo de arestas/arcos necessários para percorrer esse caminho.

Lema 3.1.1. *Seja $G=(V,E)$ um grafo orientado ou não-orientado e seja $s \in V$ um vértice arbitrário, então $\delta(s,v) \leq \delta(s,u)+1$ para qualquer aresta/arco $(u,v) \in E$.*

Prova: Se u pode ser atingido a partir de s, então o mesmo ocorre com v. Desse modo, o caminho mínimo de s para v não pode ser mais longo do que o caminho de s para u seguido pela aresta/arco (u,v) e a desigualdade vale. Se u não pode ser alcançado por s, então $\delta(s,u)=\infty$, e a desigualdade é válida. ■

Lema 3.1.2. *Seja $G=(V,E)$ um grafo orientado ou não-orientado e suponha que G tenha sido submetido ao algoritmo BFS (Algoritmo 3) partindo de um dado vértice de origem $s \in V$. Ao parar, o algoritmo BFS satisfará $D_v \geq \delta(s,v) \forall\, v \in V$.*

Prova: Utiliza-se a indução em relação ao número de operações de enfileiramento (*enqueue*). A hipótese indutiva é $D_v \geq \delta(s,v) \forall\, v \in V$.

A base da indução é a situação imediatamente após s ser enfileirado na linha 7 do Algoritmo 3. A hipótese indutiva se mantém válida nesse momento porque $D_s = 0 = \delta(s,s)$ e $D_v = \infty \geq \delta(s,v) \forall\, v \in V \setminus \{s\}$.

Para o passo da indução, considere um vértice v não-visitado (C_v =**false**) que é descoberto depois do último desempinhamento. Consideramos que o vértice desempunhado é $u \in V$. A hipótese da indução implica que $D_u \geq \delta(s,u)$. Pela atribuição da linha 13 e pelo Lema 3.1.1, obtem-se

$$D_v = D_u + 1 \geq \delta(s,u) + 1 \geq \delta(s,v). \tag{3.1}$$

Então, o vértice v é enfileirado e nunca será enfileirado novamente porque ele também é marcado como visitado e as operações entre as linhas 12 e 15 são apenas executadas para vértices não-visitados. Desse modo, o valor de D_v nunca muda novamente e a hipótese de indução é mantida. ■

Lema 3.1.3. *Suponha que durante a execução do algoritmo de busca em largura (Algoritmo 3) em um grafo $G = (V, E)$, a fila Q contenha os vértices $(v_1, v_2, \ldots, v_r)$, onde v_1 é o início da fila e v_r é o final da fila. Então, $D_{v_r} \leq D_{v_1} + 1$ e $D_{v_i} \leq D_{v_{i+1}}$ para todo $i \in \{1, 2, \ldots, r-1\}$.*

Prova: A prova é realizada por indução relacionada ao número de operações de fila.

Para a base da indução, imediatamente antes do laço de repetição (antes da linha 8), têm-se apenas o vértice s na fila. O lema se mantém nessa condição.

Para o passo da indução, deve-se provar que o lema se mantém para depois do desenfileiramento quanto do enfileiramento de um vértice. Se o início v_1 é desenfileirado, v_2 torna-se o início. Pela hipótese de indução, $D_{v_1} \leq D_{v_2}$. Então $D_{v_r} \leq D_{v_1} + 1 \leq D_{v_2} + 1$. Assim, o lema prossegue com v_2 no início.

Quando enfileira-se um vértice v (linha 15), ele se torna v_{r+1}. Nesse momento, já se removeu da fila o vértice u cujo as adjacências estão sendo analisadas, e pela hipótese de indução, o novo início v_1 deve ter $D_{v_1} \geq D_u$. Assim, $D_{v_{i+1}} = D_v = D_u + 1 \leq D_{v_1} + 1$. Pela hipótese indutiva, têm-se $D_{v_r} \leq D_u + 1$, portanto $D_{v_r} \leq D_u + 1 = D_v = D_{v_{i+1}}$ e o lema se mantém quando um vértice é enfileirado. ■

Corolário 3.1.4. *Suponha que os vértices v_i e v_j sejam enfileirados durante a execução do algoritmo de busca em largura (Algoritmo 3) e que v_i seja enfileirado antes de v_j. Então, $D_{v_i} \leq D_{v_j}$ no momento que v_j é enfileirado.*

Prova: Imediata pelo Lema 3.1.3 e pela propriedade de que cada vértice recebe um valor D finito no máximo uma vez durante a execução do algoritmo. ■

Teorema 3.1.5. *Seja $G = (V, E)$ um grafo orientado ou não-orientado, e suponha que o algoritmo de busca em largura (Algoritmo 3) seja executado em G partindo de um dado vértice $s \in V$. Então, durante sua execução, o algoritmo descobre todo o vértice $v \in V$ atingível por s. Ao findar sua execução, o algoritmo retornará a distância mínima entre s e $v \in V$, então $D_v = \delta(s, v) \forall\, v \in V$.*

Prova: Por contradição, suponha que algum vértice receba um valor d não igual à distância de seu caminho mínimo. Seja v um vértice com $\delta(s,v)$ mínimo que recebe tal valor d incorreto. O vértice v não poderia ser s, pois o algoritmo define $D_s = 0$, o que estaria correto. Então deve-se encontrar um outro $v \neq s$. Pelo Lema 3.1.2, $D_v \geq \delta(s,v)$ e portanto, temos $D_v > \delta(s,v)$. O vértice v deve poder ser visitado a partir de s, se não puder, $\delta(s,v) = \infty \geq D_v$. Seja u o vértice imediatamente anterior a v em um caminho mínimo de s a v, de modo que $\delta(s,v) = \delta(s,u) + 1$. Como $\delta(s,u) < \delta(s,v)$, e em razão de selecionar-se v, têm-se $D_u = \delta(s,u)$. Reunindo essas propriedades, têm-se

$$D_v > \delta(s,v) = \delta(s,u) + 1 = D_u + 1. \tag{3.2}$$

Considere o momento que o algoritmo opta por desenfileirar o vértice u de Q. Nesse momento, o vértice v pode ter sido não-visitado, visitado e está na fila, ou visitado e já foi removido da fila. O restante da prova trabalha em cada um desses casos:

- Se v é não-visitado (C_v =**false**), então a operação na linha 13 define $D_v = D_u + 1$, contradizendo o que é dito na Equação 3.2.
- Se v já foi visitado (C_v =**true**) e foi removido da fila, pelo Corolário 3.1.4, têm-se $D_v \leq D_u$ que também contradiz o que é dito na Equação 3.2.
- Se v já foi visitado e permanece na fila, quando v fora enfileirado w era o vértice antecessor imediato no caminho até v, logo $D_v = D_w + 1$. Considere também que w já foi desenfileirado. Porém, pelo Corolário 3.1.4, $D_w \leq D_u$, então, temos $D_v = D_w + 1 \leq D_u + 1$, contradizendo a Equação 3.2.

■

3.1.2.2 Árvores em Largura

O algoritmo de busca em largura (Algoritmo 3) criar uma árvore de busca em largura à medida que efetua busca no grafo $G = (V,E)$. Também chamada de “subgrafo dos

predecessores", uma árvore de busca em lagura pode ser definida como $G_\pi = (V_\pi, E_\pi)$, na qual $V_\pi = \{v \in V : A_v \neq \textbf{null}\} \cup \{s\}$ e $E_\pi = \{(A_v, \pi, v) : v \in V_\pi \setminus \{s\}\}$.

3.2 Busca em Profundidade

A busca em profundidade (Depth-First Search - DFS) realiza a visita a vértices cada vez mais profundos/distantes de um vértice de origem s até que todos os vértices sejam visitados. Parte-se a busca do vértice mais recentemente descoberto do qual ainda saem arestas inexploradas. Depois que todas as arestas foram visitadas no mesmo caminho, a busca retorna pelo mesmo caminho para passar por arestas inexploradas. Quando não houver mais arestas inexploradas a busca em profundidade pára (CORMEN et al., 2012).

O Algoritmo 4 apresenta um pseudo-código para a busca em profundidade. Note que no lugar de usar uma fila, como na busca em largura (vide Algoritmo 3, utiliza-se uma pilha. Os arranjos C_v, T_v, e A_v $\forall v \in V$ são respectivamente o arranjo de marcação de visitados, do tempo de visita e do vértice antecessor à visita.

Cormen et al. (2012) afirma que é mais comum realizar a busca em profundidade de várias fontes. Desse modo, seu livro reporta um algoritmo que sempre que um subgrafo conexo é completamente buscado, parte-se de um outro vértice de origem não-visitado ainda (um vértice não atingível por s).

3.2.1 Complexidade da Busca em Profundidade

Da mesma maneira que a complexidade da busca em largura, a busca em profundidade possui complexidade $O(|V| + |E|)$. As operações da pilha resultariam tempo $O(|V|)$. Muitos arestas/arcos incidem em vértices já visitados, então inclui-se na complexidade de uma DFS a varredura de todas as adjacências, que demandaria $\Theta(|E|)$.

Algoritmo 4: Busca em profundidade.

Input : um grafo $G = (V, E)$, vértice de origem $s \in V$

```
// configurando todos os vértices
```
1 $C_v \leftarrow$ **false** $\forall v \in V$
2 $T_v \leftarrow \infty \; \forall v \in V$
3 $A_v \leftarrow$ **null** $\forall v \in V$
```
// configurando o vértice de origem
```
4 $C_s \leftarrow$ **true**
5 tempo $\leftarrow 0$
```
// preparando fila de visitas
```
6 $S \leftarrow$ **Pilha()**
7 S.push(s)
```
// propagação das visitas
```
8 **while** $S.empty() =$ ***false*** **do**
9 tempo $\leftarrow$ tempo $+1$
10 $u \leftarrow S$.pop()
11 $T_u \leftarrow$ tempo
12 **foreach** $v \in N(u)$ **do**
13 **if** $C_v =$ ***false*** **then**
14 $C_v \leftarrow$ **true**
15 $A_v \leftarrow u$
16 S.push(v)
17 **return** (C, T, A)

CAPÍTULO 4

Caminhos e Ciclos

Este capítulo tem o objetivo de introduzir o conceito de caminhos e ciclos, e seus principais problemas. Dois problemas clássicos serão definidos e algoritmos para os mesmos, apresentados.

Antes de iniciar a abordar os conteúdos deste capítulo, é importante entender o que é um caminho e um ciclo, para estabelecer suas diferenças no contexto de grafos. Um caminho[1] é uma sequência de vértices $\langle v_1, v2, \ldots v_n \rangle$ conectados por uma aresta ou arco. Gross e Yellen (2006) definem um caminho como um grafo com dois vértices com grau 1 e os demais vértices com grau 2, formando uma estrutura linear. Um ciclo (ou circuito)[2] é uma cadeia fechada de vértices $\langle v_1, v_2, \ldots, v_n, v_1 \rangle$ onde cada par consecutivo é conectado por uma aresta ou arco. É como um caminho com o fim e o início conectados.

4.1 Caminhos e Ciclos Eulerianos

Dado um grafo orientado ou não orientado $G = (V, E)$, um caminho Euleriano é uma "trilha" ou seja, uma sequência de arestas/arcos onde cada aresta/arco é visitada(o) uma

[1] Em inglês, chamado de *path*.
[2] Em inglês, chamado de *cycle* ou *circuit*.

única vez. O ciclo Euleriano é semelhante ao caminho, com exceção de que começa e termina na mesma aresta/arco. Um grafo é dito Euleriano se possui um ciclo Euleriano.

Os problemas de caminho e ciclo Euleriano surgiram com o conhecido problema das Sete Pontes de Königsberg por Euler em 1736. O problema consistia em atravessar as todas as sete pontes da cidade de Königsberg da Prussia (hoje Kaliningrado na Rússia) sem repetí-las.

Desafio

Observando um mapa antigo das sete pontes, você consegue determinar o caminho Euleriano?

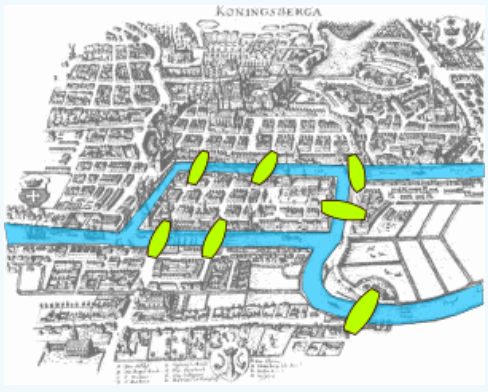


Figura 5 – Mapa das sete pontes de Königsberg na época de in Euler.

4.1.1 Algoritmo de Hierholzer

O algoritmo de Hierholzer (Algoritmo 5) foi desenvolvido em 1873. Ele identifica o ciclo Euleriano em tempo $O(|E|)$.

Algoritmo 5: Algoritmo de Hierholzer.

Input : um grafo $G = (V, E)$

```
1  foreach e ∈ E do
2      C_e ← false
3  v ← selecionar um v ∈ V arbitrariamente
   // "buscarSubcicloEuleriano" invoca o Algoritmo 6
4  (r, Ciclo) ← buscarSubcicloEuleriano(G, v, C)
5  if r = false then
6      return (false,null)
7  else
8      if ∃e ∈ E : C_e = false then
9          return (false,null)
10     else
11         return (true, Ciclo)
```

Algoritmo 6: Algoritmo de Auxiliar "$buscarSubcicloEuleriano$".

Input : um grafo $G = (V, E)$, um vértice $v \in V$, um vetor C

```
1  Ciclo ← ()
2  t ← v
3  repeat
4      if ∄{v, u} ∈ E : C_e = false then
5          return (false, null)
6      else
7          {v, u} ← selecionar uma aresta e ∈ E tal que C_e = false
8          C_{v,u} ← true
9          v ← u
10 until v ≠ t
11 foreach x ∈ {u ∈ Ciclo : ∃{u, w} ∈ {e ∈ E : C_e = false}} do
12     (r, Ciclo') ← buscarSubcicloEuleriano(G, v, C)
13     if r = false then
14         return (false, null)
15     Assumindo que Ciclo = ⟨v_1, v_2, ..., x, ..., v_1⟩ e Ciclo' = ⟨x, u_1, u_2, ..., u_k, x⟩, alterar
       Ciclo para Ciclo = ⟨v_1, v_2, ..., x, u_1, u_2, ..., u_k, x, ..., v_1⟩, ou seja, inserir o Ciclo' no
       lugar da posição de x em Ciclo.
16 return (true, Ciclo)
```

Desafio

Explique porque a complexidade de Algoritmo de Hierholzer é de $O(|E|)$.

Teorema 4.1.1. *Um grafo não-orientado $G = (V, E)$ é (ou possui um ciclo) Euleriano se e somente se G é conectado e cada vértice tem um grau par.*

Prova: Para o grafo conectado G, para todo $m \geq 0$, considere $S(m)$ ser a hipótese de que se G têm m arestas e todos os graus dos vértices forem pares, então G é Euleriano.

Vamos a prova por indução.

A base da indução é o $S(0)$. Nessa hipótese, G não tem arestas, então para todo $v \in V$, $d_v = 0$. Como zero é par G é trivialmente Euleriano.

O passo da indução implica que as hipoteses $S(0) \wedge S(1) \wedge \ldots \wedge S(k-1) \implies S(k)$. Suponha um $k \geq 1$ e assuma que $S(1) \wedge \ldots \wedge S(k+1)$ é verdade. Precisa-se provar que $S(k)$ é verdade. Suponha que G tenha k arestas, é conectado e possui somente vértices com valor de grau par.

- Desde que G é um grafo conectado e possui vértices com grau par, o menor grau é 2. Então esse grafo G precisa ter um ciclo C.
- Suponha um novo grafo H gerado a partir de G sem as arestas que estão no ciclo C. Note que H pode estar desconectado. Pode-se dizer que H é a união dos componentes conectados $H_1, H_2, \ldots H_t$. O grau dos vértices cada H_i precisa ser par.
- Aplicando a hipótese de indução a cada H_i, que é $S(|E(H_1)|), \ldots, S(|E(H_t)|)$, cada H_i terá um ciclo Euleriano C_i.
- Pode-se criar um circuito Euleriano para G por dividir o ciclo C em ciclos C_i. Primeiro, comece em qualquer vértice em C_i e percorra até atingir outro H_i. Então, percorra C_i e volte ao C até atingir o próximo H_i.

Finalmente, G precisa ser Euleriano. Isso completa o passo da indução como $S(0) \wedge S(1) \wedge \ldots \wedge S(k-1) \implies S(k)$. Por esse princípio, para $m \geq 0$, $S(m)$ é verdadeiro. ■

4.2 Caminhos e Ciclos Hamiltonianos

Ciclos ou caminhos Hamiltonianos são aqueles que percorrem todos os vértices de um grafo apenas uma vez. Mais especificamente para um ciclo Hamiltoniano, o início e o fim terminam no mesmo vértice. O nome Hamiltoniano vem de William Rowan Hamilton, o inventor de um jogo que desafia a buscar um ciclo pelas arestas de dodecaedro (figura tridimensional de 12 faces).

Um grafo é dito Hamiltoniano se possui um ciclo Hamiltoniano.

Há $|V|!$ diferentes sequências de vértices que podem ser caminhos Hamiltonianos, então, um algoritmo de força-bruta demanda muito tempo computacional. O problema de decisão para encontrar um caminho ou ciclo Hamiltoniano é considerado NP-Completo.

4.2.1 Caixeiro Viajante

Dado um grafo completo[3] $G = (V, E, w)$ no qual V é o conjunto de vértices, E é o conjunto de arestas e $w : E \to \mathbb{R}^+$ é a função dos pesos (ou custo ou distâncias), busca-se pelo ciclo Hamiltoniano de menor soma total de peso (menor custo ou distância).

Um dos algoritmos mais eficientes para resolvê-lo é o de programação dinâmica Held-Karp (ou Bellman–Held–Karp, no Algoritmo 7). No entanto, o mesmo demanda tempo computacional de $O(2^{|V|}|V|^2)$.

Algoritmo 7: Algoritmo de Bellman-Held-Karp.

Input : um grafo $G = (V, E = V \times V, w)$

```
for k ← 2 to |V| do
    C({k}, k) ← w((1, k))
for s ← 2 to |V| − 1 do
    foreach S ∈ {x ∈ {2,3,...,|V|} : |x| = s} do
        foreach v ∈ S do
            C(S, v) ← min_{u≠v, u∈S} {C(S\{v}, u) + w((u, v))}
return min_{v∈V\{1}} {C({2,3,...,|V|}, v) + w((v, 1))}
```

Desafio

Execute o Algoritmo 7 sobre o grafo $G = (V = \{1,2,3,4\}, E = V \times V, w)$, no qual $w(\{1,2\}) \to 10$, $w(\{1,3\}) \to 15$, $w(\{1,4\}) \to 20$, $w(\{2,3\}) \to 35$, $w(\{2,4\}) \to 25$ e $w(\{3,4\}) \to 30$.

A resposta deve ser 80.

[3] Em um grafo completo, o conjunto de arestas é definido por $E = V \times V$.

CAPÍTULO 5

Caminhos Mínimos

Em um problema de Caminho Mínimo, há um grafo ponderado orientado ou não $G = (V, E, w)$, onde V é o conjunto de vértices, E é o conjunto de arcos ou arestas, e $w : E \to \mathbb{R}$ é a função que representa o peso entre dois vértices (distância ou custo das arestas). Para um caminho $p = \langle v_0, v_1, \ldots, v_k \rangle$ seu peso é dado por $w(p) = \sum_{i=1}^{k} w\big((v_{i-1}, v_i)\big)$ (CORMEN et al., 2012).

O peso de um caminho mínimo de u a v é dado por

$$\delta(u,v) = \begin{cases} \textbf{min}\{w(p) : u \overset{p}{\rightsquigarrow} v\}, & \text{se há um caminho de } u \text{ para } v\,, \\ \infty, & \text{caso contrário.} \end{cases} \tag{5.1}$$

Há algumas variantes para os problemas de caminho mínimo (CORMEN et al., 2012):

- Problema de caminhos mínimos de fonte única: dado um grafo ponderado $G = (V, E, w)$ e um vértice de origem $s \in V$, encontrar o caminho de custo $\delta(s, v)$ para todo o $v \in V$;

- Problema de caminhos mínimos para um destino:dado um grafo ponderado $G = (V, E, w)$, um vértice de destino $t \in V$, determinar o caminho de custo $\delta(v, t)$ para todo o $v \in V$;

- Problema de caminhos mínimos para um par: dado um grafo ponderado $G = (V, E, w)$, um vértice de origem $s \in V$ e um vértice de destino t, determinar o caminho de custo $\delta(s, t)$;

- Problema de caminhos mínimos para todos os pares: dado um grafo ponderado $G = (V, E, w)$, encontrar o caminho de curso $\delta(u, v)$ para todo o par $u, v \in V$.

Pesos Negativos

Os problemas de caminho mínimo geralmente operam sem erros em grafos com pesos negativos. Uma exceção a isso é quando há um ciclo com peso negativo. Nesse caso, nunca haverá um peso definido, pois é sempre possível diminuir o peso total do caminho percorrendo o ciclo mais uma vez.

Inicialização e Relaxamento

Algoritmo 8: Inicialização de G.

Input : um grafo $G = (V, E, w)$, um vértice de origem $s \in V$

```
// inicialização
```

1 $D_v \leftarrow \infty \; \forall v \in V$

2 $A_v \leftarrow$ **null** $\forall v \in V$

3 $D_s \leftarrow 0$

4 **return** (D, A)

Algoritmo 9: Relaxamento de v.

Input : um grafo $G = (V, E, w)$, $(u, v) \in E$, A, D

1 **if** $D_v > D_u + w\big((u, v)\big)$ **then**

2 $\quad D_v \leftarrow D_u + w\big((u, v)\big)$

3 $\quad A_v \leftarrow u$

4 **return** D_v, A_v

5.1 Propriedades de Caminhos Mínimos

Propriedade de subcaminhos de caminhos mínimos o são

Lema 5.1.1. *Dado um grafo ponderado $G = (V, E, w)$, um caminho mínimo entre v_1 e v_k $p = \langle v_1, v_2, \ldots, v_k \rangle$, suponha que para quaisquer i e j, $1 \leq i \leq j \leq k$. Todo o subcaminho de p chamado de $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ é um caminho mínimo de v_i a v_j.*

Prova: Se o caminho p for decomposto em $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, têm-se $w(p) = w(p_{0j}) + w(p_{ij}) + w(p_{jk})$. Suponha que exista um caminho p'_{ij} de v_i a v_j com peso $w(p'_{ij}) < w(p_{ij})$. Então, $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ é um caminho de v_1 a v_k cujo o peso $w(p) = w(p_{0j}) + w(p'_{ij}) + w(p_{jk})$ é menor do que $w(p)$, o que contradiz a hipótese de que p seja um caminho mínimo de v_1 a v_k. ■

Propriedade de desigualdade triangular

Lema 5.1.2. *Seja $G = (V, E, w)$ um grafo ponderado e $s \in V$ um vértice de origem, então para todas as arcos/arestas $(u, v) \in E$ tê-se*

$$\delta(s, v) \leq \delta(s, u) + w\big((u, v)\big). \tag{5.2}$$

Prova: Suponha que p seja um caminho entre s e v. Então, p não tem peso maior do que qualquer outro caminho de s a v. Especificamente, p não possui peso maior que o caminho de s até o vértice u que utiliza a aresta/arco (u, v) para atingir o destino v. ■

Propriedade de limite superior

Lema 5.1.3. *Seja $G = (V, E, w)$ um grafo ponderado dirigido ou não com a função de peso $w : E \to \mathbb{R}$. Seja $s \in V$ o vértice de origem, considera-se também o grafo G inicializado (Algoritmo 8). Então, $D_v \geq \delta(s, v)$ para todo $v \in V$, e esse invariante é mantido para qualquer sequência de etapas de relaxamentos em G (Algoritmo 9). Além disso, tão logo D_v alcance seu limite inferior $\delta(s, v)$, nunca mais se altera.*

Prova: Prova-se que o invariante $D_v \geq \delta(s,v)$ para todo o vértice $v \in V$ por indução em relação ao número de etapas de relaxamento.

Para a base da indução, $D_v \geq \delta(s,v)$ é verdadeiro após a inicialização (Algoritmo 8), pois esse procedimento define que $D_v = \infty$ para todo $v \in V \setminus \{s\}$, ou seja, $D_v \geq \delta(s,v)$ mesmo que v seja inatingível em um caminho mínimo a partir de s. Nesse momento $D_s = 0 \geq \delta(s,s)$, observando que $\delta(s,s) = \infty$ caso s participa de um ciclo negativo.

Para o passo da indução, considere o relaxamento de uma aresta/arco (u,v). Pela hipótese de indução, $D_x \geq \delta(s,x)$ para todo o $x \in V$ antes do relaxamento. O único valor de D que pode mudar é D_v. Se ele mudar, têm-se

$$\begin{aligned} D_v &= D_u + w\big((u,v)\big) \\ &\geq \delta(s,u) + w\big((u,v)\big) \text{ (pela hipótese da indução)} \\ &\geq \delta(s,v) \text{ (pela desigualdade triangular,} \\ &\qquad \text{Lema 5.1.2)} \end{aligned} \tag{5.3}$$

e, portanto o invariante é mantido.

Para demonstrar que D_v não se altera depois que $D_v = \delta(s,v)$, por ter alcaçado seu limite inferior, D_v não pode diminuir porque $D_v \geq \delta(s,v)$ e não pode aumentar porque o relaxamento não aumenta valores de D. ■

Propriedade de inexistência de caminho

Corolário 5.1.4. *Supõe-se que, em um grafo $G = (V,E,w)$ ponderado dirigido ou não, nenhum caminho conecte o vértice de origem $s \in V$ a um vértice $v \in V$. Então, depois que o grafo G é inicializado (Algoritmo 8), temos $D_v = \delta(s,v) = \infty$ e essa desigualdade é mantida como um invariante para qualquer sequência de etapas de relaxamento (Algoritmo 9) nas arestas de G;*

Prova: Pela propriedade de limite superior (Lema 5.1.3), têm-se sempre $\infty = \delta(s,v) \leq D_v$, portanto $D_v = \infty = \delta(s,v)$. ■

Propriedade de convergência

Lema 5.1.5. *Seja $G = (V, E, w)$ um grafo ponderado dirigido ou não, $s \in V$ um vértice de origem e $s \rightsquigarrow u \rightarrow v$ um caminho mínimo de s a v em G. Suponha que G seja inicializado (Algoritmo 8) e depois uma sequência de etapas de relaxamento (Algoritmo 9) é executado para todas as arestas/arcos de G. Se $D_u = \delta(s, u)$ em qualquer tempo anterior da chamada, então $D_u = \delta(s, u)$ igual em toda a chamada.*

Prova: Pela propriedade do limite superior (Lema 5.1.3), se $D_u = \delta(s, u)$ em algum momento antes do relacamento da aresta/arco (u, v), então essa igualdade se mantém válida a partir de sua definição. Em particular, após o relaxamento (Algoritmo 9) da aresta/arco (u, v), têm-se:

$$\begin{aligned} D_v &\leq D_u + w\big((u, v)\big) \text{ pelo Corolário 5.1.4)} \\ &= \delta(s, u) + w\big((u, v)\big) \\ &= \delta(s, v) \text{ pelo Lema 5.1.1.)} \end{aligned} \tag{5.4}$$

Pela propriedade do limite superior (Lema 5.1.3) $D_v \geq \delta(s, v)$, da qual concluímos que $D_v = \delta(s, v)$, e essa igualdade é mantida daí em diante. ■

Propriedade de relaxamento de caminho

Lema 5.1.6. *Seja $G = (V, E, w)$ um grafo ponderado dirigido ou não e $s \in V$ um vértice de origem. Considere qualquer caminho mínimo $p = \langle v_1, v_2, \ldots v_k \rangle$ de $s = v_1$ a v_k. Se G é inicializado (Algoritmo 8) e depois ocorre uma sequência de etapas de relaxamento (Algoritmo 9) que inclui, pela ordem, relaxar as arestas/arcos $(v_1, v_2), (v_2, v_3), \ldots (v_{k-1}, v_k)$, então $D_k = \delta(s, v_k)$ depois desses relaxamentos e todas as vezes daí em diante. Essa propriedade se mantém válida, não importa quais outros relaxamentos forem realizados.*

Prova: Esta prova é realizada por indução, na qual têm-se $D_{v_i} = \delta(s, v_i)$ depois que o i-ésimo aresta/arco do caminho p é relaxado.

Para a base, $i = 1$ antes que quaisquer arestas/arcos em p sejam relaxados. Têm-se $D_{v_1} = D_s = 0 = \delta(s,s)$ pela inicialização. Pela propriedade do limite superior (Lema 5.1.3) o valor D_s nunca se altera depois da inicialização.

Pelo passo da indução, supõe-se que $v_{i-1} = \delta(s, v_{i-1})$ e examina-se o que acontece quando se relaxa a aresta (v_{i-1}, v_i). Pela propriedade de convergência (Lema 5.1.5), após o relaxamento dessa aresta, têm-se $D_v = \delta(s, v_i)$ e essa igualdade é mantida todas as vezes depois disso. ■

Lema 5.1.7.

Prova: ■

5.2 Bellman-Ford

O algoritmo de Bellman-Ford resolve o problema de caminhos minimos de uma única fonte. Um pseudo-código está representado no Algoritmo 10. Como entrada para o algoritmo deve-se determinar um grafo ponderado orientado ou não $G = (V, E, w)$, onde V é o conjunto de vértices, E o conjunto de arestas/arcos e $w : E \to \mathbb{R}$, e um vértice de origem $s \in V$. O algoritmo devolve um valor booleano **false** quando não foi encontrado um ciclo de peso negativo em G. Caso contrário, retorna **true**, o antecessor de cada vértice v no caminho mínimo em A_v e a peso $\delta(s, v)$ em D_v.

O algoritmo vai progressivamente diminuindo a estimativa de peso do caminho de

s a $v \in V$ até que se obtenha o caminho mínimo e $D_v = \delta(s, v)$ para todo $v \in V$.

Algoritmo 10: Algoritmo de Bellman-Ford.

Input : um grafo $G = (V, E, w)$, um vértice de origem $s \in V$

```
// inicialização
```

1 $D_v \leftarrow \infty \; \forall v \in V$

2 $A_v \leftarrow$ **null** $\forall v \in V$

3 $D_s \leftarrow 0$

4 **for** $i \leftarrow 1$ ***to*** $|V| - 1$ **do**

5 **foreach** $(u, v) \in E$ **do**

```
// relaxamento
```

6 **if** $D_v > D_u + w\big((u, v)\big)$ **then**

7 $D_v \leftarrow D_u + w\big((u, v)\big)$

8 $A_v \leftarrow u$

9 **foreach** $(u, v) \in E$ **do**

10 **if** $D_v > D_u + w\big((u, v)\big)$ **then**

11 **return** (**false**,**null**,**null**)

12 **return** (**true**, D, A)

5.2.1 Complexidade de Bellman-Ford

Quanto a complexidade computacional em tempo computacional de Bellman-Ford, observando as primeiras instruções, têm-se a inicialização que demanda $\Theta(|V|)$ pois as estruturas são inicializadas para cada vértice. A partir do primeiro conjunto de laços de repetição, há o laço mais externo que repete $|V| - 1$ vezes. Para cada repetição desse laço, passa-se por cada aresta em E, logo esse primeiro conjunto de laços dita $(|V| - 1)|E|$ execuções da comparação na linha 6. O último laço de repetição, faz ao máximo $|E|$ verificações da comparação na linha 10. Então, o algoritmo de Bellman-Ford é executado no tempo computacional de $O(|V||E|)$.

5.2.2 Corretude de Bellman-Ford

Lema 5.2.1. *Seja $G = (V, E, w)$ um grafo ponderado e um vértice de origem $s \in V$, suponha que G não possua nenhum ciclo de peso negativo que possa ser alcançado por s. Então, depois de executar as $|V| - 1$ iterações nas linhas 4 a 8 com o algoritmo de Bellman-Ford (Algoritmo 10), têm-se $D_v = \delta(s, v)$ para todo $v \in V$.*

Prova: Prova-se o esse lema através da propriedade de relaxamento de caminho (Lema 5.1.6). Considera-se que qualquer vértice v possa ser atingido por s e seja $p = \langle v_1, v_2, \ldots, v_k \rangle$ um caminho mínimo de s a v, no qual $v_1 = s$ e $v_k = v$. Como caminhos mínimos são simpes, p tem no máximo $|V| - 1$ arestas/arcos, sendo $k \leq |V| - 1$. Cada uma das $|V| - 1$ iterações do laço da linha 4 relaxa todas as $|E|$ arestas/arcos. Entre as arestas relaxadas na i-ésima iteração, para $i = 1, 2, \ldots k$, está (v_{i-1}, v_i). Então, pela propriedade de relaxamento de caminho $D_v = D_{v_k} = \delta(s, v_k) = \delta(s, v)$. ■

Corolário 5.2.2. *Seja $G = (V, E, w)$ um grafo ponderado dirigido ou não e $s \in V$ o vértice de origem, supõe-se que G não tenha nenhum ciclo negativo que possa ser atingido por s. Então, para cada vértice $v \in V$, existe um caminho de s a v sse o algoritmo de Bellman-Ford termina com $D_v < \infty$ quando é executado para G e s.*

Prova: Se $v \in V$ pode ser atingido por s, então existe uma aresta/arco (u, v). Então, $\delta(s, v) < \infty$ através da propriedade de convergência (Lema 5.1.5). ■

Teorema 5.2.3. *Considera-se o algoritmo de Bellman-Ford (Algoritmo 10) executado para um grafo $G = (V, E, w)$ (((CONTINUAR)))*

Prova:

■

5.3 Dijkstra

Algoritmo 11: Algoritmo de Dijkstra.

Input : um grafo $G = (V, E)$, um vértice de origem $s \in V$

$D_v \leftarrow \infty \; \forall v \in V$

$A_v \leftarrow \textbf{null} \; \forall v \in V$

$C_v \leftarrow \textbf{false} \; \forall v \in V$

$D_s \leftarrow 0$

for $\exists v \in V(C_v =$ ***false***$)$ **do**

  $u \leftarrow \textbf{arg min}_{v \in V}\{D_v | C_v =$**false**$\}$

  $C_u \leftarrow \textbf{true}$

  foreach $v \in \{v \in V | (u, v) \in A \wedge C_v =$***false***$\}$ **do**

    if $D_v > D_u + w\big((u, v)\big)$ **then**

      $D_v \leftarrow D_u + w\big((u, v)\big) \; A_v \leftarrow u$

return (D, A)

5.4 Floyd-Warshall

$$W\Big(G(V, E, w)\Big)_{uv} = \begin{cases} 0, & \text{se } u = v, \\ w\big((u, v)\big), & \text{se } u \neq v \wedge (u, v) \in E, \\ \infty, & \text{se } u \neq v \wedge (u, v) \notin E. \end{cases} \tag{5.5}$$

Algoritmo 12: Algoritmo de Floyd-Warshall.

Input : um grafo $G = (V, E, w)$

$D^{(0)} \leftarrow W\Big(G\Big)$

foreach $k \in V$ **do**

  seja $D^{(k)} = (d_{uv}^{(k)})$ uma nova matriz $|V| \times |V|$

  foreach $u \in V$ **do**

    foreach $v \in V$ **do**

      $d_{uv}^{(k)} \leftarrow \mathbf{min}\Big\{d_{uv}^{(k-1)}, d_{uk}^{(k-1)} + d_{kv}^{(k-1)}\Big\}$

return $D^{(|V|)}$

Agradecimentos

Referências

CORMEN, T. H. et al. *Algoritmos: teoria e prática*. Rio de Janeiro: Elsevier, 2012. Citado 6 vezes nas páginas 4, 15, 19, 21, 24 e 33.

GROSS, J. T.; YELLEN, J. *Graph Theory and Its Applications*. FL: CRC Press, 2006. Citado na página 27.

NETTO, P. O. B. *Grafos: teoria, modelos, algoritmos*. São Paulo: Edgard Blucher, 2006. Citado 4 vezes nas páginas 3, 9, 10 e 13.

APÊNDICE A

Revisão de Matemática Discreta

Conjuntos é uma coleção de elementos sem repetição em que a sequência não importa. No Brasil, utilizamos a seguinte notação para enumerar todos os elementos de um conjunto. Na Equação (A.1), é possível visualizar a representação de um conjunto denominado A, formado pelos elementos e_1, e_2, ..., e_n. Devido ao uso da vírgula como separador de decimais, usa-se formalmente o ponto-e-vírgula. Para essa disciplina, podemos utilizar a vírgula como o separador de elementos em um conjunto, desde que utilizados o ponto como separador de decimais[1]. Para dar nome a um conjunto, geralmente utiliza-se uma letra maiúscula ou uma palavra com a inicial em maiúscula.

$$A = \{e_1; e_2; \ldots; e_n\} \tag{A.1}$$

Há duas formas de definir conjuntos. A forma por enumeração por elementos, utiliza notação semelhante a da Equação (A.1). São exemplos de definição de conjuntos por enumeração:

- $N = \{\diamondsuit, \spadesuit, \heartsuit, \clubsuit\}$;
- $V = \{a, e, i, o, u\}$;
- $G = \{\alpha, \beta, \gamma, \delta, \epsilon, \zeta, \eta, \theta, \iota, \kappa, \lambda, \mu, \nu, \xi, \pi, \rho, \sigma, \tau, \upsilon, \phi, \chi, \psi, \omega\}$;
- $R = \{-100.9, 12.432, 15.0\}$;
- $D = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

A forma por descrição de propriedades utiliza-se de uma notação que evidencia a natureza de cada elemento pela descrição de um em um formato genérico. Por exemplo o conjunto D, descrito na Equação (A.2), denota um conjunto com os mesmos elementos em $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

$$D = \{x \in \mathbb{Z} | x > 1 \wedge x \leq 10\} \tag{A.2}$$

. Então para que complicar utilizando uma notação não enumerativa? Por dois motivos: por questões de simplicidade, dado a quantidade de conjuntos; ou para representar conjuntos infinitos, como no exemplo dos inteiros pares $Pares = \{x \in \mathbb{Z} | x \equiv 0 (\mod 2)\}$.

[1] Nas anotações presentes nesse documento, utiliza-se a "notação americana". Para a Equação (A.1) , teria-se $A = \{e_1, e_2, \ldots, e_n\}$.

Para o conjunto dos pares, ainda podemos utilizar uma descrição mais informal, mas que é dependente da conhecimento sobre a linguagem Portuguesa: *Pares* = $\{x \in \mathbb{Z} |\ x$ *é inteiro e par* $\}$.

Para denotar a cardinalidade (quantidade de elementos) de um conjunto, utilizamos o símbolo "|". Para os conjuntos apresentados acima, é correto afirmar que:

- $|N| = 4$;
- $|V| = 5$;
- $|R| = 3$;
- $|D| = 10$;
- $|Pares| = \infty$.

A cardinalidade pode ser utilizada para identificar quantos símbolos são necessários para representar um elemento. Por exemplo, $|12,66| = 5$

Para denotar conjuntos vazios, adota-se duas formas de representação: $\{\}$ ou $\emptyset$. Utilizando o operador de cardinalidade, têm-se $|\{\}| = |\emptyset| = 0$.

Como principais operações entre conjuntos, pode-se destacar:

- União ($\cup$): união de dois conjuntos. Exemplo: $\{1,2,3,4,5\} \cup \{2,4,6,8\} = \{1,2,3,4,5,6,8\}$;
- Intersecção ($\cap$): intersecção de dois conjuntos. Exemplo: $\{1,2,3,4,5\} \cup \{2,4,6,8\} = \{2,4\}$;
- Diferença ($-$ ou $\setminus$): diferença de dois conjuntos. Exemplo $\{1,2,3,4,5\} \setminus \{2,4,6,8\} = \{1,3,5\}$;
- Produto cartesiano ($\times$): Exemplo $\{1,2,3\} \times \{A,B\} = \{(1,A),(2,A),(3,A),(1,B),(2,B),(3,B)\}$;
- Conjunto de partes (ou *power set*): o conjunto de todos os subconjuntos dos elementos de um conjunto. Para o conjunto $A = \{1,2,3\}$ o conjunto das partes seria $2^A = P(A) = \{\{\},\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.

Funções são representadas de forma diferente na matemática discreta. Busca-se estabelecer a relação entre um conjunto de domínio (entrada da função) e um contradomínio (resposta da função). A Equação (A.3) exibe a forma como é utilizada para formalizar uma função. Nesse formato, passa-se a natureza da entrada e da saída de um problema. Por exemplo, a função que gera a correspondência entre o domínio dos inteiros positivos em base decimal para base binária seria $f : x \in Z^+ \rightarrow \{0,1\}^{\log_2(|x|+1)}$.

$$\text{nome da funcao} : \text{dominio} \rightarrow \text{contradominio} \tag{A.3}$$

Para representar uma coleção de itens onde a sequência importa e a repetição pode ocorrer, utiliza-se as tuplas. Uma tupla é representada da forma demonstrada na Equação (A.4).

$$A = (e_1, e_2, \ldots, e_n) \tag{A.4}$$

. Um exemplo de uma tupla, pode ser lista de chamada de uma turma ordenada lexicograficamente.