

# MATEMÁTICA DISCRETA

---

Ano Letivo 2024/2025      (Versão: 13 de Maio de 2025)

Departamento de Matemática, Universidade de Aveiro  
<https://elearning.ua.pt/>

# **CAPÍTULO V**

## **ELEMENTOS DE TEORIA DOS GRAFOS**

### **PARTE III**

#### **ÁRVORES E FLORESTAS**

1. Árvores e florestas

2. Árvores abrangentes de custo mínimo

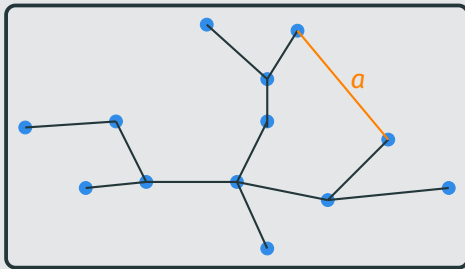
# **1. ÁRVORES E FLORESTAS**

**Definição**

Um grafo simples  $G$  diz-se uma **floresta** se  $G$  não contém ciclos. Uma floresta conexa designa-se por **árvore**.

**Nota**

Uma floresta é um grafo simples cujas componentes conexas são árvores.

**Exemplo (Árvore)**

Acrescentando a aresta **a**, o grafo já não é uma árvore.

**Teorema**

*Para um grafo simples  $G$  com pelo menos um vértice, as seguintes afirmações são equivalentes.*

- (i)  $G$  é uma árvore.*
- (ii)  $G$  é «minimamente conexo», ou seja,  $G$  é conexo e cada aresta é uma ponte.*
- (iii)  $G$  é «maximamente acíclico», ou seja,  $G$  não contém ciclos, mas acrescentando uma aresta obtém-se um ciclo.*

**Definição**

Seja  $G$  um grafo. Um subgrafo abrangente  $T$  de  $G$  diz-se **árvore abrangente** de  $G$  quando  $T$  é uma árvore.

**Corolário**

*Cada grafo finito conexo admite uma árvore abrangente. (Por exemplo, podemos escolher um subgrafo «maximamente acíclico».)*

## Lema

*Cada árvore finita com pelo menos dois vértices tem pelo menos dois vértices de grau 1 (designados por **folhas**).*

## Lema

*Uma árvore com  $n$  vértices tem precisamente  $n - 1$  arestas.*

## Teorema

*Um grafo  $G$  conexo com  $n$  vértices é uma árvore se e só se  $G$  tem  $n - 1$  arestas.*

## Teorema

*Um grafo  $G$  sem ciclos com  $n \geq 1$  vértices é uma árvore se e só se  $G$  tem  $n - 1$  arestas.*

**Teorema**

*Um grafo finito  $G$  é uma floresta se e só se*

$$\varepsilon(G) = \nu(G) - \text{cc}(G).$$

**Demonstração.**

Suponhamos que  $G$  é uma floresta e sejam  $G_1, \dots, G_k$  as componentes conexas de  $G$ . Logo,  $\text{cc}(G) = k$  e

$$\varepsilon(G) = \varepsilon(G_1) + \dots + \varepsilon(G_k) \quad \text{e} \quad \nu(G) = \nu(G_1) + \dots + \nu(G_k).$$

Para cada  $i = 1, 2, \dots, k$ ,  $\varepsilon(G_i) = \nu(G_i) - 1$  (o lema anterior para árvores), portanto,

$$\varepsilon(G) = \nu(G) - k.$$





**Teorema**

Um grafo finito  $G$  é uma floresta se e só se

$$\varepsilon(G) = \nu(G) - \text{cc}(G).$$

**Demonstração.**

Suponha agora que  $\varepsilon(G) - \nu(G) + \text{cc}(G) = 0$  e sejam  $G_1, \dots, G_k$  as componentes conexas de  $G$ . Logo,

$$0 = \underbrace{(\varepsilon(G_1) - \nu(G_1) + 1)}_{\geq 0} + \dots + \underbrace{(\varepsilon(G_k) - \nu(G_k) + 1)}_{\geq 0};$$

ou seja,  $\varepsilon(G_i) - \nu(G_i) + 1 = 0$ , para cada  $i = 1, \dots, k$ . Pelo teorema anterior (sobre árvores), cada componente conexa é uma árvore. Portanto,  $G$  é uma floresta.





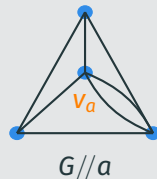
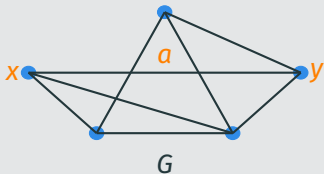
### Fusão de extremos de uma aresta

Seja  $G = (V, E, \psi)$  um grafo e seja  $a \in E$  com  $\psi(a) = \{x, y\}$ . Denotamos por  $G//a$  o grafo obtido a partir de  $G$  por **fusão** de  $x$  e  $y$ . Mais concretamente,  $G//a = (V', E', \psi')$  onde

$$V' = V \setminus \{x, y\} \cup \{v_a\}, \quad E' = E \setminus \{a\}$$

e  $\psi(e) = \psi'(e)$  para toda a aresta  $e \in E$  com  $\psi(e) \cap \{x, y\} = \emptyset$ , em todos os outros casos  $\psi'(e)$  é dado por  $\psi(e)$  com  $v_a$  no lugar de  $x$  e  $y$  (que se fundem no vértice  $v_a$ ).

### Exemplo



### Nota

Seja  $G$  um grafo finito e seja  $a$  uma aresta de  $G$ . Por definição,

$$\varepsilon(G//a) = \varepsilon(G) - 1.$$

### Teorema

Seja  $G$  um grafo finito e sejam  $a, b$  arestas distintas de  $G$ . Então,

$$(G//a) - b = (G - b)//a,$$

ou seja, a operação de fusão de extremos de arestas comuta com a operação de eliminação de arestas.

## Teorema

Seja  $G$  um grafo finito e conexo, e  $a$  uma aresta de  $G$  que não é um lacete. Então,

$$\tau(G) = \tau(G - a) + \tau(G // a).$$


## Demonstração.

Temos

$$\begin{aligned} \tau(G) &= |\{\text{as árvores sem } a\}| + |\{\text{as árvores com } a\}| \\ &= \tau(G - a) + \tau(G // a). \end{aligned}$$

□

## Nota

- Se  $a$  é um lacete em  $G$ , então  $\tau(G) = \tau(G - a)$ .
- Para  em  $G$  com  $d(v_1) = 1$ :  $\tau(G) = \tau(G - v_1)$ .

## Exemplos

Não escrevemos aqui os «nomes» dos vértices.

$$\tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) = 4.$$

$$\tau \left( \begin{array}{c} \bullet \\ \diagup \quad \textcolor{red}{\parallel} \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) = \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) + \tau(\bullet \text{---} \bullet \text{---} \bullet)$$

$$= 4 + 2 \cdot 2 = 8.$$

## Exemplos

Não escrevemos aqui os «nomes» dos vértices.

$$\begin{aligned}
 \tau \left( \begin{array}{c} \text{Graph 1} \end{array} \right) &= \tau \left( \begin{array}{c} \text{Graph 2} \end{array} \right) + \tau \left( \begin{array}{c} \text{Graph 3} \end{array} \right) \\
 &= \tau \left( \begin{array}{c} \text{Graph 4} \end{array} \right) + \tau \left( \begin{array}{c} \text{Graph 5} \end{array} \right) \\
 &= 4 + 8 = 12.
 \end{aligned}$$

The graphs are as follows:

- Graph 1:** A diamond-shaped graph with 5 vertices. The top vertex is connected to two side vertices. The bottom vertex is connected to two side vertices. The two side vertices are connected to each other. The top and bottom vertices are connected to the central vertex. The edge between the top and bottom vertices is highlighted in red.
- Graph 2:** A diamond-shaped graph with 5 vertices. The top vertex is connected to two side vertices. The bottom vertex is connected to two side vertices. The two side vertices are connected to each other. The top and bottom vertices are connected to the central vertex.
- Graph 3:** A diamond-shaped graph with 5 vertices. The top vertex is connected to two side vertices. The bottom vertex is connected to two side vertices. The two side vertices are connected to each other. The top and bottom vertices are connected to the central vertex.
- Graph 4:** A diamond-shaped graph with 5 vertices. The top vertex is connected to two side vertices. The bottom vertex is connected to two side vertices. The two side vertices are connected to each other. The top and bottom vertices are connected to the central vertex.
- Graph 5:** A diamond-shaped graph with 5 vertices. The top vertex is connected to two side vertices. The bottom vertex is connected to two side vertices. The two side vertices are connected to each other. The top and bottom vertices are connected to the central vertex.

## Exemplos

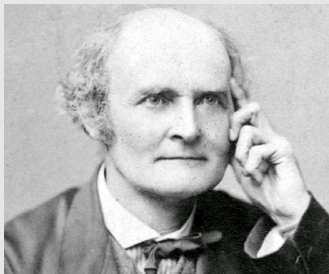
Não escrevemos aqui os «nomes» dos vértices.

$$\begin{aligned}
 \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) &= \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) + \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) \\
 &= \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) + \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) + \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) \\
 &= \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) + \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) + \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) + \tau \left( \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagdown \quad \diagup \\ \bullet \end{array} \right) \\
 &= 4 + 3 + 2 + 3 = 12.
 \end{aligned}$$



**Teorema (Fórmula de Cayley)**

*Para cada  $n \geq 1$ , o número de árvores com  $n$  vértices (etiquetadas) é  $n^{n-2}$ .*

**Referência**

Arthur Cayley (1821 – 1895), matemático britânico.

**Corolário**

*Para cada  $n \geq 1$ ,  $\tau(K_n) = n^{n-2}$ .*

### A ideia

Sejam  $n \geq 2$  e  $V$  um conjunto de  $n$  elementos (tipicamente  $V = \{1, 2, \dots, n\}$ ). Tendo em conta o Teorema de Cayley definimos uma bijeção entre

*o conjunto de todas as árvores  $T = (V, E)$*

e

*o conjunto de todas as sequências  $(a_1, a_2, \dots, a_{n-2})$  de comprimento  $n - 2$  com  $a_i \in V$ .*

A sequência  $(a_1, a_2, \dots, a_{n-2})$  associada à árvore  $T$  diz-se **código de Prüfer** de  $T$ .

Consequentemente, o número de árvores  $T = (V, E)$  é  $n^{n-2}$ .

» saltar

### O procedimento

Sejam  $n \geq 2$  e  $V$  um conjunto de  $n$  elementos. Definimos a função

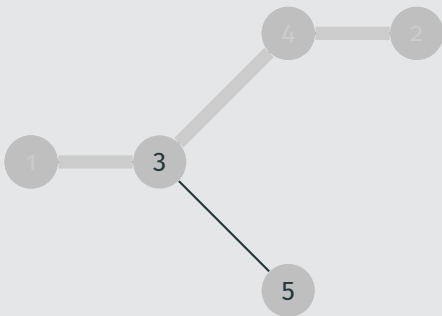
$$\text{pruefer}: \{\text{árvores em } V\} \longrightarrow \{(a_1, \dots, a_{n-2}) \mid a_i \in V\}$$

de seguinte maneira. Em primeiro lugar, escolhemos uma ordem total em  $V$ , e depois aplicamos o seguinte algoritmo:

1.  $T$  = a árvore em consideração,  $i = 1$ .
2. Se  $T$  tem dois (ou menos) vértices, **Parar**.
3. procurar o menor vértice  $v$  com grau 1 (a menor folha).
4.  $a_i$  = o único vizinho de  $v$ .
5.  $T = T - v$  (o que ainda é uma árvore!!) e  $i = i + 1$ .
6. **Voltar para 2.**

**Exemplo**

A árvore  $T$ :



O código de Prüfer de  $T$ :  $\text{pruefer}(T) = (3, 4, 3)$ .

**Nota**

Cada vértice  $v$  aparece  $d(v) - 1$  vezes em  $(a_1, \dots, a_{n-2})$ . Em particular, um vértice  $v$  é uma folha se e somente se  $v$  não ocorre em

### O procedimento

Sejam  $n \geq 2$  e  $V$  um conjunto de  $n$  elementos totalmente ordenado. Definimos a função

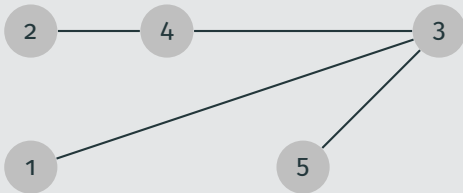
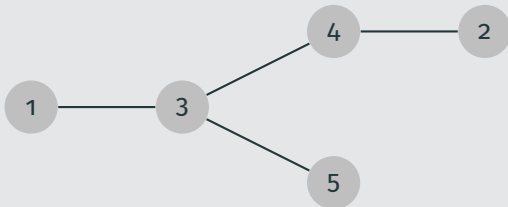
$$\text{unpruefer}: \{(a_1, \dots, a_{n-2}) \mid a_i \in V\} \longrightarrow \{\text{árvores em } V\}$$

de seguinte maneira:

- (o) (Desenhar os  $n$  vértices no papel/quadro/areia/....)
- (1)  $P$  = a sequência  $(a_1, \dots, a_{n-2})$  dada,  $L$  = a lista ordenada dos vértices.
- (2) Se  $L$  tem comprimento dois (e portanto  $P$  tem comprimento zero), então ligar os dois vértices correspondentes e
- (3) Considerar o menor elemento em  $L$  que não pertence a  $P$ , e o primeiro elemento de  $P$ . Ligar as dois vértices correspondentes e
- (4) **Voltar para 2.**

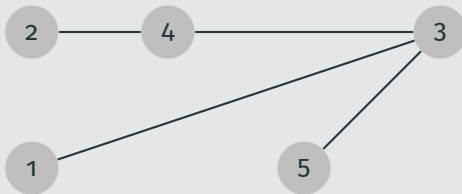
**Exemplo**

Consideremos  $P = (3, 4, 3)$  e  $L = (1, 2, 3, 4, 5)$ .

**Para comparar**

**Exemplo**

Consideremos  $P = (3, 4, 3)$  e  $L = (1, 2, 3, 4, 5)$ .

**Teorema**

*Verificam-se as igualdades*

$$\text{pruefer} \circ \text{unpruefer} = \text{id} \quad \text{e} \quad \text{unpruefer} \circ \text{pruefer} = \text{id},$$

*logo  $\text{unpruefer} = \text{pruefer}^{-1}$  e por isso  $\text{pruefer}$  e  $\text{unpruefer}$  são funções bijetivas.*

## **2. ÁRVORES ABRANGENTES DE CUSTO MÍNIMO**



**O contexto**

Consideremos grafos finitos  $G = (V, E, \psi)$  com uma função

$$W: E \longrightarrow [0, \infty]$$

de custos não negativos nas arestas. Dada um subgrafo  $H$  de  $G$ , com o conjunto de arestas  $E' \subseteq E$ , definimos o custo de  $H$  como

$$\sum_{e \in E'} W(e).$$

**O objetivo**

Para um grafo conexo finito  $G = (V, E, \psi)$  com  $W: E \longrightarrow [0, \infty]$ , **encontrar uma árvore abrangente de custo mínimo.**

## Dois algoritmos

- O algoritmo de Kruskal.
- O algoritmo de Prim.

---

Joseph Bernard Kruskal (1928 – 2010) matemático, estatístico, informático e psicometrista estadunidense, e Robert Clay Prim (1921) matemático e informático estadunidense.

### Descrição do algoritmo

Consideremos um grafo conexo  $G = (V, E, \psi)$  e  $W: E \rightarrow [0, \infty]$ .

1. Ordenar as arestas  $(a_1, \dots, a_m)$  de  $G$  por ordem não decrescente do seu custo, ou seja,

$$W(a_1) \leq W(a_2) \leq \dots \leq W(a_m).$$

2.  $E' = \emptyset, i = 1$ .

3. **Enquanto**  $T = (V, E')$  não é conexa:

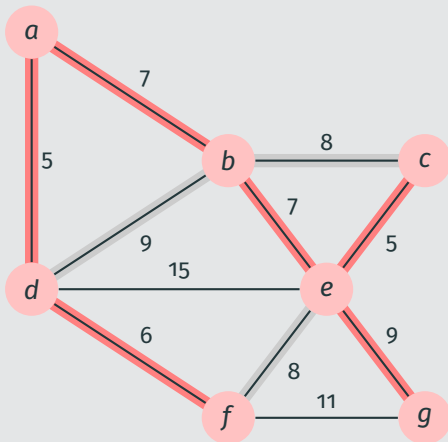
- **Se**  $(V, E' \cup \{a_i\})$  não tem ciclos, **então**  $E' = E' \cup \{a_i\}$ .
- $i = i + 1$ .
- **Saltar para** o início de 3.

4. Devolver a árvore abrangente  $(V, E')$  de  $G$  de custo mínimo.

## Exemplo

Ordenar as arestas:  $ad, ce, df, ab, be, bc, ef, bd, eg, fg, de$ .

1.  $E' = \emptyset$
2.  $E' = \{ad\}$
3.  $E' = \{ad, ce\}$
4.  $E' = \{ad, ce, df\}$
5.  $E' = \{ad, ce, df, ab\}$
6.  $E' = \{ad, ce, df, ab, be\}$
7.  $E' = \{ad, ce, df, ab, be\}, bc \notin E'$
8.  $E' = \{ad, ce, df, ab, be\}, ef \notin E'$
9.  $E' = \{ad, ce, df, ab, be\}, bd \notin E'$
10.  $E' = \{ad, ce, df, ab, be, eg\}$



**Terminar:** O grafo  $T = (V, E')$  é conexo.  $W(T) = 39$ .

### Descrição do algoritmo

Consideramos um grafo conexo  $G = (V, E, \psi)$  e  $W: E \rightarrow [0, \infty]$ .

1. Escolher um vértice  $u \in V$ .
2.  $V' = \{u\}$  e  $E' = \emptyset$ .
3. **Enquanto**  $V' \subsetneq V$ :
  - Entre todas as arestas  $e \in E$  com

$$\psi(e) = vw, \quad v \in V', \quad w \notin V',$$

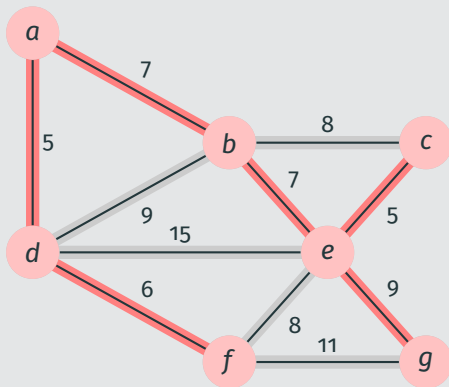
determinar uma aresta de menor custo:  $e^*$  com  $\psi(e^*) = v^*w^*$ ,  
 $v^* \in V'$  e  $w^* \notin V'$ .

- $V' = V' \cup \{w^*\}$ ,  $E' = E' \cup \{e^*\}$ .
  - **Saltar para** o início de 3.
4. Devolver a árvore abrangente  $(V, E')$  de  $G$  de custo mínimo.

## Exemplo

Escolhemos o vértice  $d$ .

1.  $V' = \{d\}, E' = \emptyset$
2.  $V' = \{d, a\}, E' = \{ad\}$
3.  $V' = \{d, a, f\}, E' = \{ad, df\}$
4.  $V' = \{d, a, f, b\},$   
 $E' = \{ad, df, ab\}$
5.  $V' = \{d, a, f, b, e\},$   
 $E' = \{ad, df, ab, be\}$
6.  $V' = \{d, a, f, b, e, c\},$   
 $E' = \{ad, df, ab, be, ec\}$
7.  $V' = \{d, a, f, b, e, c, g\},$   
 $E' = \{ad, df, ab, be, ec, eg\}$



**Terminar:**  $V' = V.$   $W(V, E') = 39.$

Grafos em  $\text{\LaTeX}$  e tikz:

<http://www.texample.net/tikz/examples/prims-algorithm/>