Algoritmos e Estruturas de Dados II

Algoritmos em Grafos

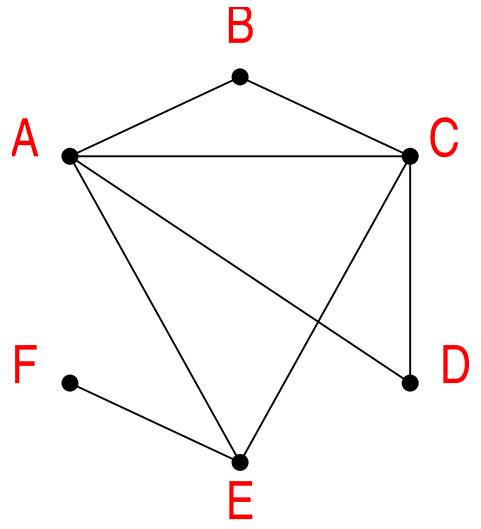
Antonio Alfredo Ferreira Loureiro

loureiro@dcc.ufmg.br

http://www.dcc.ufmg.br/~loureiro

Motivação

- Suponha que existam seis sistemas computacionais (A, B, C, D, E, e F) interconectados entre si da seguinte forma:

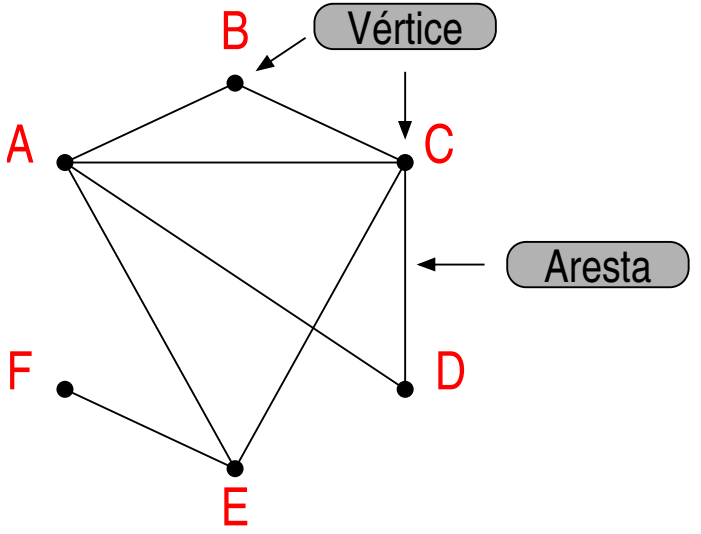


➜ Esta informação pode ser representada por este *diagrama*, chamado de **grafo**.

➜ Este diagrama identifica unicamente um grafo.

Motivação

- Dois objetos especiais:
  - Vértices
  - Arestas



Definição

Um grafo G consiste de dois conjuntos finitos:

1. Vértices $V(G)$
2. Arestas $E(G)$

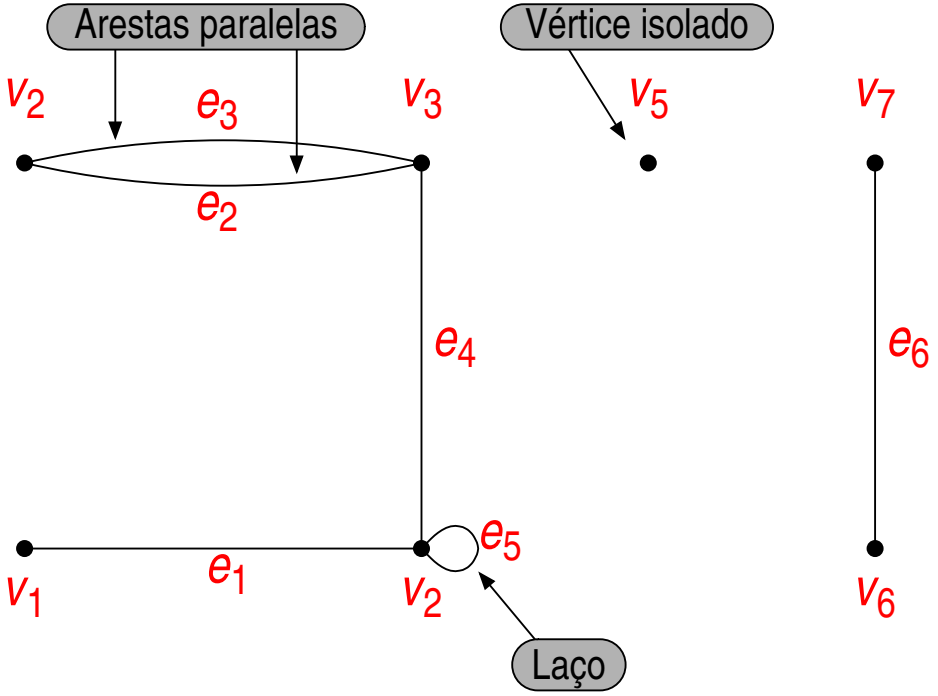
Em geral, um grafo G é representado como:

$$G = (V, E)$$

Terminologia

- Cada aresta está associada a um conjunto de um ou dois vértices, chamados nós terminais.
- **Extremidade** de uma aresta: vértice da aresta.
- Função aresta–extremidade: associa aresta a vértices.
- **Laço** (*Loop*): aresta somente com nó terminal.
- Arestas **paralelas**: arestas associadas ao mesmo conjunto de vértices.
- Uma aresta é dita **conectar** seus nós terminais.
- Dois vértices que são conectados por uma aresta são chamados de **adjacentes**.
- Um vértice que é nó terminal de um laço é dito ser **adjacente a si próprio**.
- Uma aresta é dita ser **incidente** a cada um de seus nós terminais.
- Duas arestas incidentes ao mesmo vértice são chamadas de **adjacentes**.
- Um vértice que não possui nenhuma aresta incidente é chamado de **isolado**.
- Um grafo com nenhum vértice é chamado de **vazio**.

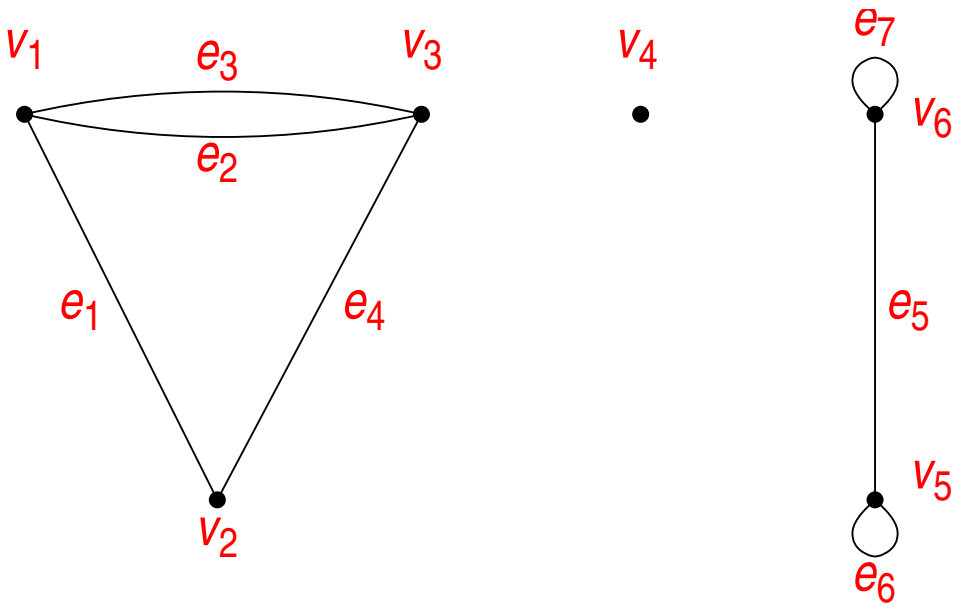
Terminologia



Terminologia

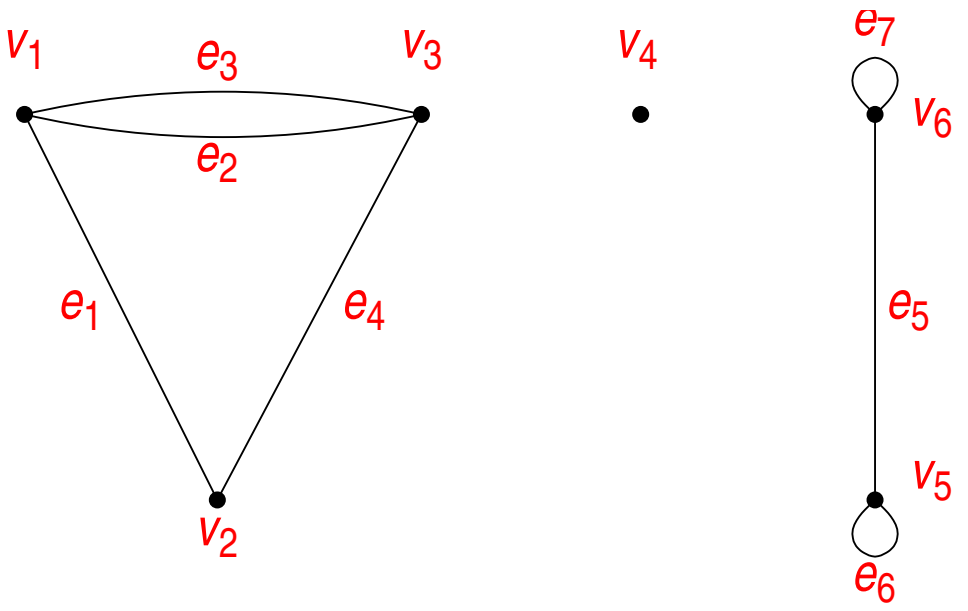
- Conjunto de vértices:
  $\{v_1, v_2, v_3, v_4, v_5, v_6\}$.
- Conjunto de arestas:
  $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$.
- Função aresta–vértice:

Aresta	Vértice
e_1	$\{v_1, v_2\}$
e_2	$\{v_1, v_3\}$
e_3	$\{v_1, v_3\}$
e_4	$\{v_2, v_3\}$
e_5	$\{v_5, v_6\}$
e_6	$\{v_5\}$
e_7	$\{v_6\}$



Terminologia

- e_1, e_2 e e_3 são incidentes a v_1.
- v_2 e v_3 são adjacentes a v_1.
- e_2, e_3 e e_4 são adjacentes a e_1.
- e_6 e e_7 são laços.
- e_2 e e_3 são paralelas.
- v_5 e v_6 são adjacentes entre si.
- v_4 é um vértice isolado.



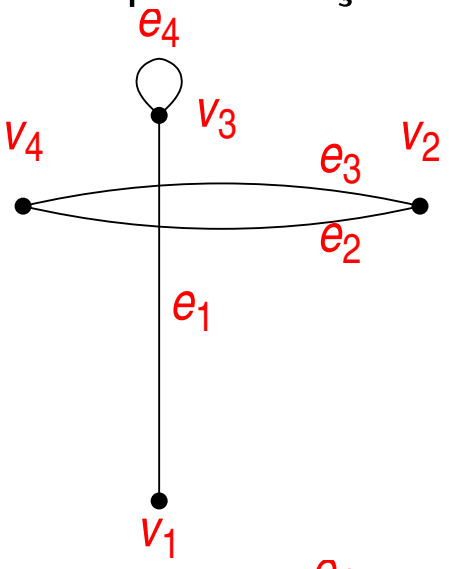
Terminologia

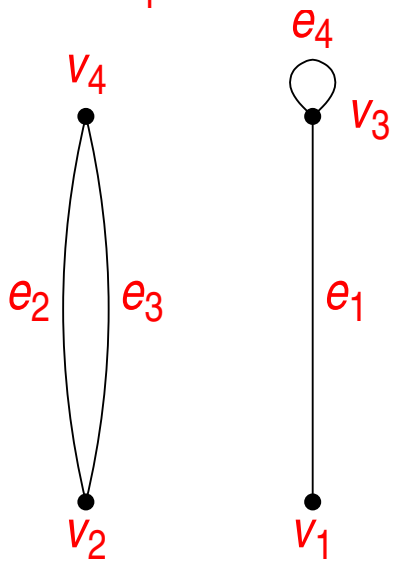
Seja um grafo especificado como:

- Conjunto de vértices: $\{v_1, v_2, v_3, v_4\}$.
- Conjunto de arestas: $\{e_1, e_2, e_3, e_4\}$.
- Função aresta–vértice:

Aresta	Vértice
e_1	$\{v_1, v_3\}$
e_2	$\{v_2, v_4\}$
e_3	$\{v_2, v_4\}$
e_4	$\{v_3\}$

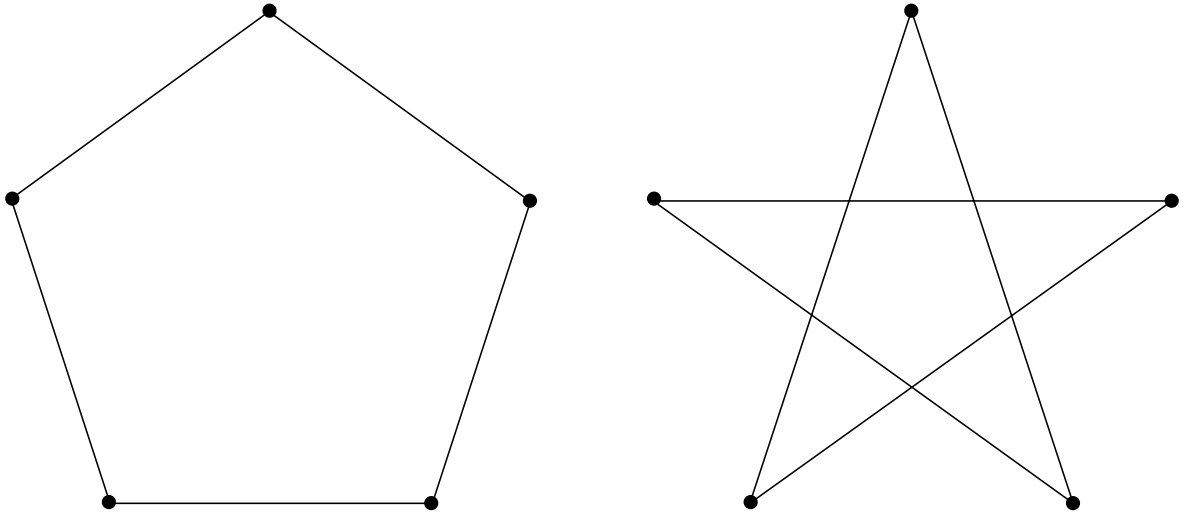
Duas possíveis representações deste grafo:





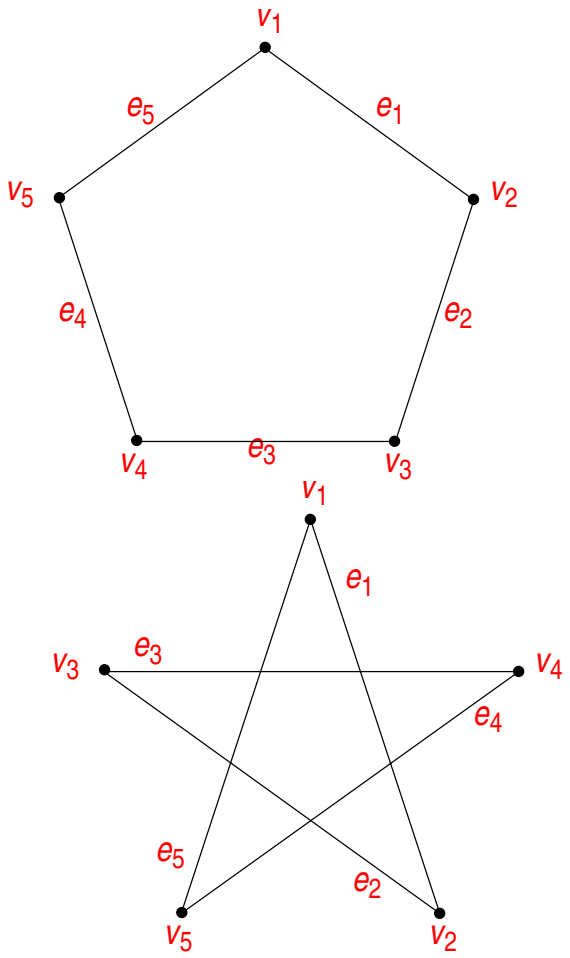
Terminologia

Considere os dois diagramas abaixo. Rotule os vértices e as arestas de tal forma que os dois diagramas representem o mesmo grafo.



Terminologia

Uma possível identificação de vértices e rótulos pode ser:



Os dois diagramas são representados por:

- Conjunto de vértices: $\{v_1, v_2, v_3, v_4, v_5\}$.
- Conjunto de arestas: $\{e_1, e_2, e_3, e_4, e_5\}$.
- Função aresta–vértice:

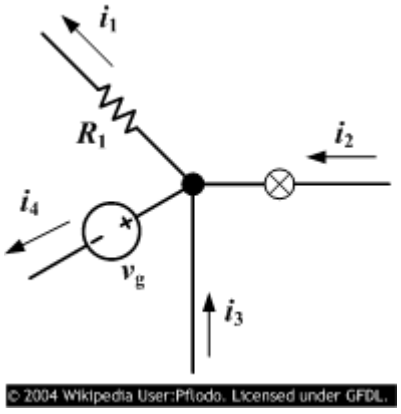
Aresta	Vértice
e_1	$\{v_1, v_2\}$
e_2	$\{v_2, v_3\}$
e_3	$\{v_3, v_4\}$
e_4	$\{v_4, v_5\}$
e_5	$\{v_5, v_1\}$

Modelos usando grafos

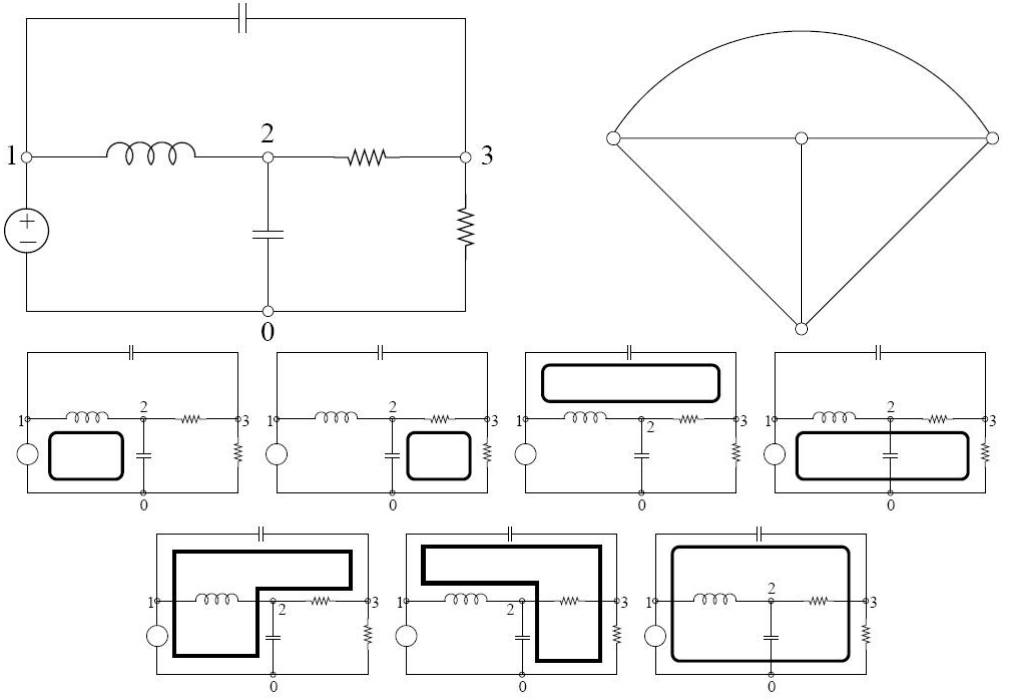
Grafo	**Vértice**	**Aresta**
Comunicação	Centrais telefônicas, Computadores, Satélites	Cabos, Fibra óptica, Enlaces de microondas
Circuitos	Portas lógicas, registradores, processadores	Filamentos
Hidráulico	Reservatórios, estações de bombeamento	Tubulações
Financeiro	Ações, moeda	Transações
Transporte	Cidades, Aeroportos	Rodovias, Vias aéreas
Escalonamento	Tarefas	Restrições de precedência
Arquitetura funcional de um software	Módulos	Interações entre os módulos
Internet	Páginas Web	*Links*
Jogos de tabuleiro	Posições no tabuleiro	Movimentos permitidos
Relações sociais	Pessoas, Atores	Amizades, Trabalho conjunto em filmes

Modelos usando grafos
Circuito elétrico: Leis de Kirchoff

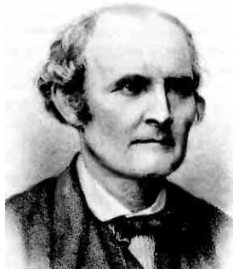
Gustav Kirchoff (1824–1887), físico alemão. Foi o primeiro a analisar o comportamento de "árvores matemáticas" com a investigação de circuitos elétricos.



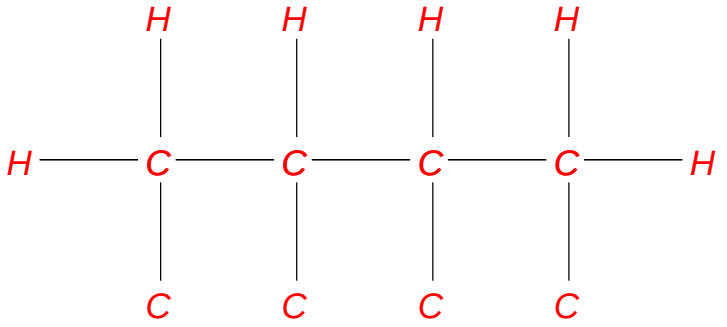
$$i_1 + i_4 = i_2 + i_3$$



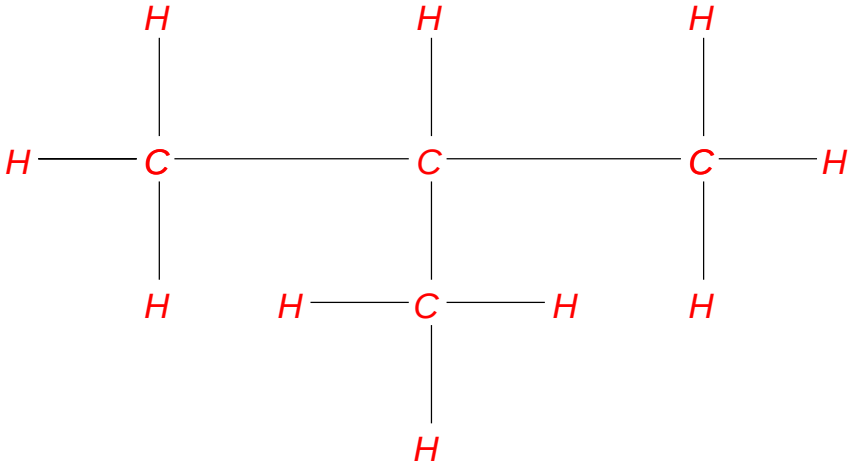
Modelos usando grafos
Estruturas de moléculas de hidrocarboneto



Arthur Cayley (1821–1895), matemático inglês. Logo após o trabalho de Kirchoff, Cayley usou "árvores matemáticas" para enumerar todos os isômeros para certos hidrocarbonetos.



Butano

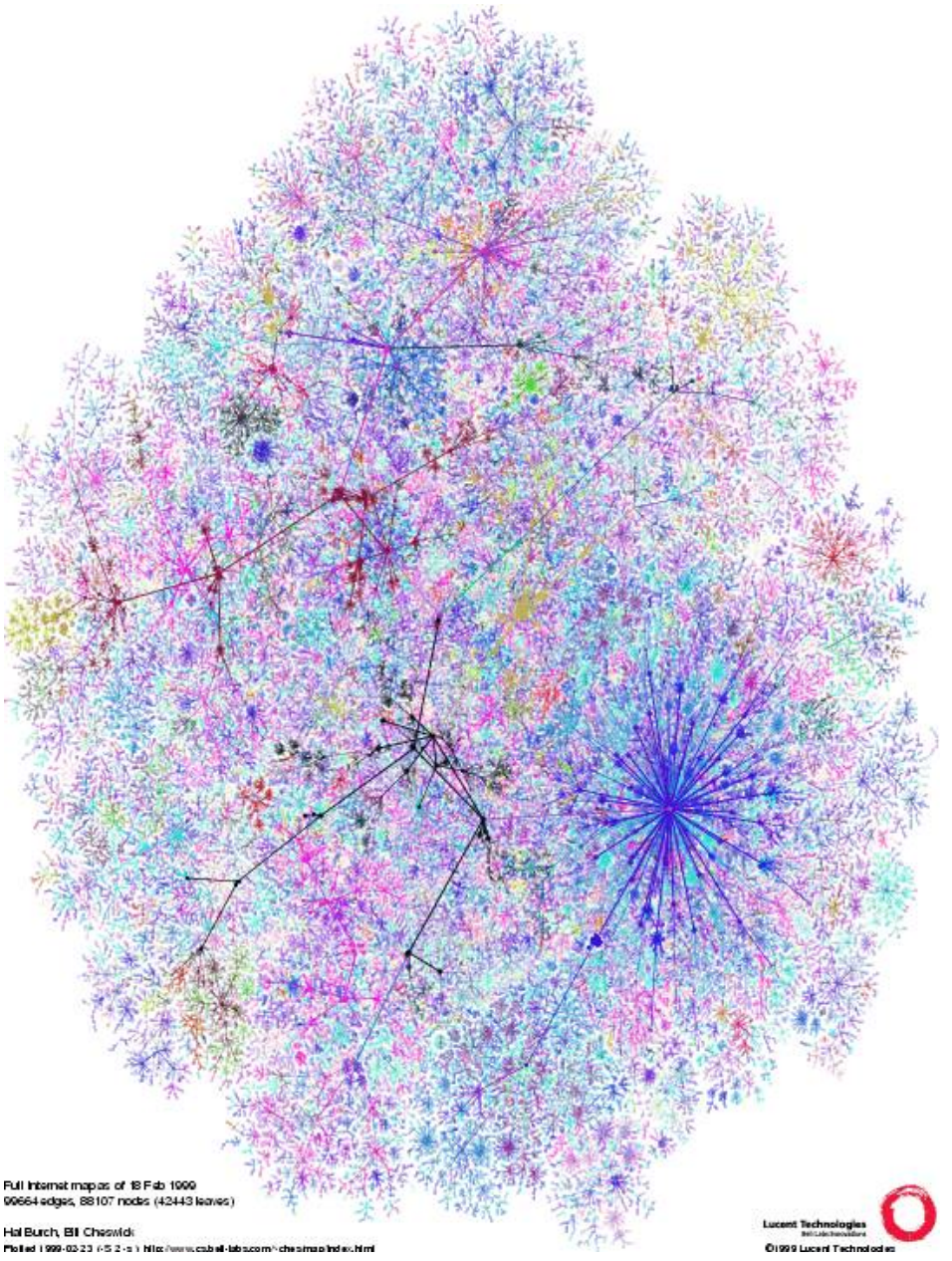


Isobutano

Modelos usando grafos
Conectividade na Internet

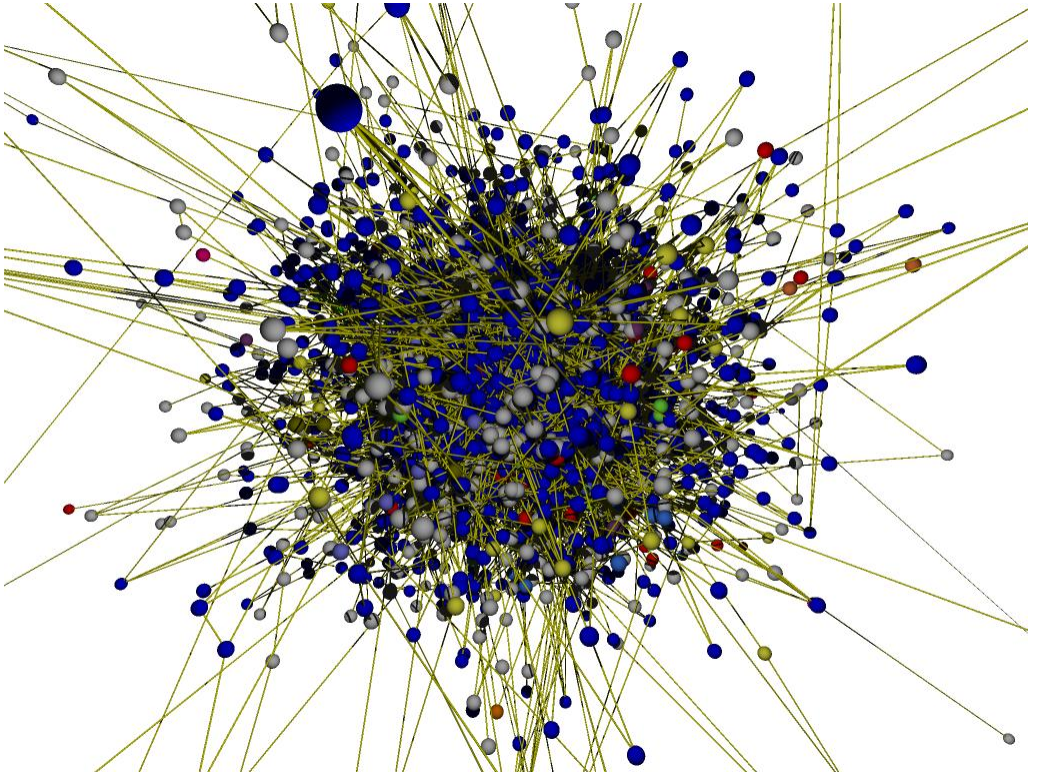
Este grafo mostra a conectividade entre roteadores na Internet, resultado do trabalho "*Internet Mapping Project*" de Hal Burch e Bill Cheswick.

Atualmente o trabalho está sendo desenvolvido comercialmente pela empresa Lumeta (`www.lumeta.com`).



Modelos usando grafos
Conectividade na Internet

Este trabalho de Stephen Coast (`http://www.fractalus.com/steve/stuff/ipmap/`) está "medindo" e mapeando a estrutura e desempenho da Internet. Este é um de seus trabalhos iniciais.

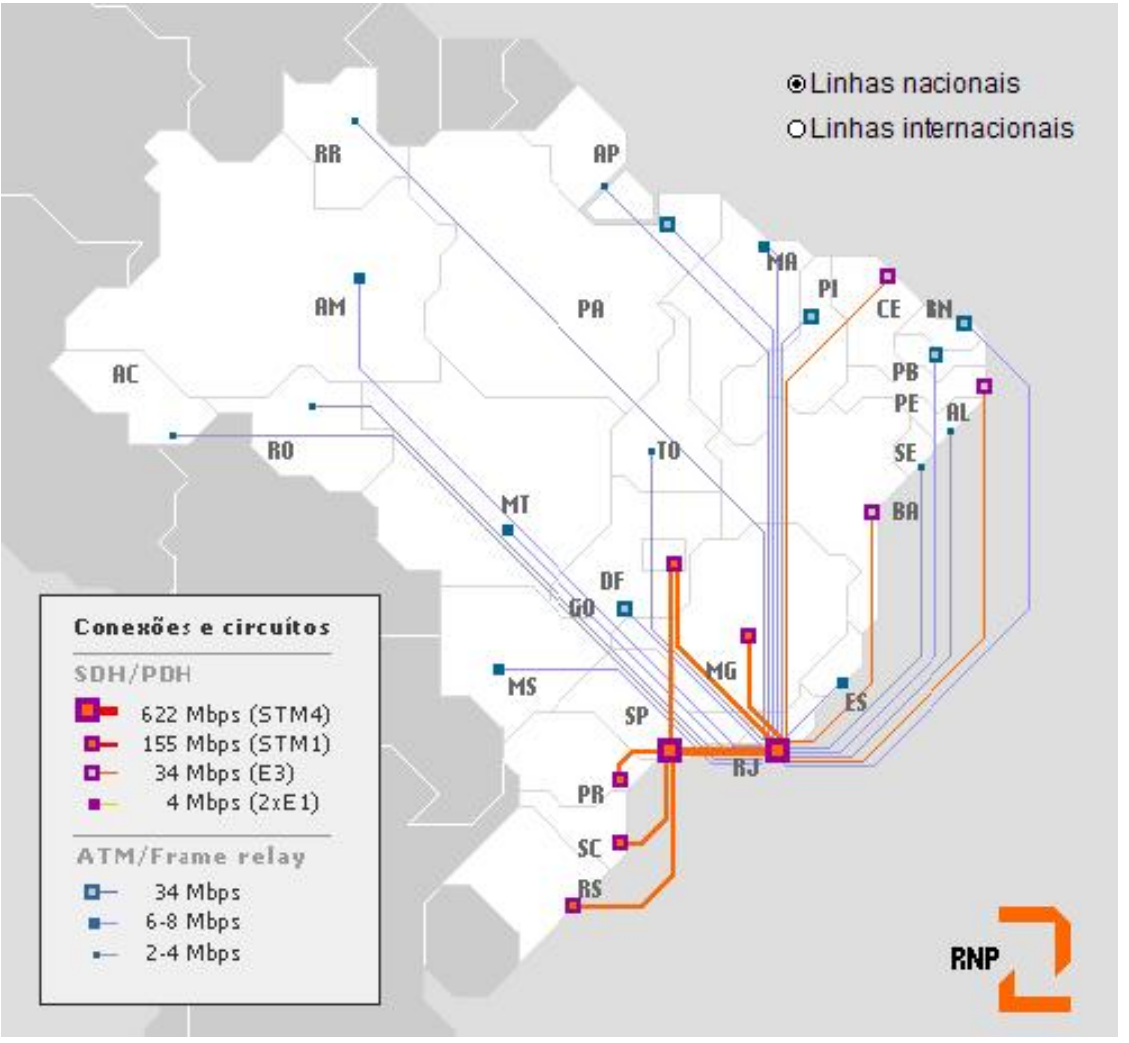


Modelos usando grafos
Conectividade na RNP2

A Rede Nacional de Pesquisa (RNP) criou a primeira infra-estrutura de comunicação (*back-bone*) no Brasil para interconexão com a Internet. Atualmente, este *backbone* é conhecido como RNP2.

O grafo de conectividade da RNP2 tem uma "estrutura" (topologia) basicamente na forma de estrela. Note que diferentes enlaces de comunicação (arestas) possuem diferentes capacidades.

A Internet é formada basicamente por interconexão de Sistemas Autônomos (AS – *Autonomous System*), onde cada AS é um *backbone* distinto.



Modelos usando grafos
Grafo de derivação sintática

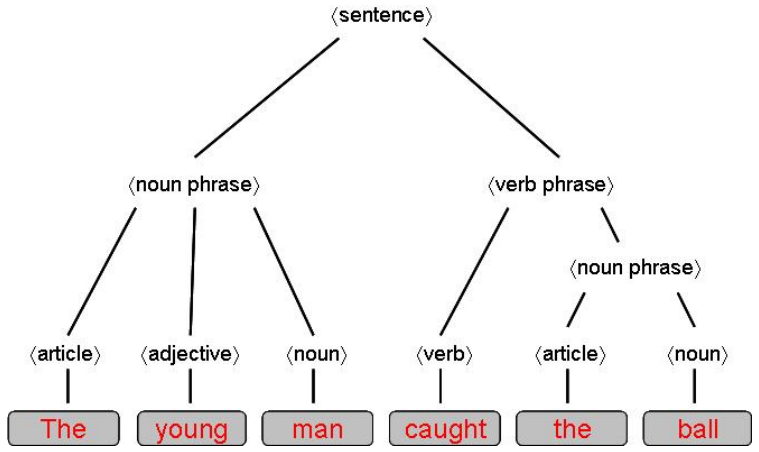
Noam Chomsky

John Backus

Peter Naur

Chomsky e outros desenvolveram novas formas de descrever a sintaxe (estrutura gramatical) de linguagens naturais como inglês. Este trabalho tornou-se bastante útil na construção de compiladores para linguagens de programação de alto nível. Neste estudo, árvores (grafos especiais) são usadas para mostrar a derivação de sentenças corretas gramaticalmente a partir de certas regras básicas.

É comum representar estas regras, chamadas de produção, usando uma notação proposta por Backus (1959) e modificada por Naur (1960) usada para descrever a linguagem de programação Algol. Esta notação é chamada de BNF (Backus-Naur Notation).



Notação BNF (subconjunto da gramática da língua inglesa):

⟨sentence⟩ ::= ⟨noun phrase⟩⟨verb phrase⟩
⟨noun phrase⟩ ::= ⟨article⟩⟨noun⟩ |
⟨article⟩⟨adjective⟩⟨noun⟩
⟨verb phrase⟩ ::= ⟨verb⟩⟨noun phrase⟩
⟨article⟩ ::= the
⟨adjective⟩ ::= young
⟨noun⟩ ::= man | ball
⟨verb⟩ ::= caught

Modelos usando grafos
Vegetarianos e Canibais (1)

- Seja uma região formada por vegetarianos e canibais.
- Inicialmente, dois vegetarianos e dois canibais estão na margem esquerda (ME) de um rio.
- Existe um barco que pode transportar no máximo duas pessoas e sempre atravessa o rio com pelo menos uma pessoa.
- O objetivo é achar uma forma de transportar os dois vegetarianos e os dois canibais para a margem direita (MD) do rio.
- Em nenhum momento, o número de canibais numa margem do rio pode ser maior que o número de vegetarianos, caso contrário, . . .

Modelos usando grafos
Vegetarianos e Canibais (2)

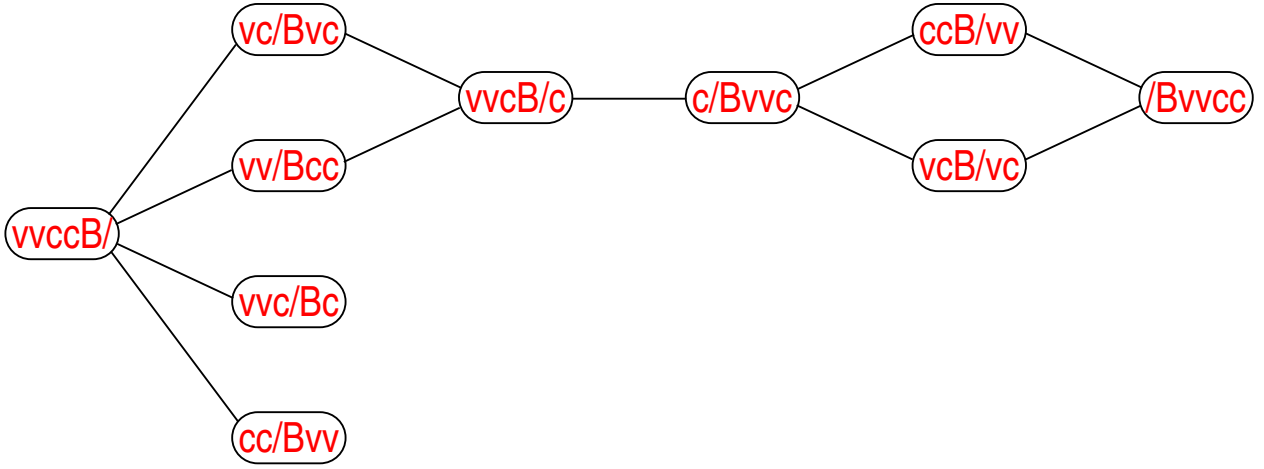
- Solução:
  - Notação para representar cada cenário possível.
  - Modelo para representar a mudança de um cenário em outro válido.

- Notação: ME/MD
  - vvccB/ $\longrightarrow$ ME: 2v, 2c e o barco (B); MD: –.
  - vc/Bvc $\longrightarrow$ ME: 1v, 1c; MD: B, 1v e 1c.

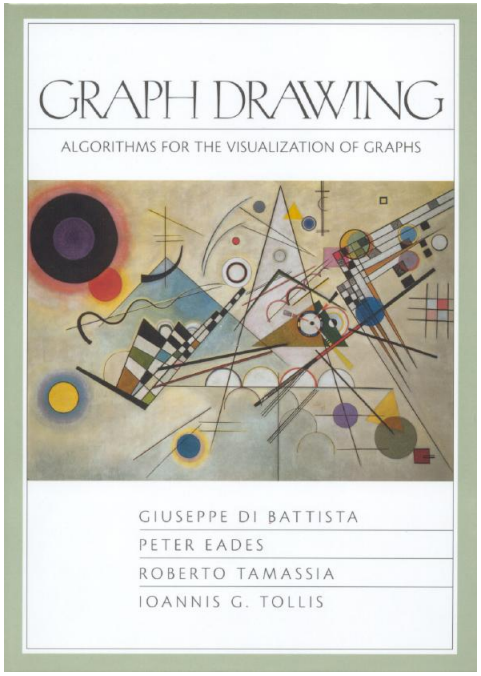
- Modelo: grafo
  - Vértice: cenário válido.
  - Aresta: transição válida de um dado cenário em outro.

Modelos usando grafos
Vegetarianos e Canibais (3)

Uma possível seqüência válida de cenários é:



Modelos usando grafos
Visualizando grafos



Graph Drawing: Algorithms for the Visualization of Graphs. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, e Ioannis G. Tollis. Prentice Hall Engineering, Science & Math, 432 pp., ISBN 0-13-301615-3.

Para muitas aplicações é importante desenhar grafos com certas restrições:

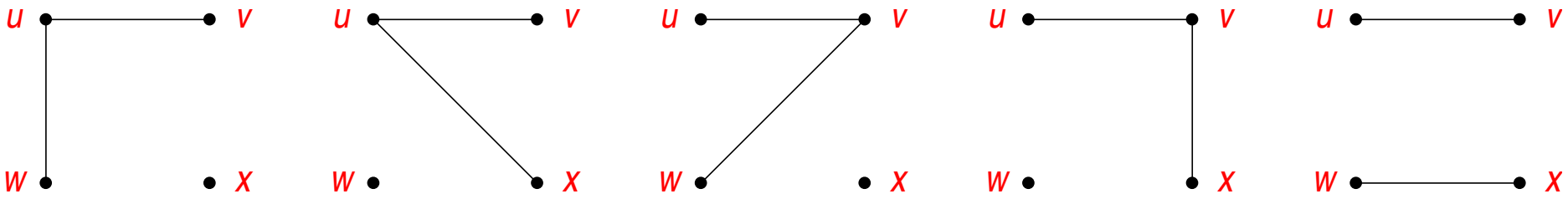
– Planares, i.e., não há cruzamento de arestas

Grafo simples

Definição: Um grafo simples é um grafo que não possui laços nem arestas paralelas. Num grafo simples, uma aresta com vértices (nós terminais) u e v é representada por uv.

Exemplo: Quais são os grafos com quatro vértices $\{u, v, w, x\}$ e duas arestas, sendo que uma delas é a aresta uv?

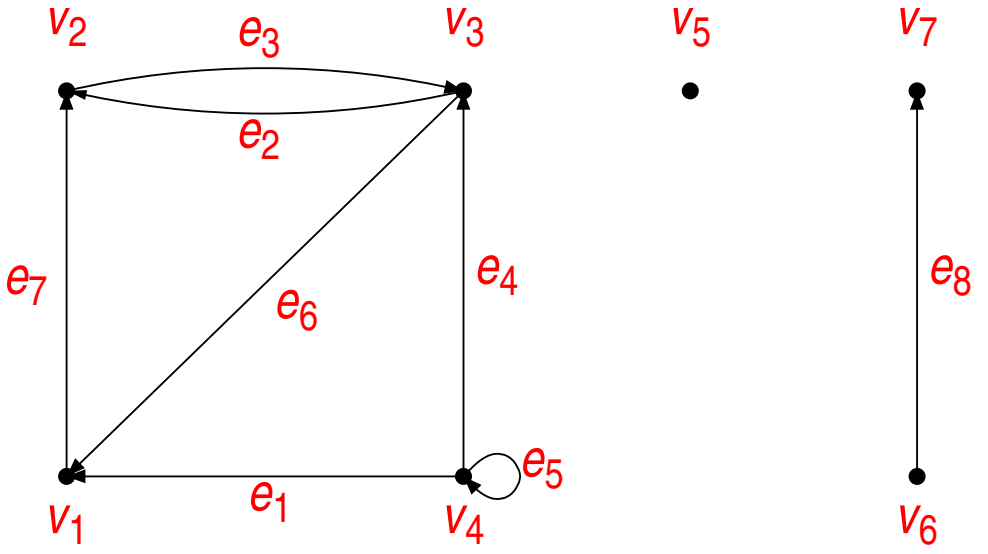
- Dado quatro vértices, existem $C(4, 2) = 6$ subconjuntos, que definem arestas diferentes: $\{uv, uw, ux, vw, vx, wx\}$.
- Logo, todos os grafos simples de quatro vértices e duas arestas, sendo uma delas a uv são:



Grafo dirigido (1)

Definição: Um grafo dirigido ou digrafo ou direcionado G consiste de dois conjuntos finitos:

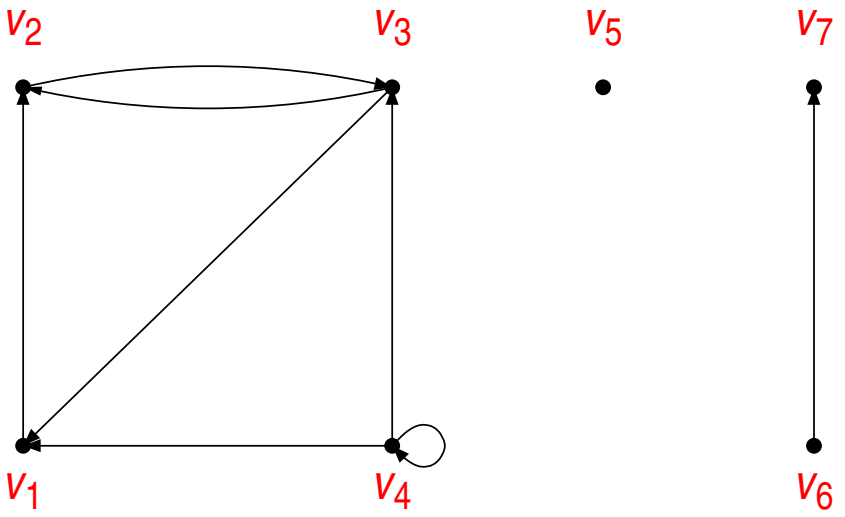
1. Vértices $V(G)$
2. Arestas dirigidas $E(G)$, onde cada aresta é associada a um par ordenado de vértices chamados de nós terminais. Se a aresta e é associada ao par (u, v) de vértices, diz-se que e é a aresta dirigida de u para v.



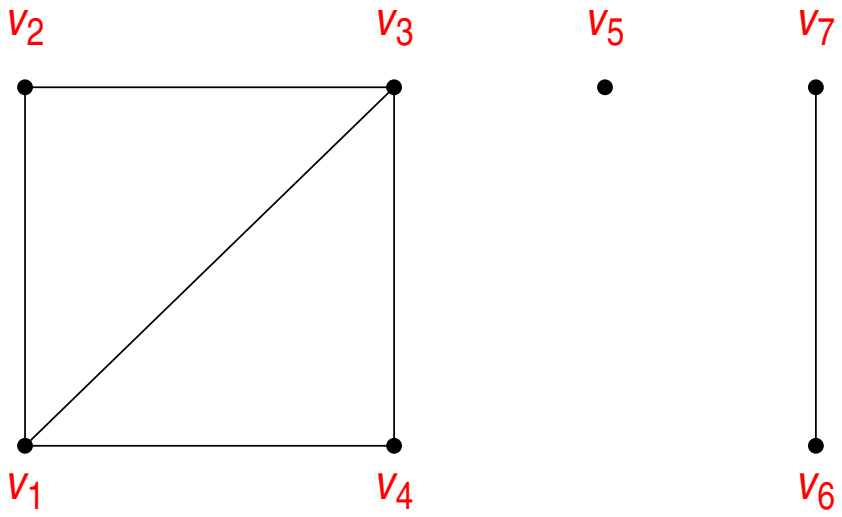
Grafo dirigido (2)

Para cada grafo dirigido, existe um grafo simples (não dirigido) que é obtido removendo as direções das arestas, e os *loops*.

Grafo dirigido:



Grafo não dirigido correspondente:



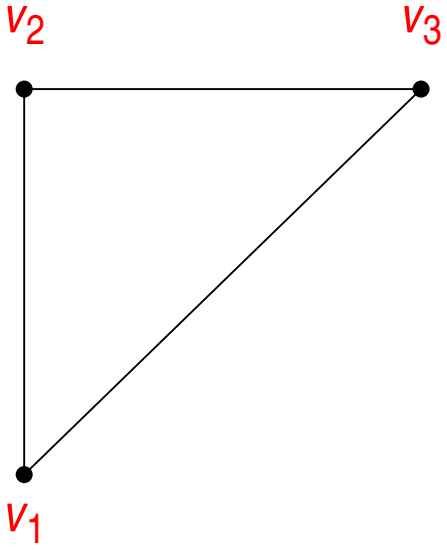
Grafo dirigido (3)

- A versão dirigida de um grafo não dirigido $G = (V, E)$ é um grafo dirigido $G' = (V', E')$ onde $(u, v) \in E'$ sse $(u, v) \in E$.

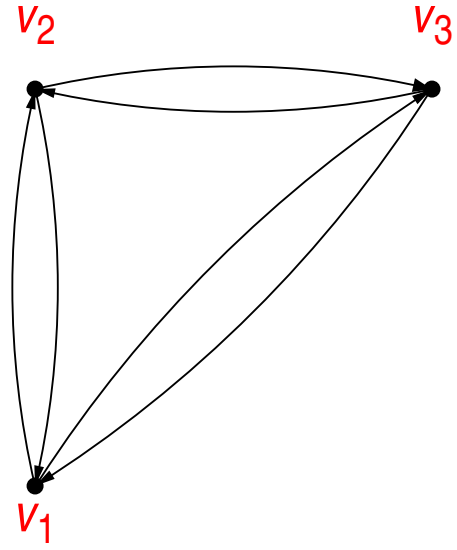
- Cada aresta não dirigida (u, v) em G é substituída por duas arestas dirigidas (u, v) e (v, u).

- Em um grafo dirigido, um vizinho de um vértice u é qualquer vértice adjacente a u na versão não dirigida de G.

Grafo não dirigido:



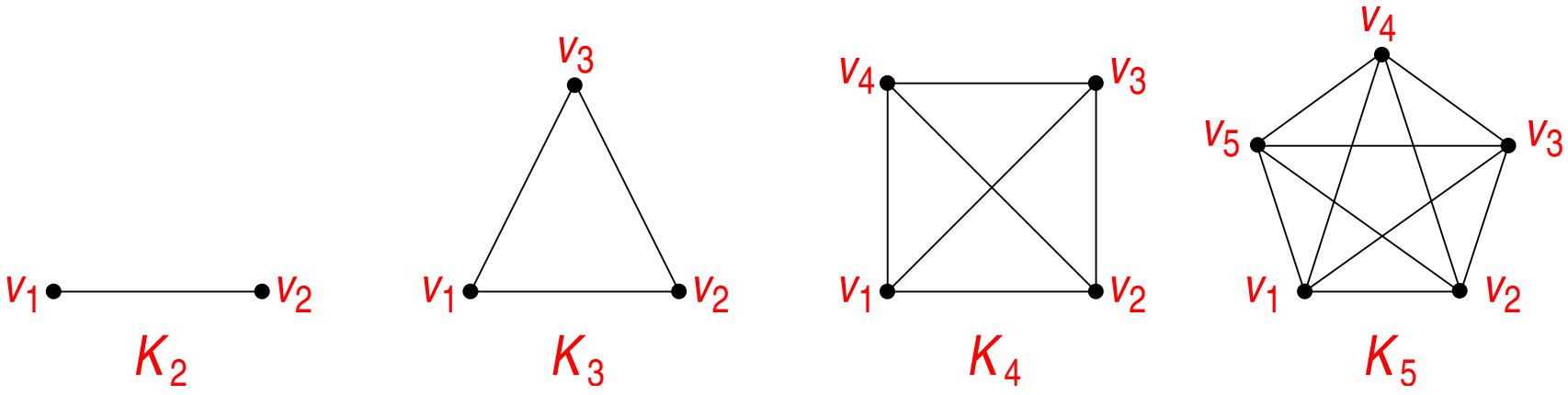
Grafo dirigido correspondente:



Grafo completo (1)

Definição: Um grafo completo de n vértices, denominado K_n*, é um grafo simples com n vértices $v_1, v_2, \ldots, v_n$, cujo conjunto de arestas contém exatamente uma aresta para cada par de vértices distintos.

Exemplo: Grafos completos com 2, 3, 4, e 5 vértices.



*A letra K representa a letra inicial da palavra *komplett* do alemão, que significa "completo".

Grafo completo (2)

Dado o grafo completo K_n temos que

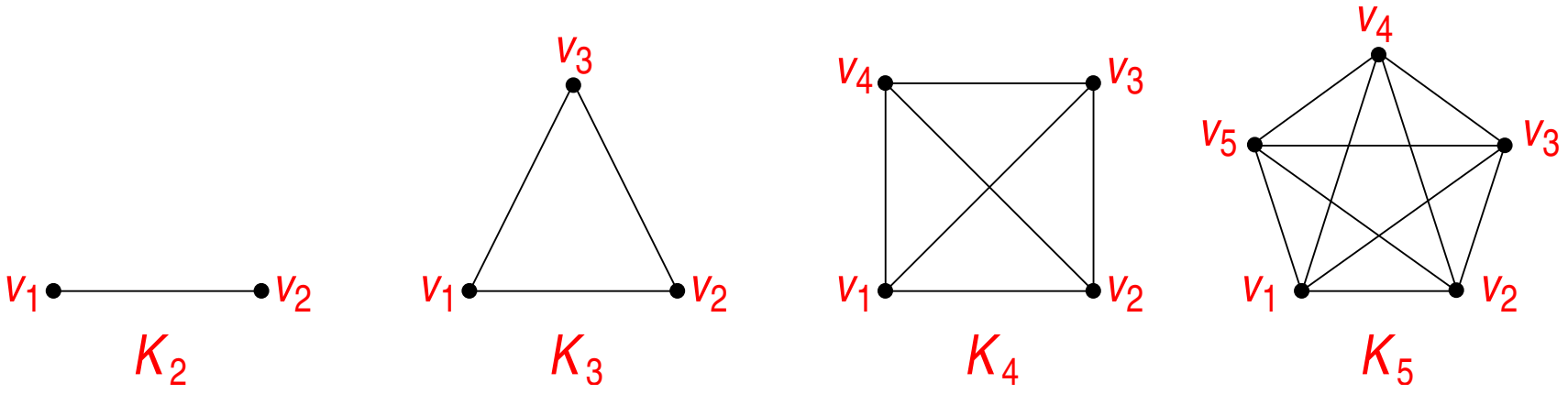
Vértice	está conectado aos vértices (não conectados ainda)	através de # arestas
v_1	$v_2, v_3, \ldots, v_n$	$n-1$
v_2	$v_3, v_4, \ldots, v_n$	$n-2$
$\vdots$	$\vdots$	$\vdots$
v_{n-1}	v_n	1
v_n	–	0

ou seja, se contarmos o número total de arestas de K_n temos

$$\sum_{i=1}^{n-1} i = \frac{(n-1)\cdot n}{2} = \frac{n^2-n}{2} = \frac{(|V|^2-|V|)}{2}$$

Grafo completo (3)

Os grafos K_2, K_3, K_4, e K_5



possuem a seguinte quantidade de arestas:

Grafo	# arestas
K_2	1
K_3	3
K_4	6
K_5	10

Quantidade de grafos distintos com n vértices (1)

O número total de grafos distintos com n vértices ($|V|$) é

$$2^{\frac{n^2-n}{2}} = 2^{\frac{(|V|^2-|V|)}{2}}$$

que representa a quantidade de maneiras diferentes de escolher um subconjunto a partir de

$$\frac{n^2-n}{2} = \frac{(|V|^2-|V|)}{2}$$

possíveis arestas de um grafo com n vértices.

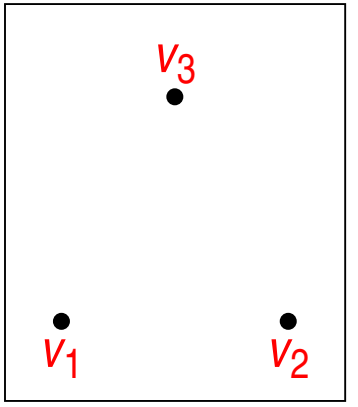
Quantidade de grafos distintos com n vértices (2)

Exemplo: Quantos grafos distintos com 3 vértices existem?

- Um grafo com 3 vértices v_1, v_2 e v_3 possui no máximo 3 arestas, ou seja, $E = \{v_1v_2, v_1v_3, v_2v_3\}$.
- O número de sub-conjuntos distintos de E é dado por $\mathcal{P}(E)$, ou seja, o conjunto potência de E que vale $2^{|E|}$.

$$\mathcal{P}(E) = \left\{ \begin{array}{c} \emptyset, \\ \{v_1v_2\}, \\ \{v_1v_3\}, \\ \{v_2v_3\}, \\ \{v_1v_2, v_2v_3\}, \\ \{v_1v_3, v_2v_3\}, \\ \{v_1v_2, v_1v_3\}, \\ \{v_1v_2, v_1v_3, v_2v_3\} \end{array} \right\}$$

Cada elemento de $\mathcal{P}(E)$ deve ser mapeado num grafo com 3 vértices levando a um grafo distinto:

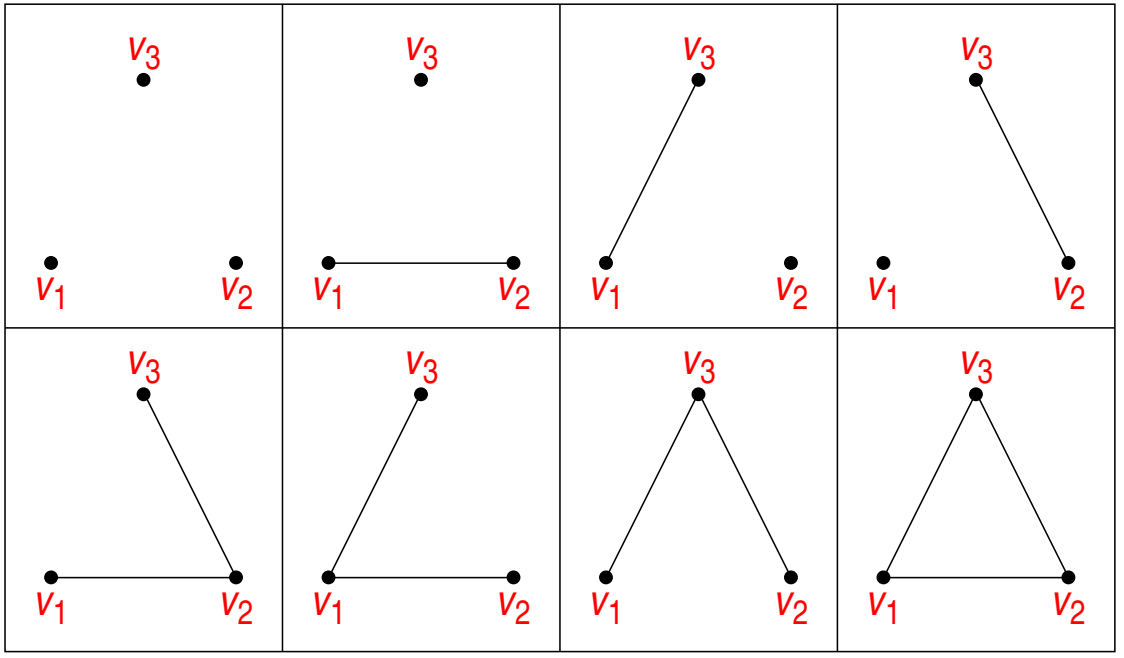


Quantidade de grafos distintos com n vértices (3)

Exemplo: Quantos grafos distintos com 3 vértices existem (continuação)?

- Para cada elemento (sub-conjunto) do conjunto potência de E temos um grafo distinto associado, ou seja, o número total de grafos com 3 vértices é:

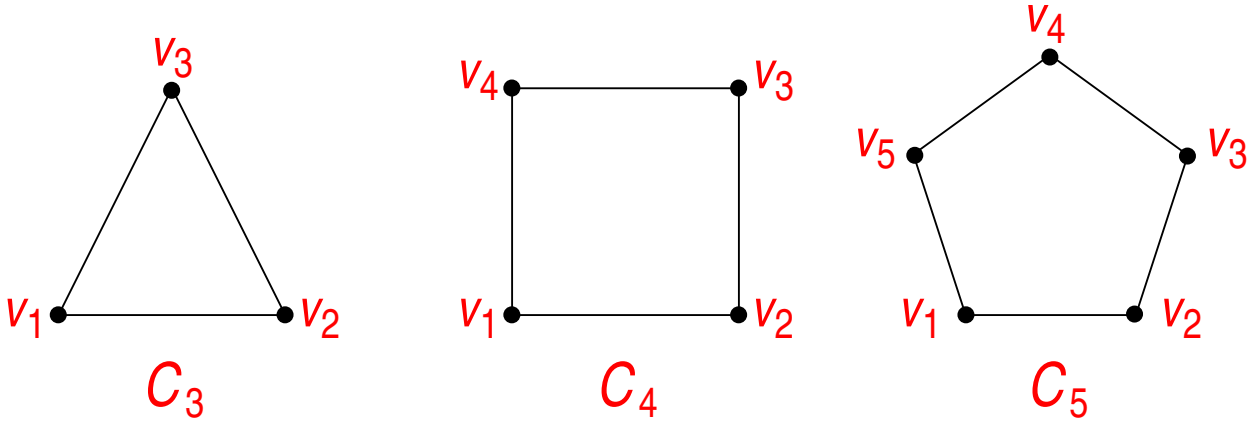
$$2^{\frac{n^2-n}{2}} = 2^{\frac{3^2-3}{2}} = 2^3 = 8$$



Grafo ciclo

Definição: Um grafo ciclo de n vértices, denominado C_n, $n \geq 3$, é um grafo simples com n vértices $v_1, v_2, \ldots, v_n$, e arestas v_1v_2, v_2v_3, $\ldots$, $v_{n-1}v_n$, v_nv_1.

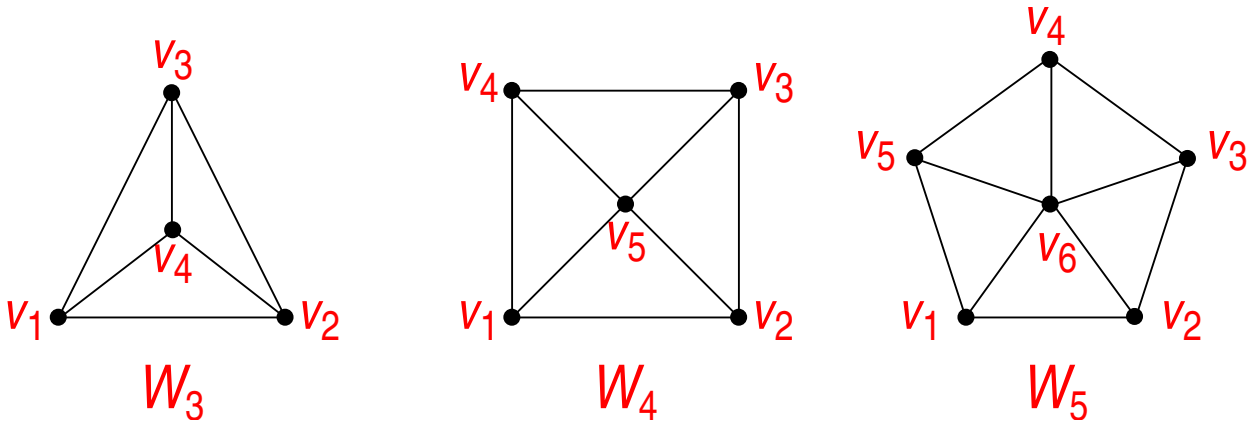
Exemplo: Grafos ciclos de 3, 4, e 5 vértices.



Grafo roda

Definição: Um grafo roda, denominado W_n, é um grafo simples com $n + 1$ vértices que é obtido acrescentado um vértice ao grafo ciclo C_n, $n \geq 3$, e conectando este novo vértice a cada um dos n vértices de C_n.

Exemplo: Grafos rodas de 3, 4, e 5 vértices.



Grafo Cubo-n (1)

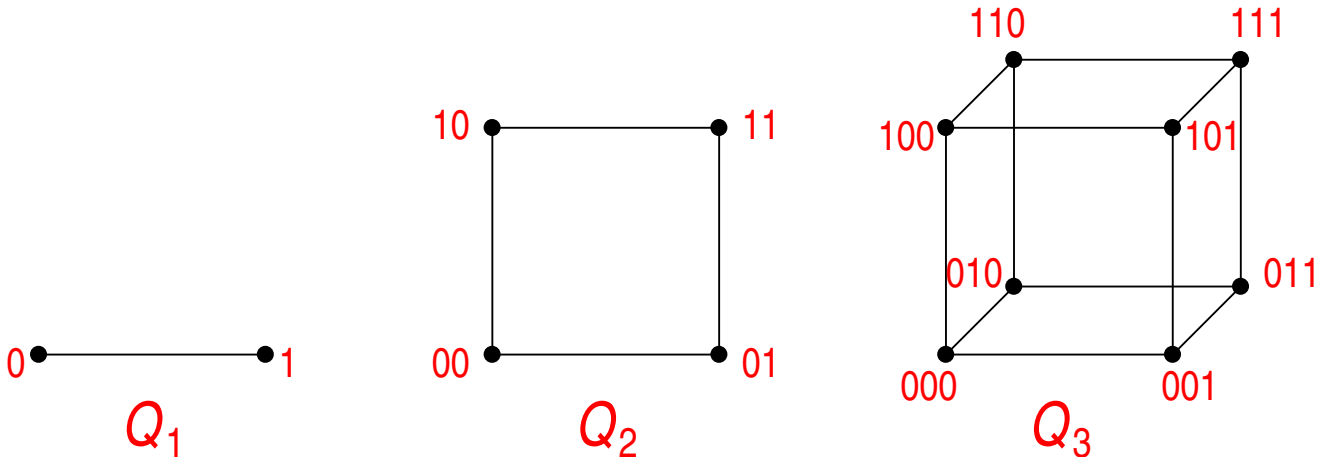
Definição: Um grafo cubo-n de 2^n vértices, denominado Q_n, é um grafo simples que representa os 2^n strings de n bits. Dois vértices são adjacentes sse os strings que eles representam diferem em exatamente uma posição.

O grafo Q_{n+1} pode ser obtido a partir do grafo Q_n usando o seguinte algoritmo:

1. Faça duas cópias de Q_n;
2. Prefixe uma das cópias de Q_n com 0 e a outra com 1;
3. Acrescente uma aresta conectando os vértices que só diferem no primeiro bit.

Grafo Cubo-n (2)

Exemplo: Grafos Q_n, para n = 1, 2, e 3 vértices.



Grafo bipartido (1)

Definição: Um grafo bipartido* de m, n vértices, denominado $K_{m,n}$, é um grafo simples com vértices $v_1, v_2, \ldots, v_m$ e $w_1, w_2, \ldots, w_n$, que satisfaz as seguintes propriedades:

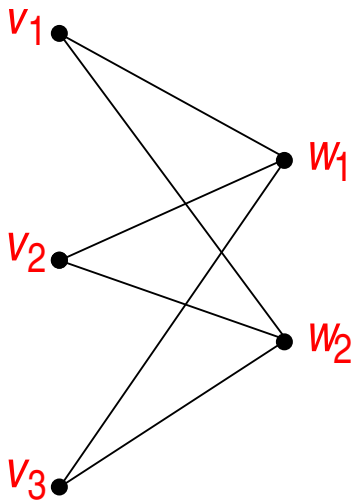
$$\begin{aligned} \forall \quad & i, k = 1, 2, \ldots, m \wedge \\ \forall \quad & j, l = 1, 2, \ldots, n \end{aligned}$$

1. $\exists$ uma aresta entre cada par de vértices v_i e w_j;
2. $\sim\exists$ uma aresta entre cada par de vértices v_i e v_k;
3. $\sim\exists$ uma aresta entre cada par de vértices w_j e w_l;

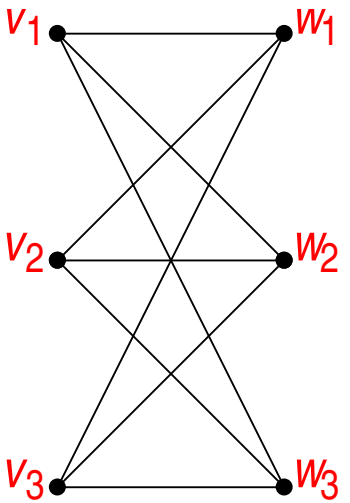
*Também chamado na literatura de grafo bipartido completo.

Grafo bipartido (2)

Exemplo: Grafos bipartidos $K_{3,2}$ e $K_{3,3}$.



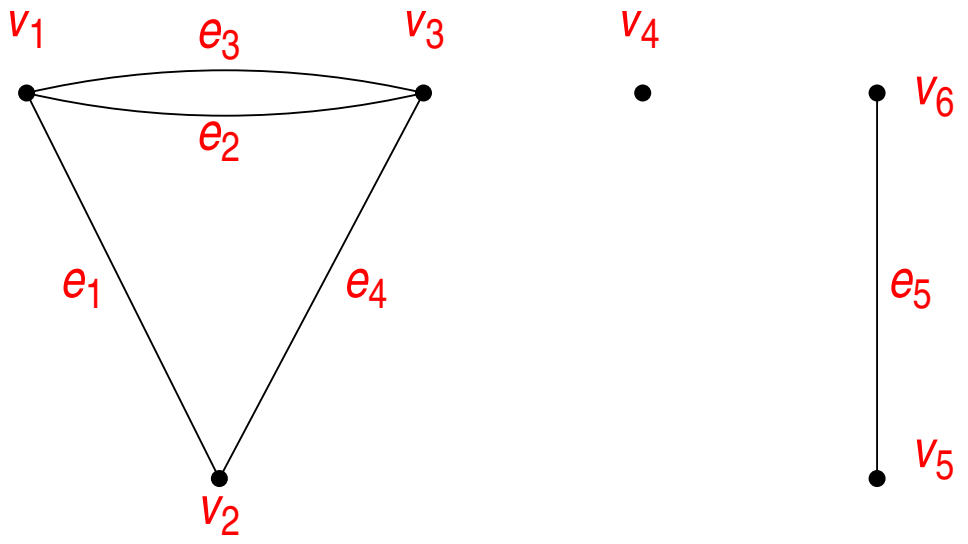
$K_{3,2}$



$K_{3,3}$

Multigrafo

Definição: Um multigrafo é um grafo que não possui laços mas pode ter arestas paralelas. Formalmente, um multigrafo $G = (V, E)$ consiste de um conjunto V de vértices, um conjunto E de arestas, e uma função f de E para $\{\{u, v\}|u, v \in V, u \neq v\}$. As arestas e_1 e e_2 são chamadas múltiplas ou paralelas se $f(e_1) = f(e_2)$.



➜ Várias aplicações precisam ser modeladas como um multigrafo.

Pseudografo

Definição: Um pseudografo é um grafo que pode ter laços e arestas paralelas. Formalmente, um pseudografo $G = (V, E)$ consiste de um conjunto V de vértices, um conjunto E de arestas, e uma função f de E para $\{\{u, v\}|u, v \in V\}$.

➜ Pseudografo é mais geral que um multigrafo.

Multigrafo dirigido

Definição: Um multigrafo dirigido é um grafo que pode ter laços e arestas paralelas. Formalmente, um multigrafo dirigido $G = (V, E)$ consiste de um conjunto V de vértices, um conjunto E de arestas, e uma função f de E para $\{\{u, v\}|u, v \in V\}$. As arestas e_1 e e_2 são arestas múltiplas se $f(e_1) = f(e_2)$.

Hipergrafo

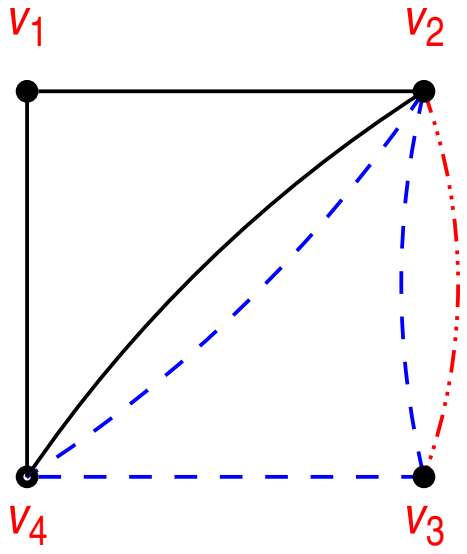
Definição: Um hipergrafo $H(V, F)$ é definido pelo par de conjuntos V e F, onde:

- V é um conjunto não vazio de vértices;
- F é um conjunto que representa uma "família" e partes não vazias de V.

Um hipergrafo é um grafo não dirigido em que cada aresta conecta um número arbitrário de vértices.

Seja, por exemplo, o grafo $H(V, F)$ dado por:

$$V = \{v_1, v_2, v_3, v_4\}$$
$$F = \{\{v_1, v_2, v_4\}, \{v_2, v_3, v_4\}, \{v_2, v_3\}\}$$



Terminologia de grafos

Tipo	Aresta	Arestas múltiplas?	Laços permitidos?
Grafo simples	Não dirigida	Não	Não
Multigrafo	Não dirigida	Sim	Não
Pseudografo	Não dirigida	Sim	Sim
Grafo dirigido	Dirigida	Não	Sim
Multigrafo dirigido	Dirigida	Sim	Sim

Grafo valorado

Definição: Um grafo valorado é um grafo em que cada aresta tem um valor associado. Formalmente, um grafo valorado $G = (V, E)$ consiste de um conjunto V de vértices, um conjunto E de arestas, e uma função f de E para P, onde P representa o conjunto de valores (pesos) associados às arestas.

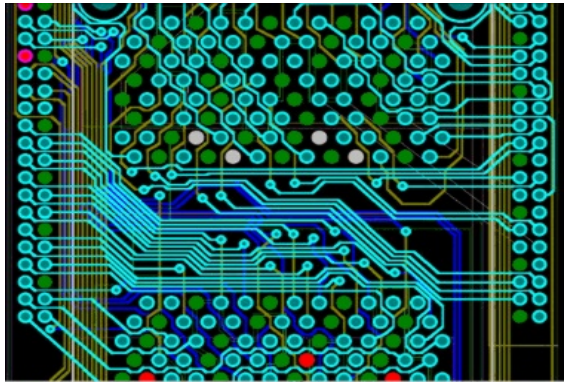
➜ Grafo valorado é usado para modelar vários problemas importantes em Ciência da Computação.

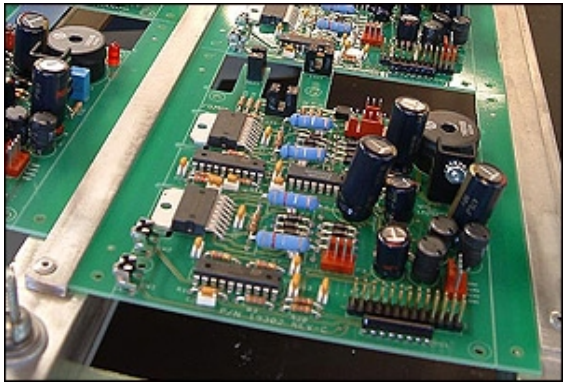
Grafo imersível

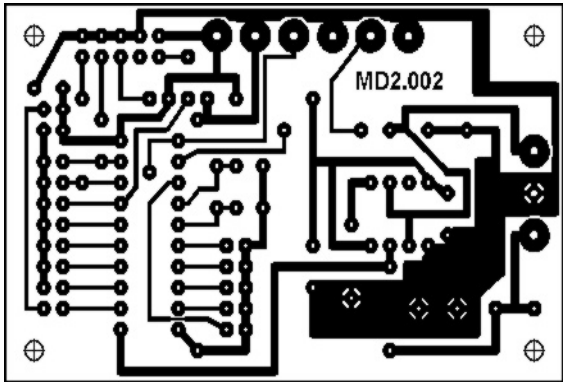
Definição: Um grafo é imersível em uma superfície S se puder ser representado geograficamente em S de tal forma que arestas se cruzem nas extremidades (vértices).

Um **grafo planar** é um grafo que é imersível no plano.

➜ As conexões de uma placa de circuito impresso devem ser representadas por um grafo planar.





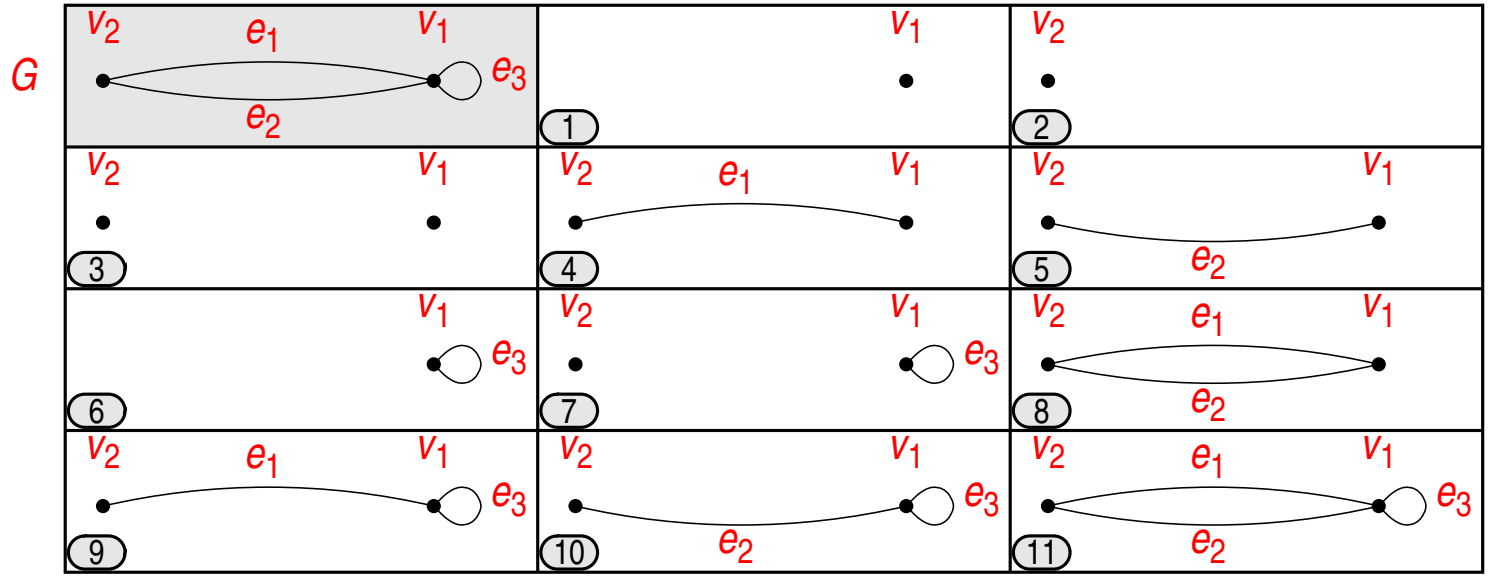


Subgrafo

Definição: Um grafo $H = (V', E')$ é dito ser um subgrafo de um grafo $G = (V, E)$ sse:

- cada vértice de H é também um vértice de G, ou seja, $V' \subseteq V$;
- cada aresta de H é também uma aresta de G, ou seja, $E' \subseteq E$; e
- cada aresta de H tem os mesmos nós terminais em G, ou seja, se $(u, v) \in E'$ então $(u, v) \in E$.

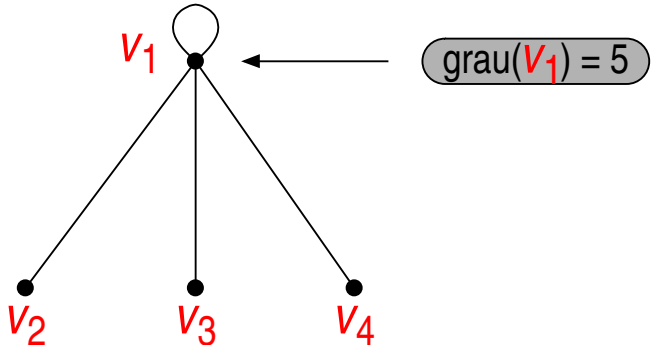
Exemplo: Todos os subgrafos do grafo G:



Grau de um vértice (1)

Definição: Seja G um grafo e um vértice v de G. O grau de v, denominado *grau*(v) (*deg*(v)), é igual ao número de arestas que são incidentes a v, com uma aresta que seja um laço contada duas vezes. O grau total de G é a soma dos graus de todos os vértices de G.

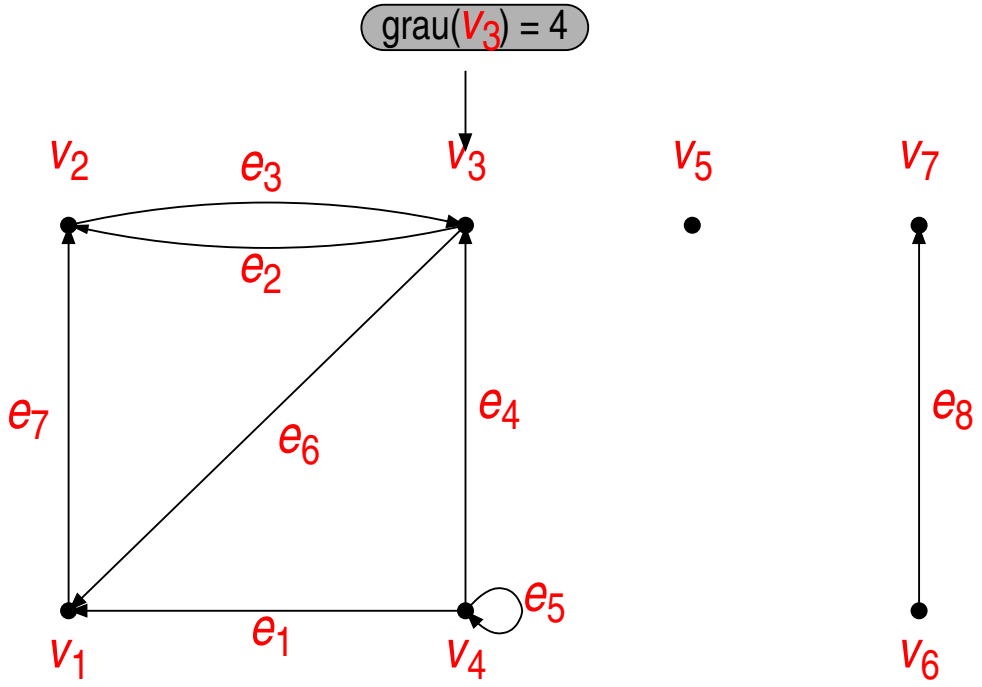
Exemplo: Determinando o grau de v_1 no grafo abaixo.



Grau de um vértice (2)

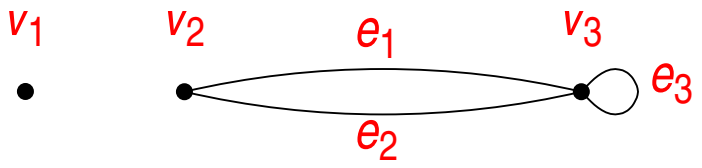
Em um grafo dirigido o grau de um vértice v é o número de arestas quem saem dele (*out-deg*(v)) mais o número de arestas que chegam nele (*in-deg*(v)).

Exemplo: Determinando o grau de v_3 no grafo abaixo.



Grau de um vértice (3)

Exemplo: Seja o grafo G abaixo. Determine o grau de cada vértice e o grau total de G.



- *grau*$(v_1) = 0$, já que não existe aresta incidente a v_1, que é um vértice isolado.
- *grau*$(v_2) = 2$, já que e_1 e e_2 são incidentes a v_2.
- *grau*$(v_3) = 4$, já que e_1, e_2 e e_3 são incidentes a v_3, sendo que e_3 contribui com dois para o grau de v_3.

➜ Grau de G = *grau*(v_1) + *grau*(v_2) + *grau*(v_3) = 0 + 2 + 4 = 6

➜ Grau de G = 2 $\times$ número de arestas de G, que é 3, ou seja, cada aresta contribui com dois para o grau total do grafo.

Grau de um vértice (4)

Teorema (do aperto de mãos ou *handshaking*): Seja G um grafo. A soma dos graus de todos os vértices de G é duas vezes o número de arestas de G. Especificamente, se os vértices de G são $v_1, v_2, \ldots, v_n$, onde n é um inteiro positivo, então

$$\begin{aligned} \text{Grau de } G &= \textit{grau}(v_1) + \textit{grau}(v_2) + \ldots + \textit{grau}(v_n) \\ &= 2 \times \text{número de arestas de } G. \end{aligned}$$

Grau de um vértice (5)

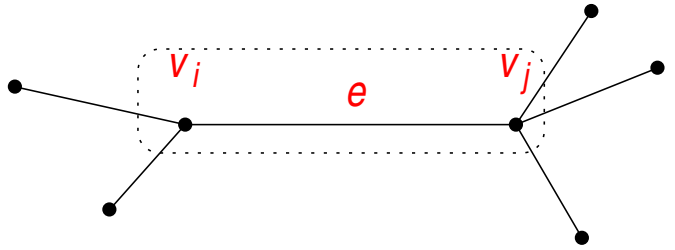
Prova:

- Seja G um grafo arbitrário mas escolhido arbitrariamente.
- Se G não possui vértices então não possui arestas, e o grau total é 0, que é o dobro das arestas, que é 0.
- Se G tem n vértices $v_1, v_2, \ldots, v_n$ e m arestas, onde n é um inteiro positivo e m é um inteiro não negativo. A hipótese é que cada aresta de G contribui com 2 para o grau total de G.
- Suponha que e seja uma aresta arbitrária com extremidades v_i e v_j. Esta aresta contribui com 1 para o grau de v_i e 1 para o grau de v_j.

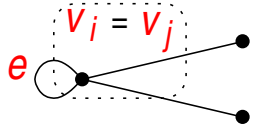
Grau de um vértice (6)

Prova (continuação):

- Isto é verdadeiro mesmo se $i = j$ já que no caso de um laço conta-se duas vezes para o grau do vértice no qual incide.



$i \neq j$



$i = j$

- Assim, a aresta e contribui com 2 para o grau total de G. Como e foi escolhido arbitrariamente, isto mostra que cada aresta de G contribui com 2 para o grau total de G.

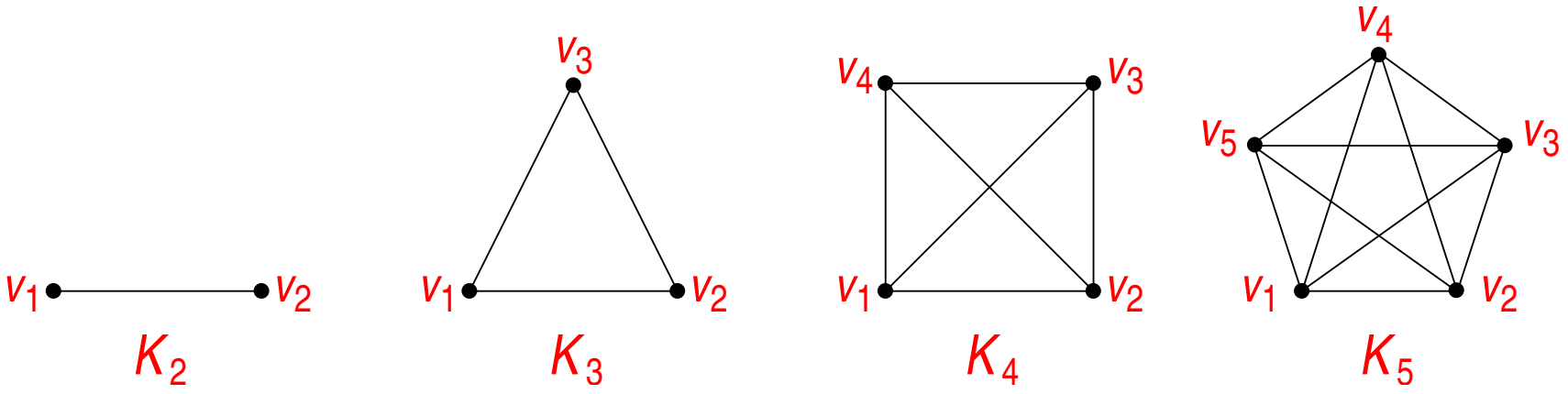
$\therefore$ O grau total de $G = 2\times$ número de arestas de G.

➜ Corolário: O grau total de um grafo é par.

Grafo regular

Definição: Um grafo é dito ser regular quando todos os seus vértices têm o mesmo grau.

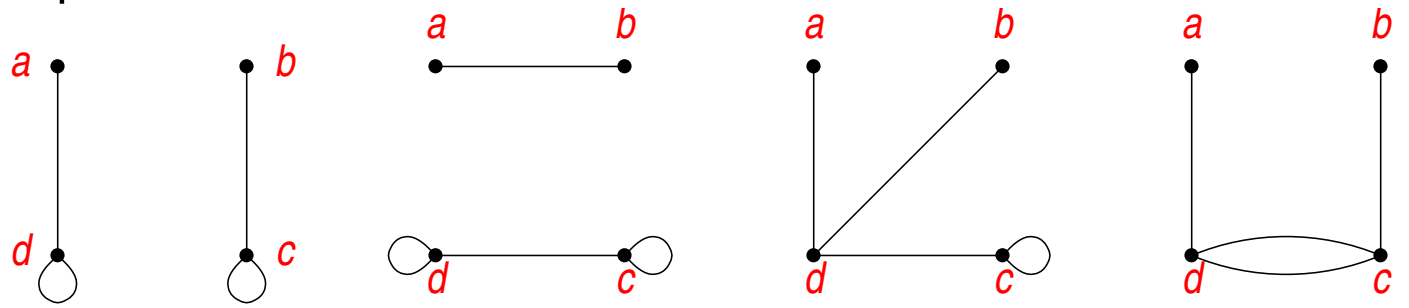
Exemplo: Os grafos completos com 2, 3, 4, e 5 vértices são grafos regulares.



Determinando a existência de certos grafos (1)

- É possível ter um grafo com quatro vértices de graus 1, 1, 2, e 3?
  Não. O grau total deste grafo é 7, que é um número ímpar.

- É possível ter um grafo com quatro vértices de graus 1, 1, 3, e 3?
  Sim. Exemplos:

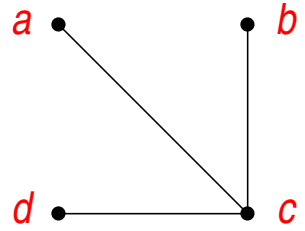


Determinando a existência de certos grafos (2)

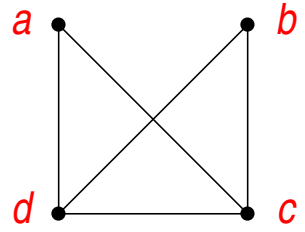
- É possível ter um grafo simples com quatro vértices de graus 1, 1, 3, e 3?
  Não.

  Prova (por contradição):

  – Suponha que exista um grafo simples G com quatro vértices de graus 1, 1, 3, e 3. Chame a e b os vértices de grau 1, e c e d os vértices de grau 3. Como *grau*$(c) = 3$ e G não possui laços ou arestas paralelas, devem existir arestas que conectam c aos vértices a, b e d.



  – Pelo mesmo raciocínio devem existir arestas que conectam d aos vértices a, b e c.



  – Mas o *grau*$(a) \geq 2$ e *grau*$(b) \geq 2$, o que contradiz a suposição que estes vértices têm grau 1.

  ∴ A suposição inicial é falsa e, conseqüentemente, não existe um grafo simples com quatro vértices com graus 1, 1, 3, e 3.

Determinando a existência de certos grafos (3)

- É possível num grupo de nove pessoas, cada um ser amigo de exatamente cinco outras pessoas?
  Não.

  Prova (por contradição):
  - Suponha que cada pessoa represente um vértice de um grafo e a aresta indique uma relação de amizade entre duas pessoas (vértices).
  - Suponha que cada pessoa seja amiga de exatamente cinco outras pessoas.
  - Então o grau de cada vértice é cinco e o grau total do grafo é 45.

  $\therefore$ Isto contradiz o corolário que o grau total de um grafo é par e, conseqüentemente, a suposição é falsa.

Característica de um grafo

Teorema: Em qualquer grafo G, existe um número par de vértices de grau ímpar.

Prova:

- Suponha que G tenha n vértices de grau ímpar e m vértices de grau par, onde n e m são inteiros não negativos. [Deve-se mostrar que n é par.]
- Se $n = 0$, então G tem um número par de vértices de grau ímpar.
- Suponha que $n \geq 1$. Seja P a soma dos graus de todos os vértices de grau par, I a soma dos graus de todos os vértices de grau ímpar, e T o grau total de G.
- Se $p_1, p_2, \ldots, p_m$ são os vértices de grau par e $i_1, i_2, \ldots, i_n$ são os vértices de grau ímpar,

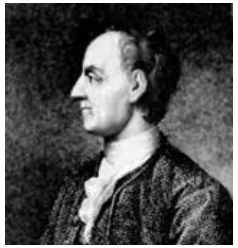
$$
\begin{aligned}
P &= \mathit{grau}(p_1) + \mathit{grau}(p_2) + \ldots + \mathit{grau}(p_m), \\
I &= \mathit{grau}(i_1) + \mathit{grau}(i_2) + \ldots + \mathit{grau}(i_n), \\
T &= \mathit{grau}(p_1) + \mathit{grau}(p_2) + \ldots + \mathit{grau}(p_m) + \\
&\quad \mathit{grau}(i_1) + \mathit{grau}(i_2) + \ldots + \mathit{grau}(i_n) \\
&= P + I \quad \text{[que deve ser um número par]}
\end{aligned}
$$

- P é par, já que $P = 0$ ou P é a soma de $\mathit{grau}(p_r)$, $0 \leq r \leq m$, que é par.
- Mas $T = P + I$ e $I = T - P$. Assim, I é a diferença de dois inteiros pares, que é par.
- Pela suposição, $\mathit{grau}(i_s)$, $0 \leq s \leq n$, é ímpar. Assim, I, um inteiro par, é a soma de n inteiros ímpares $\mathit{grau}(i_1) + \mathit{grau}(i_2) + \ldots + \mathit{grau}(i_n)$. Mas a soma de n inteiros ímpares é par, então n é par [o que devia ser mostrado].

Determinando a existência de certos grafos (4)

- É possível ter um grafo com 10 vértices de graus 1, 1, 2, 2, 2, 3, 4, 4, 4, e 6?
  Não. Duas formas de provar:
  1. Este grafo supostamente possui três vértices de grau ímpar, o que não é possível.
  2. Este grafo supostamente possui um grau total = 29, o que não é possível.

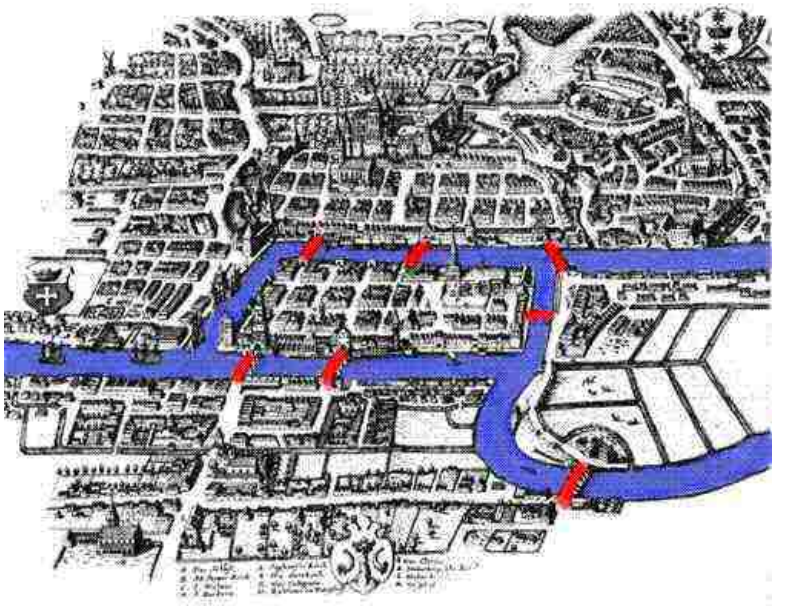
O problema das sete pontes de Königsberg ou O início da teoria dos grafos (1)



Leonard Euler (1707–1783), matemático suíço. Foi um cientista extremamente produtivo contribuindo para muitas áreas da matemática como teoria dos números, análise combinatória e análise, bem como o seu uso em áreas como música e arquitetura naval. No total escreveu mais de 1100 artigos e livros.

A área de teoria dos grafos começa em 1736 quando publica um artigo contendo a solução para o problema das sete pontes de Königsberg, na época uma cidade da Prússia e, atualmente, cidade da Rússia.

A cidade de Königsberg foi construída numa região onde haviam dois braços do Rio Pregel e uma ilha. Foram construídas sete pontes ligando diferentes partes da cidade, como mostrado na figura:



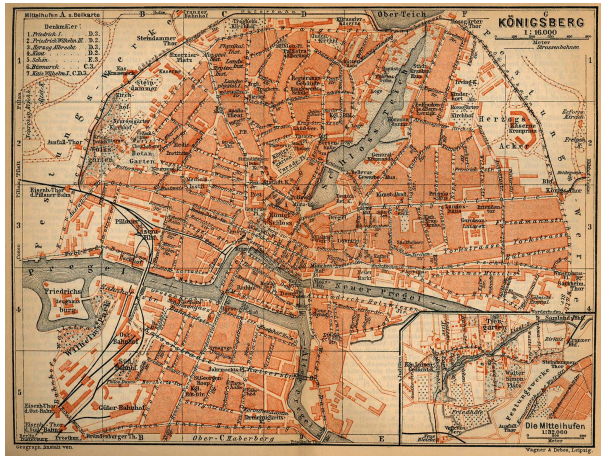
Problema: É possível que uma pessoa faça um percurso na cidade de tal forma que inicie e volte a mesma posição passando por todas as pontes somente uma única vez?

O problema das sete pontes de Königsberg ou O início da teoria dos grafos (2)

Referência: "Northern Germany as far as the Bavarian and Austrian Frontiers; Handbook for Travellers" by Karl Baedeker. Fifteenth Revised Edition. Leipzig, Karl Baedeker; New York, Charles Scribner's Sons 1910.

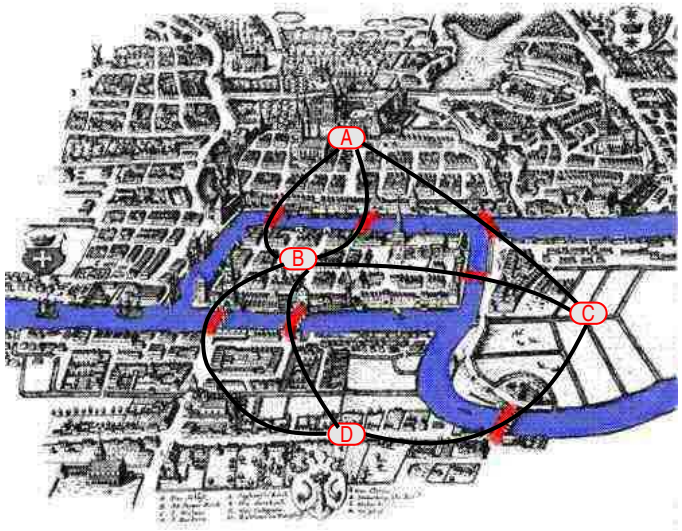
History: Kaliningrad was formerly the Prussian port of Königsberg, capital of East Prussia. It was captured by the Red Army in April 1945 and ceded to the Soviet Union at the Potsdam conference. It was renamed in honor of senior Soviet leader Mikhail Kalinin, although he never actually visited the area.

Mapa parcial (recente) da cidade.



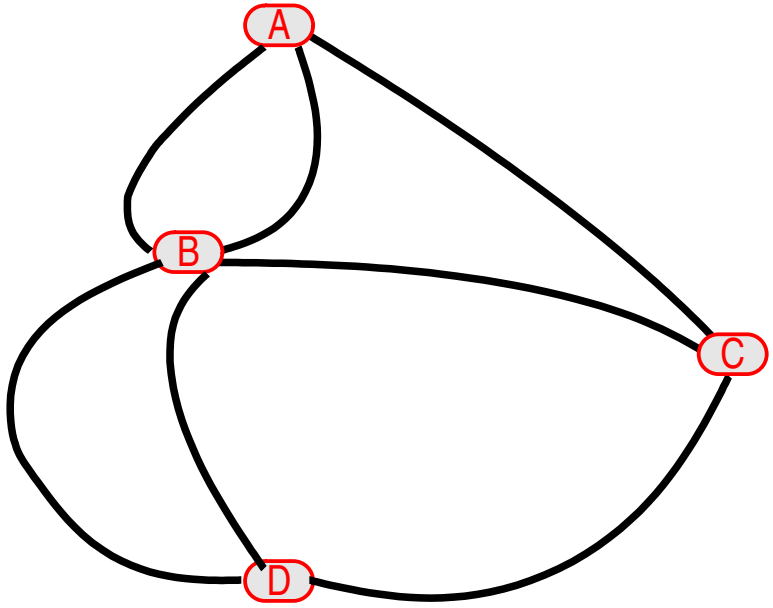
O problema das sete pontes de Königsberg (3)

- Euler resolveu este problema transformando-o em teoria dos grafos.
- Modelagem proposta por Euler:
  - Todos os “pontos” de uma dada área de terra podem ser representados por um único ponto já que uma pessoa pode andar de um lado para o outro sem atravessar uma ponte.
  - Um ponto é conectado a outro se houver uma ponte de um lado para o outro.
  - Graficamente, Euler representou o problema como:



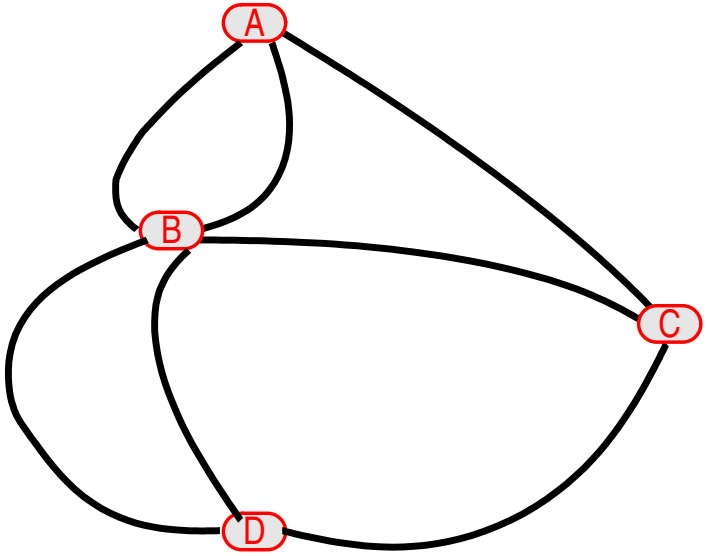
O problema das sete pontes de Königsberg (4)

- Problema a ser resolvido:
  - É possível achar um caminho que comece e termine num vértice qualquer (A, B, C, ou D) e passe por cada aresta, exatamente, e uma única vez?, ou ainda,
  - É possível desenhar este grafo que comece e termine na mesma posição sem levantar o lápis do papel?



O problema das sete pontes de Königsberg (5)

- Aparentemente não existe solução!
- Partindo do vértice A, toda vez que se passa por qualquer outro vértice, duas arestas são usadas: a de "chegada" e a de "saída".
- Assim, se for possível achar uma rota que usa todas as arestas do grafo e começa e termina em A, então o número total de "chegadas" e "saídas" de cada vértice deve ser um valor múltiplo de 2.
- No entanto, temos:
  - $grau(A) = grau(C) = grau(D) = 3$; e
  - $grau(B) = 5$.
- Assim, por este raciocínio informal não é possível ter uma solução para este problema.



Caminhamentos em grafos
Caminho (1)

Seja G um grafo não dirigido, $n \geq 1$, e v e w vértices de G.

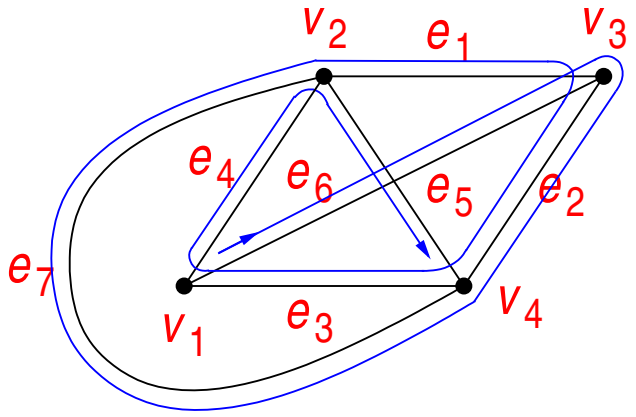
Caminho (*walk*): Um caminho de v para w é uma seqüência alternada de vértices e arestas adjacentes de G. Um caminho tem a forma:

$$(v =)v_0 e_1 v_1 e_2 v_2 \ldots v_{n-1} e_n v_n (= w)$$

ou ainda

$$v_0 [v_0, v_1] v_1 [v_1, v_2] v_2 \ldots v_{n-1} [v_{n-1}, v_n] v_n$$

onde $v_0 = v$ e $v_n = w$.



Um possível caminho entre v_1 e v_4:

$v_1 e_6 v_3 e_2 v_4 e_7 v_2 e_1 v_3 e_2 v_4 e_3 v_1 e_4 v_2 e_5 v_4$

Caminhamentos em grafos
Caminho (2)

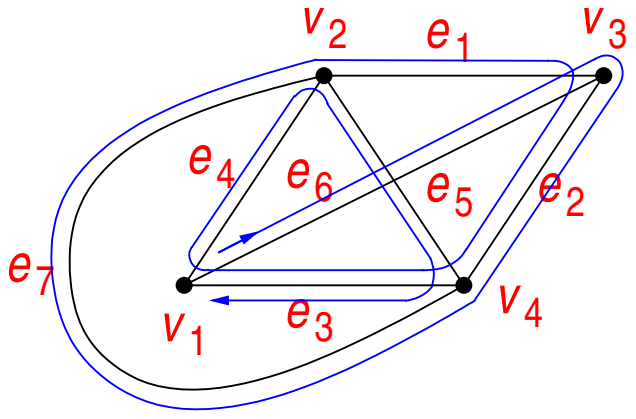
- No caso de arestas múltiplas, deve-se indicar qual delas está sendo usada.
- Vértices v_0 e v_n são extremidades do caminho.
- Tamanho (comprimento) do caminho: número de arestas do mesmo, ou seja, número de vértices menos um.
- O **caminho trivial** de v para v consiste apenas do vértice v.
- Se existir um caminho c de v para w então w é alcançável a partir de v via c.

Caminhamentos em grafos
Caminho fechado (1)

Caminho fechado (*Closed walk*): Caminho que começa e termina no mesmo vértice:

$$(v =)v_0 e_1 v_1 e_2 v_3 \ldots v_{n-1} e_n v_n (= w)$$

onde $v = w$.



Um possível caminho fechado é:

$v_1 e_6 v_3 e_2 v_4 e_7 v_2 e_1 v_3 e_2 v_4 e_3 v_1 e_4 v_2 e_5 v_4 e_3 v_1$

Um caminho fechado com pelo menos uma aresta é chamado de **ciclo**.

Caminhamentos em grafos
Caminho fechado (2)

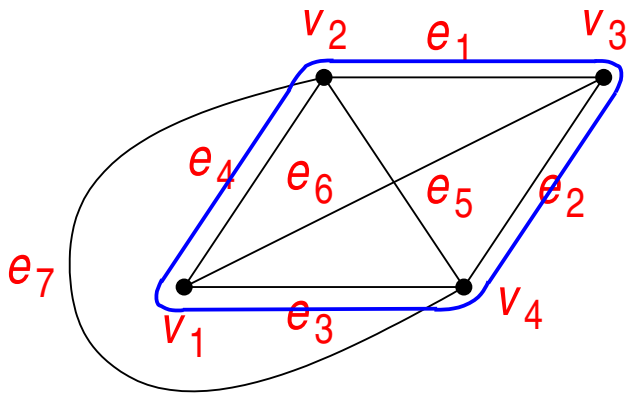
Dois caminhos fechados

$$v_0 v_1 \ldots v_n \qquad \text{e} \qquad v'_0 v'_1 \ldots v'_n$$

formam o mesmo ciclo se existir um inteiro j tal que

$$v'_i = v_{i+j} \bmod n,$$

para $i = 0, 1, \ldots, n-1$.



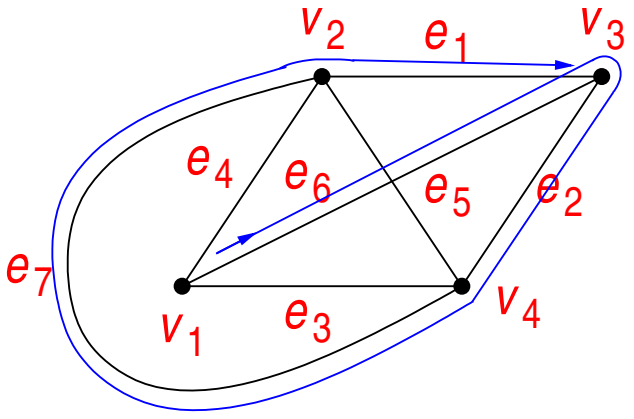
O caminho fechado $v_1 v_2 v_3 v_4 v_1$ forma o mesmo ciclo que os caminhos fechados $v_2 v_3 v_4 v_1 v_2$, $v_3 v_4 v_1 v_2 v_3$ e $v_4 v_1 v_2 v_3 v_4$.

Caminhamentos em grafos
Trajeto

Trajeto (*Path*): Caminho de v para w sem arestas repetidas:

$$(v =)v_0 e_1 v_1 e_2 v_3 \ldots v_{n-1} e_n v_n (= w)$$

onde todas as arestas e_i são distintas, ou seja, $e_i \neq e_k$, para qualquer $i \neq k$.

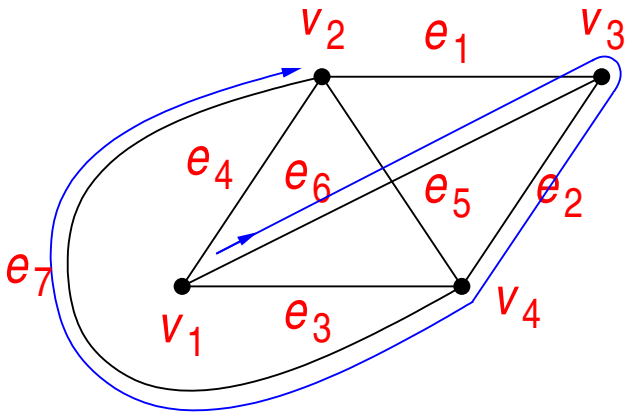


Um possível trajeto é:

$v_1 e_6 v_3 e_2 v_4 e_7 v_2 e_1 v_3$

Caminhamentos em grafos
Trajeto simples

Trajeto simples (*Simple path*): Caminho de v para w sem arestas e vértices repetidos.



Um possível trajeto simples é:

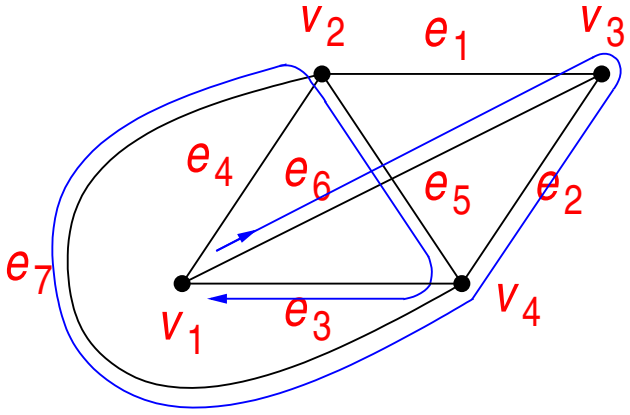
$v_1 e_6 v_3 e_2 v_4 e_7 v_2$

Caminhamentos em grafos
Circuito

Circuito (*Circuit*): Trajeto fechado, ou seja, um caminho onde não há aresta repetida e os vértices inicial e final são idênticos:

$$(v =)v_0 e_1 v_1 e_2 v_3 \ldots v_{n-1} e_n v_n (= w)$$

onde toda aresta e_i, $1 \leq i \leq n$, é distinta e $v_0 = v_n$.

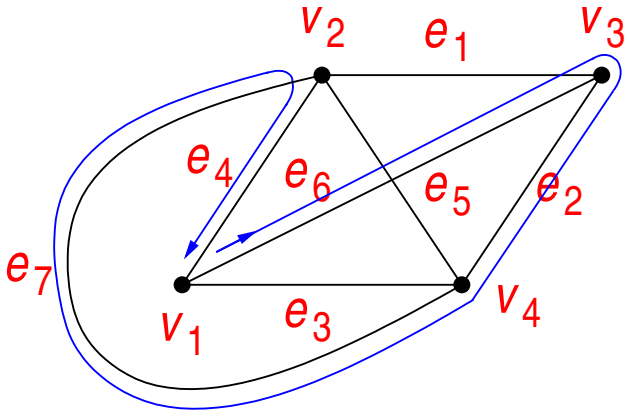


Um possível circuito é:

$v_1 e_6 v_3 e_2 v_4 e_7 v_2 e_1 v_3$

Caminhamentos em grafos
Circuito simples

Circuito simples (*Simple circuit*): Trajeto fechado, ou seja, um caminho onde não há arestas e vértices repetidos, exceto os vértices inicial e final que são idênticos.



Um possível circuito simples é:

$v_1 e_6 v_3 e_2 v_4 e_7 v_2 e_4 v_1$

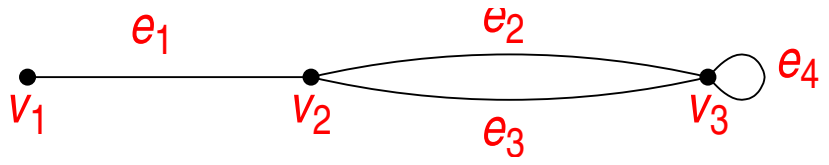
Um circuito simples também é chamado de ciclo simples.

Terminologia de caminhamentos

Tipo	Aresta repetida?	Vértice repetido?	Começa e termina no mesmo vértice?
Caminho (*walk*)	Pode	Pode	Pode
Caminho fechado (*closed walk*)	Pode	Pode	Sim
Trajeto (*path*)	Não	Pode	Pode
Trajeto simples (*simple path*)	Não	Não	Não
Circuito (*circuit*)	Não	Pode	Sim
Circuito simples (*simple circuit*)	Não	$v_0 = v_n$	Sim

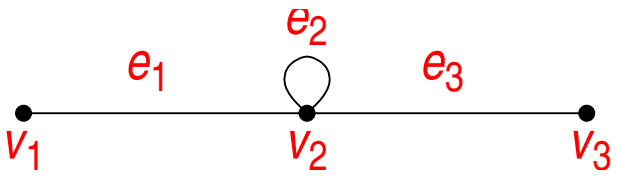
Caminhamentos em grafos
Notação simplificada (1)

Em geral um caminho pode ser identificado de forma não ambígua através de uma seqüência de arestas ou vértices.



- O caminho $e_1e_2e_4e_3$ representa de forma não ambígua o caminho $v_1e_1v_2e_2v_3e_4v_3e_3v_2$
- A notação e_1 é ambígua, se usada para referenciar um caminho, pois pode representar duas possibilidades: $v_1e_1v_2$ ou $v_2e_1v_1$.
- A notação v_2v_3 é ambígua, se usada para referenciar um caminho, pois pode representar duas possibilidades: $v_2e_2v_3$ ou $v_2e_3v_3$.

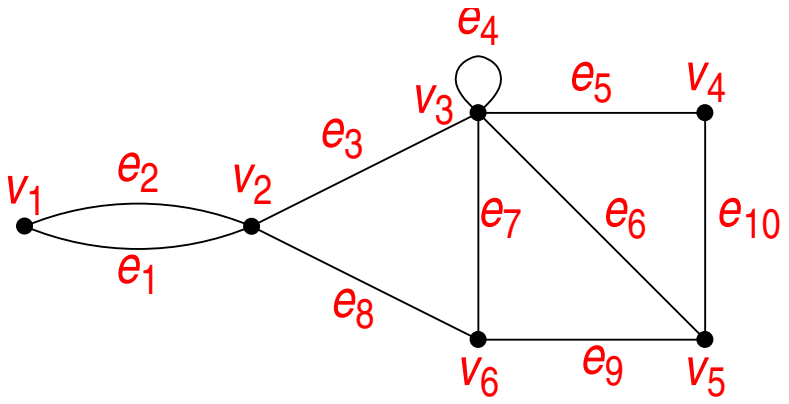
Caminhamentos em grafos
Notação simplificada (2)



- A notação $v_1v_2v_2v_3$, se for associada a um caminho, representa de forma não ambígua o caminho $v_1e_1v_2e_2v_2e_3v_3$

➜ Se um grafo G não possui arestas paralelas, então qualquer caminho em G pode ser determinado de forma única por uma seqüência de vértices.

Identificando o caminhamento (1)

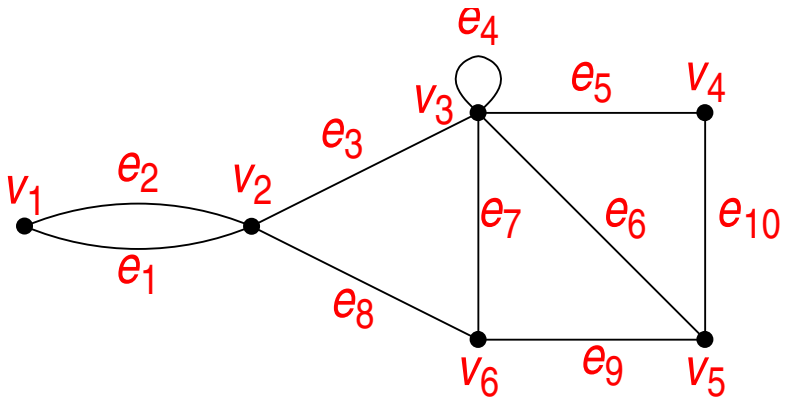


Tipo	Aresta repetida?	Vértice repetido?	Começa e termina no mesmo vértice?
Caminho (*walk*)	Pode	Pode	Pode
Caminho fechado (*closed walk*)	Pode	Pode	Sim
Trajeto (*path*)	Não	Pode	Pode
Trajeto simples (*simple path*)	Não	Não	Não
Circuito (*circuit*)	Não	Pode	Sim
Circuito simples (*simple circuit*)	Não	$v_0 = v_n$	Sim

Que tipo de caminhamento é?

- $v_1 e_1 v_2 e_3 v_3 e_4 v_3 e_5 v_4$
  - Aresta repetida? Não.
  - Vértice repetido? Sim – v_3.
  - Começa e termina no mesmo vértice? Não.
  - ➜ Trajeto.
- $e_1 e_3 e_5 e_5 e_6$
  - Aresta repetida? Sim – e_5.
  - Vértice repetido? Sim – v_3.
  - Começa e termina no mesmo vértice? Não.
  - ➜ Caminho.

Identificando o caminhamento (2)

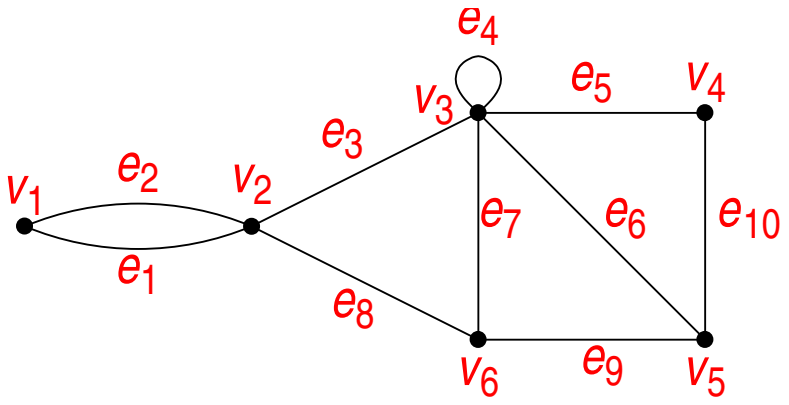


Tipo	Aresta repetida?	Vértice repetido?	Começa e termina no mesmo vértice?
Caminho (*walk*)	Pode	Pode	Pode
Caminho fechado (*closed walk*)	Pode	Pode	Sim
Trajeto (*path*)	Não	Pode	Pode
Trajeto simples (*simple path*)	Não	Não	Não
Circuito (*circuit*)	Não	Pode	Sim
Circuito simples (*simple circuit*)	Não	$v_0 = v_n$	Sim

Que tipo de caminhamento é?

- $v_2v_3v_4v_5v_3v_6v_2$
  - Aresta repetida? Não.
  - Vértice repetido? Sim – v_2 e v_3.
  - Começa e termina no mesmo vértice? Sim – v_2.
  - ➜ Circuito.
- $v_2v_3v_4v_5v_6v_2$
  - Aresta repetida? Não.
  - Vértice repetido? Sim – v_2.
  - Começa e termina no mesmo vértice? Sim – v_2.
  - ➜ Circuito simples.

Identificando o caminhamento (3)



Tipo	Aresta repetida?	Vértice repetido?	Começa e termina no mesmo vértice?
Caminho (*walk*)	Pode	Pode	Pode
Caminho fechado (*closed walk*)	Pode	Pode	Sim
Trajeto (*path*)	Não	Pode	Pode
Trajeto simples (*simple path*)	Não	Não	Não
Circuito (*circuit*)	Não	Pode	Sim
Circuito simples (*simple circuit*)	Não	$v_0 = v_n$	Sim

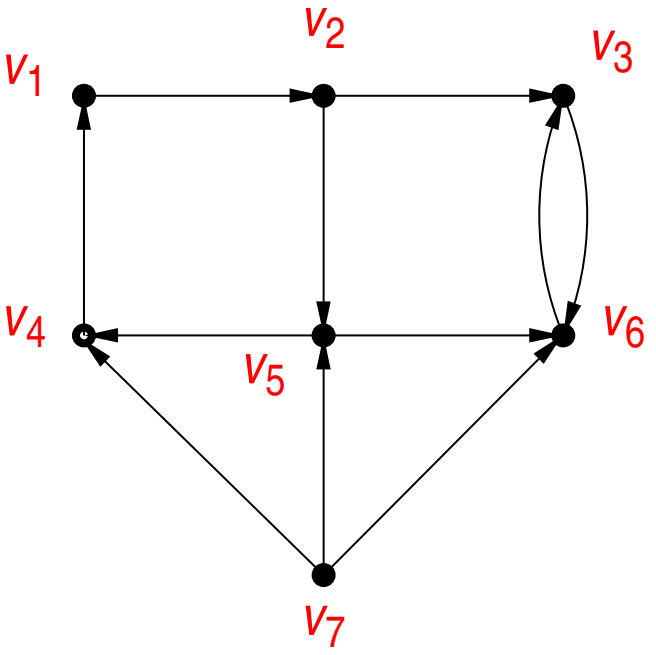
Que tipo de caminhamento é?

- $v_2v_3v_4v_5v_6v_3v_2$
  - Aresta repetida? Sim – e_3.
  - Vértice repetido? Sim – v_2 e v_3.
  - Começa e termina no mesmo vértice? Sim – v_2.
  - ➜ Caminho fechado.
- v_1
  - Aresta repetida? Não.
  - Vértice repetido? Não.
  - Começa e termina no mesmo vértice? Sim – v_1.
  - ➜ Caminho (circuito) trivial.

Fecho transitivo direto

Definição: O fecho transitivo direto (FTD) de um vértice v é o conjunto de todos os vértices que podem ser atingidos por algum caminho iniciando em v.

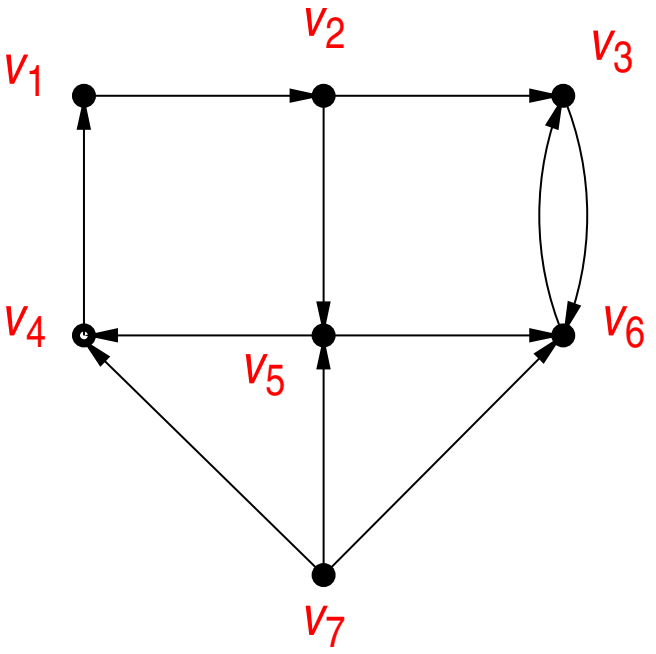
Exemplo: O FTD do vértice v_5 do grafo ao lado é o conjunto $\{v_1, v_2, v_3, v_4, v_5, v_6\}$. Note que o próprio vértice faz parte do FTD já que ele é alcançável partindo-se dele mesmo.



Fecho transitivo inverso

Definição: O fecho transitivo inverso (FTI) de um vértice v é o conjunto de todos os vértices a partir dos quais se pode atingir v por algum caminho.

Exemplo: O FTI do vértice v_5 do grafo abaixo é o conjunto $\{v_1, v_2, v_4, v_5, v_7\}$. Note que o próprio vértice faz parte do FTI já que dele pode alcançar ele mesmo.



Conectividade (1)

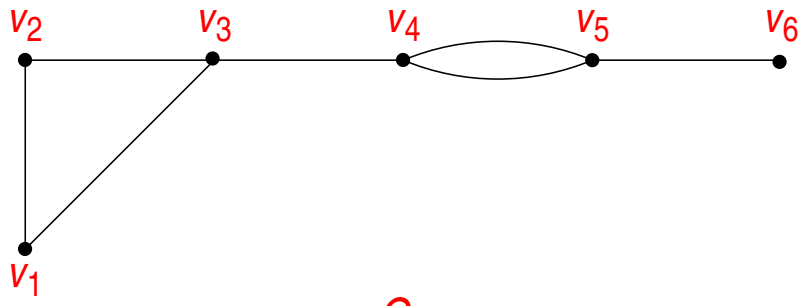
Informalmente um grafo é **conexo** (conectado) se for possível caminhar de qualquer vértice para qualquer outro vértice através de uma seqüência de arestas adjacentes.

Definição: Seja G um grafo. Dois vértices v e w de G estão **conectados** sse existe um caminho de v para w. Um grafo G é conexo sse dado um par qualquer de vértice v e w em G, existe um caminho de v para w. Simbolicamente,

$$G \text{ é conexo } \Leftrightarrow \forall \text{ vértices } v, w \in V(G), \exists \text{ um caminho de } v \text{ para } w.$$

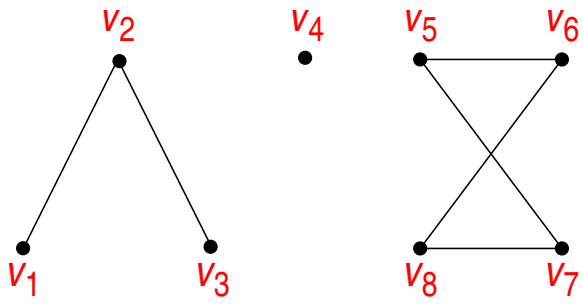
Se a negação desta afirmação for tomada, é possível ver que um grafo não é conexo sse existem dois vértices em G que não estão conectados por qualquer caminho.

Conectividade (2)

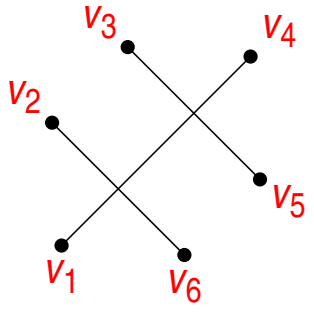


G_1

Grafo conexo.



G_2



G_3

Grafos não conexos.

Conectividade (3)
Lemas

Seja G um grafo.

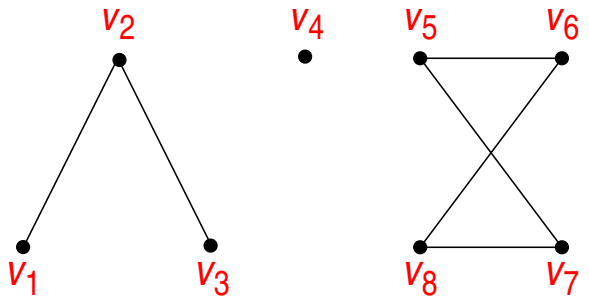
(a) Se G é conexo, então quaisquer dois vértices distintos de G podem ser conectados por um trajeto simples (*simple path*).

(b) Se vértices v e w são parte de um circuito de G e uma aresta é removida do circuito, ainda assim existe um trajeto de v para w em G.

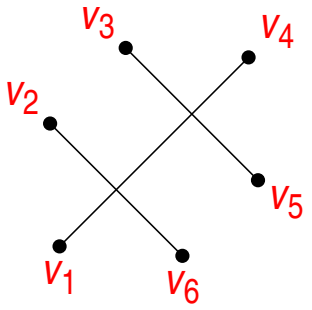
(c) Se G é conexo e contém um circuito, então uma aresta do circuito pode ser removida sem desconectar G.

Conectividade (4)

Os grafos



G_2



G_3

possuem três “partes” cada um, sendo cada parte um grafo conexo.

Um **componente conexo** de um grafo é um subgrafo conexo de maior tamanho possível.

Componente conexo (1)

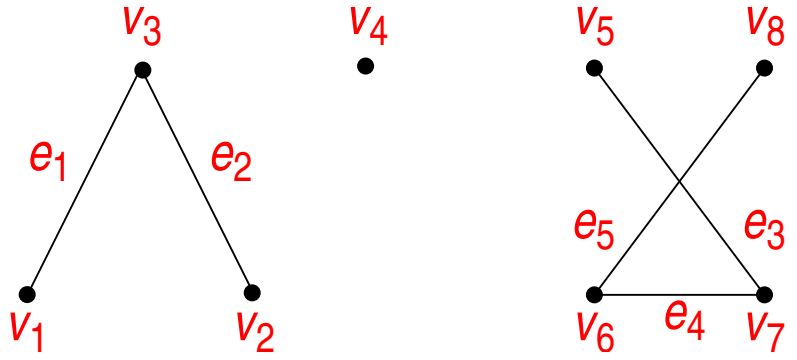
Definição: Um grafo H é um componente conexo de um grafo G sse:

1. H é um subgrafo de G;
2. H é conexo;
3. Nenhum subgrafo conexo de G tem H como um subgrafo e contém vértices ou arestas que não estão em H.

➜ Um grafo pode ser visto como a união de seus componentes conexos.

Componente conexo (2)

Os componentes conexos do grafo G abaixo são:



G possui três componentes conexos:

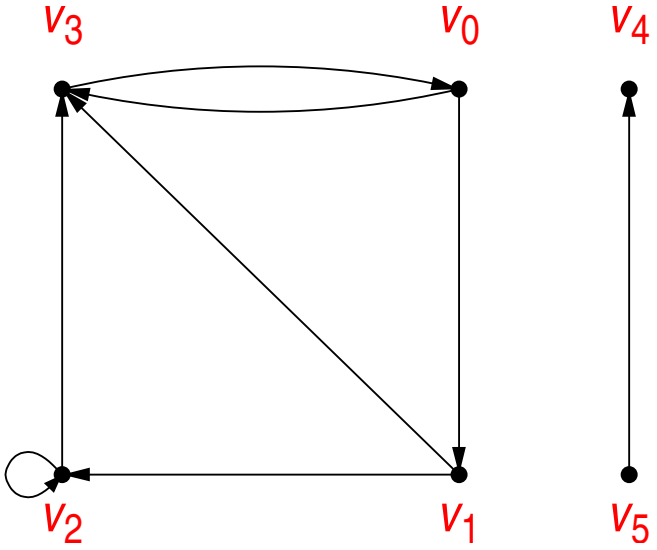
$$
\begin{array}{lll}
H_1: & V_1 = \{v_1, v_2, v_3\} & E_1 = \{e_1, e_2\} \\
H_2: & V_2 = \{v_4\} & E_2 = \emptyset \\
H_3: & V_3 = \{v_5, v_6, v_7, v_8\} & E_3 = \{e_3, e_4, e_5\}
\end{array}
$$

Componente fortemente conexo (conectado)

Um grafo dirigido $G = (V, E)$ é **fortemente conectado** se cada dois vértices quaisquer são alcançáveis a partir um do outro.

Os componentes fortemente conectados de um grafo direcionado são conjuntos de vértices sob a relação "são mutuamente alcançáveis".

Um **grafo dirigido fortemente conectado** tem apenas um componente fortemente conectado.



Os componentes fortemente conectados do grafo ao lado são:

$$H_1: \quad V_1 = \{v_0, v_1, v_2, v_3\}$$
$$H_2: \quad V_2 = \{v_4\}$$
$$H_3: \quad V_3 = \{v_5\}$$

Observe que $\{v_4, v_5\}$ não é um componente fortemente conectado já que o vértice v_5 não é alcançável a partir do vértice v_4.

Circuito Euleriano (1)

Definição: Seja G um grafo. Um **circuito Euleriano** é um circuito que contém cada vértice e cada aresta de G. É uma seqüência de vértices e arestas adjacentes que começa e termina no mesmo vértice de G, passando pelo menos uma vez por cada vértice e exatamente uma única vez por cada aresta de G.

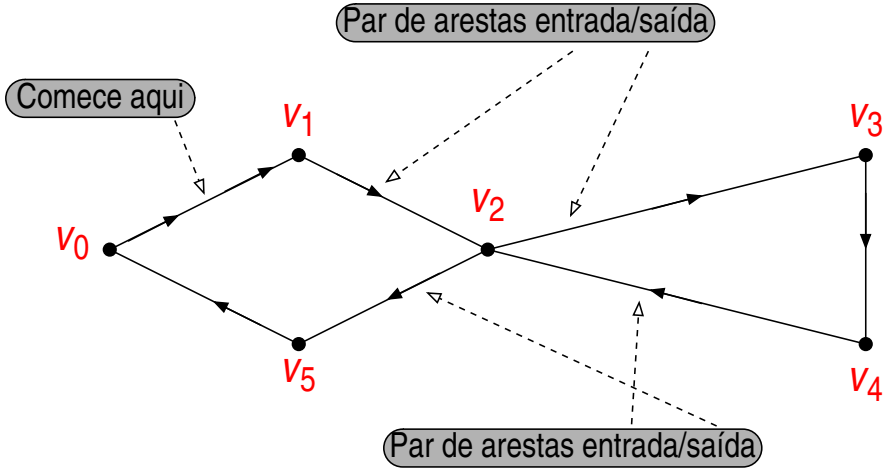
Circuito Euleriano (2)

Teorema: Se um grafo possui um circuito Euleriano, então cada vértice do grafo tem grau par.

Prova:

- Suponha que G é um grafo que tem um circuito Euleriano. [Deve-se mostrar que qualquer vértice v de G tem grau par.]
- Seja v um vértice particular de G mas escolhido aleatoriamente.
- O circuito Euleriano possui cada aresta de G incluindo todas as arestas incidentes a v.
- Vamos imaginar um caminho que começa no meio de uma das arestas adjacentes ao início do circuito Euleriano e continua ao longo deste circuito e termina no mesmo ponto.

Circuito Euleriano (3)



Prova (continuação):

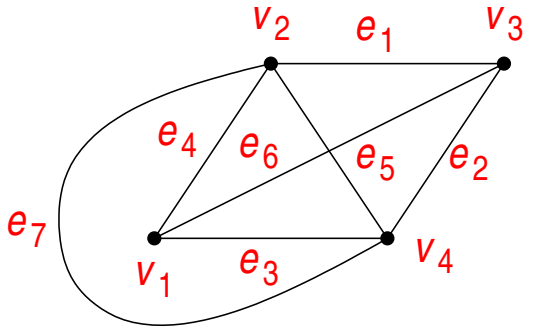
- Cada vez que o vértice v é visitado através de uma aresta de entrada, este vértice é "deixado" já que o caminho termina no meio de uma aresta.
- Já que cada circuito Euleriano passa em cada aresta de G exatamente uma única vez, cada aresta incidente a v é visitada uma única vez neste processo.
- Como o caminho que passa por v é feito através de arestas incidentes a v na forma de pares entrada/saída, o grau de v deve ser múltiplo de 2.
- Isto significa que o grau de v é par. [O que devia ser mostrado.]

Circuito Euleriano (4)

O contrapositivo deste teorema (que é logicamente equivalente ao teorema original) é:

Teorema: Se algum vértice de um grafo tem grau ímpar, então o grafo não tem um circuito Euleriano.

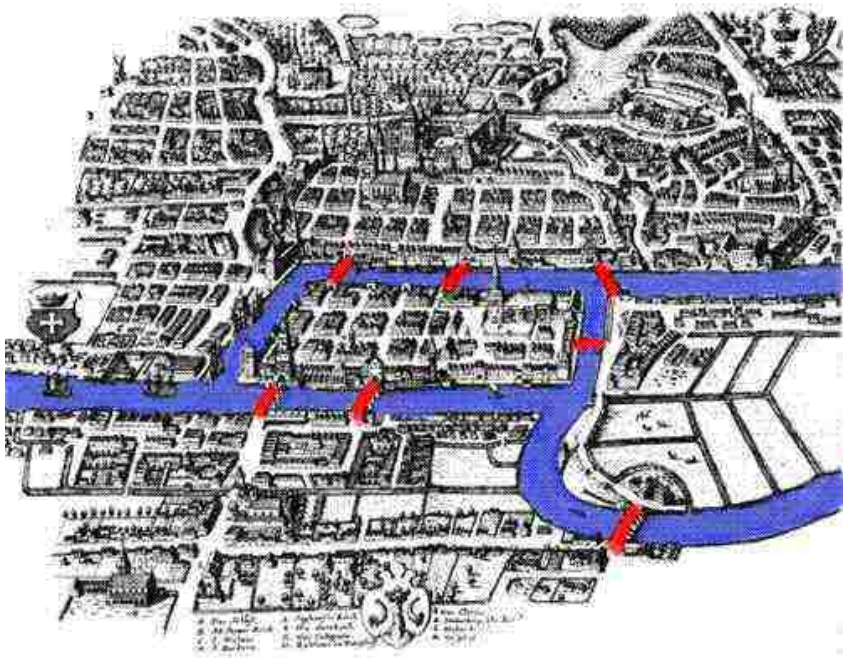
➜ Esta versão do teorema é útil para mostrar que um grafo não possui um circuito Euleriano.

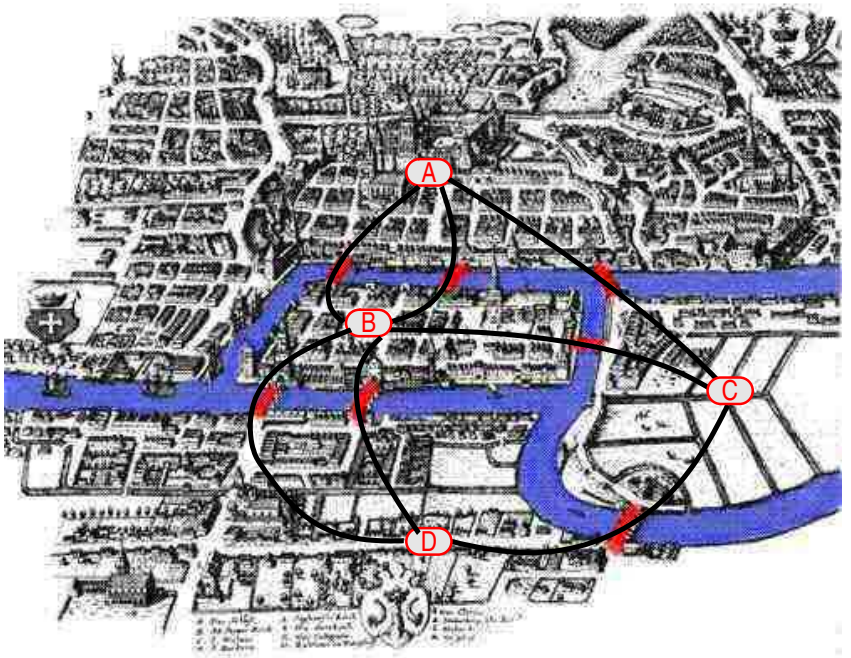


Vértices v_1 e v_3 possuem grau 3 e, assim, não possuem um circuito Euleriano.

Circuito Euleriano (5)

Revisitando o problema das sete pontes da cidade de Königsberg.





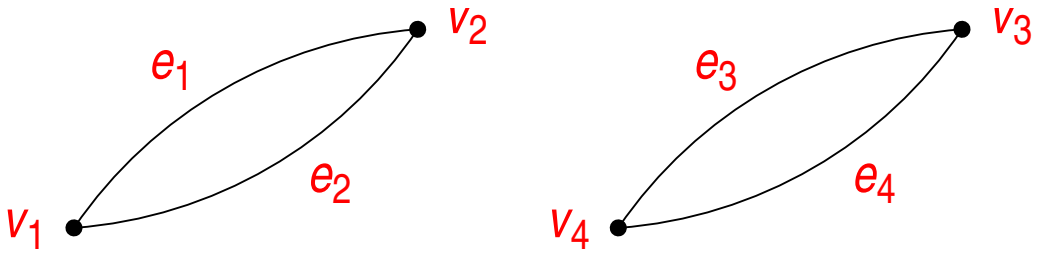
Problema: É possível que uma pessoa faça um percurso na cidade de tal forma que inicie e volte a mesma posição passando por todas as pontes somente uma única vez?

➜ Não. Todos os vértices têm grau ímpar.

Circuito Euleriano (6)

No entanto, se cada vértice de um grafo tem grau par, então o grafo tem um circuito Euleriano?

– Não. Por exemplo, no grafo abaixo todos os vértices têm grau par, mas como o grafo não é conexo, não possui um circuito Euleriano.



Circuito Euleriano (7)

Teorema: Se cada vértice de um grafo não vazio tem grau par e o grafo é conexo, então o grafo tem um circuito Euleriano.

Prova: [*Esta é uma prova construtivista, ou seja, apresenta um algoritmo para achar um circuito Euleriano para um grafo conexo no qual cada vértice tem grau par.*]

- Suponha que G é um grafo conexo não vazio e que cada vértice de G tem grau par. [Deve-se achar um circuito Euleriano para G.]
- Construa um circuito C usando o algoritmo descrito a seguir.

PASSO 1:

- Escolha qualquer vértice v de G. [Este passo pode ser executado já que pela suposição o conjunto de vértices de G é não vazio.]

Circuito Euleriano (8)

Prova (continuação):

PASSO 2:

– Escolha uma seqüência qualquer de vértices e arestas adjacentes, começando e terminando em v, sem repetir arestas. Chame o circuito resultante de C.

[Este passo pode ser executado pelas seguintes razões:

- Como o grau de cada vértice de G é par, é possível entrar num vértice qualquer que não seja o v por arestas de entrada e saída não visitadas ainda.
- Assim, uma seqüência de arestas adjacentes distintas pode ser obtida enquanto o vértice v não seja alcançado.
- Esta seqüência de arestas deve voltar em v já que existe um número finito de arestas.

]

Circuito Euleriano (9)

Prova (continuação):

PASSO 3: Verifique se C contém cada aresta e vértice de G. Se sim, C é um circuito Euleriano e o problema está terminado. Caso contrário, execute os passos abaixo.

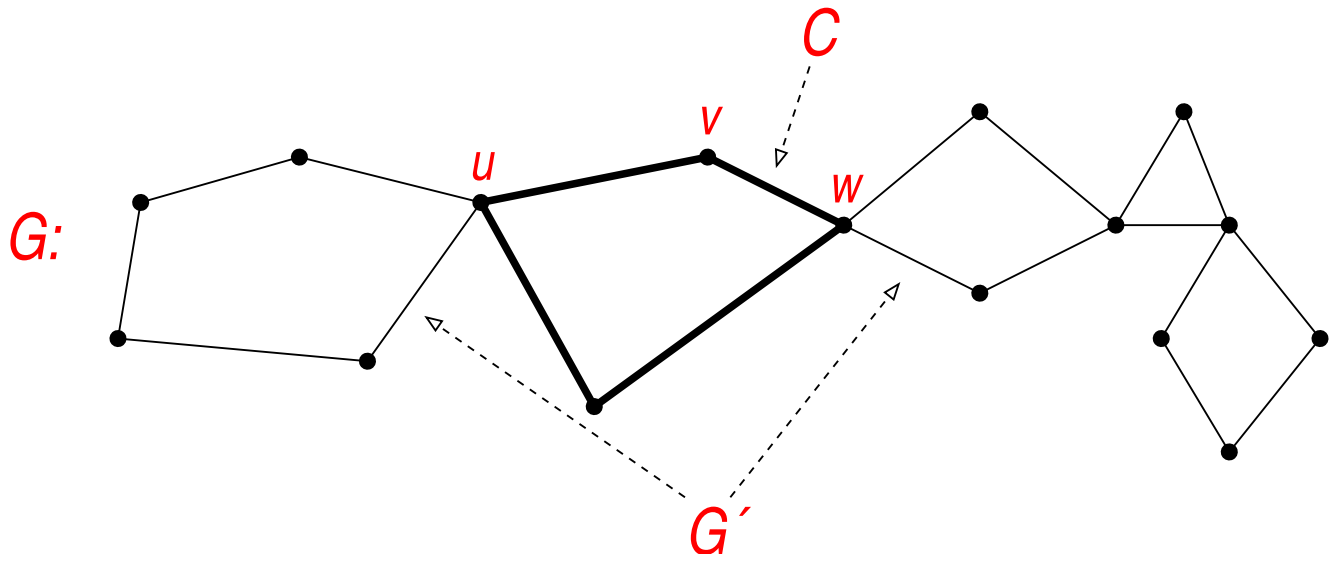
Circuito Euleriano (10)

Prova (continuação):

PASSO 3A:

- Remova todas as arestas do circuito C do grafo G e quaisquer vértices que se tornaram isolados quando as arestas de C são removidas.
- Chame o grafo resultante de G'.

[Note que G' pode não ser conexo, como ilustrado abaixo, mas cada vértice de G' tem grau par, já que removendo as arestas de C remove um número par de arestas de cada vértice e a diferença de dois números pares é par.]



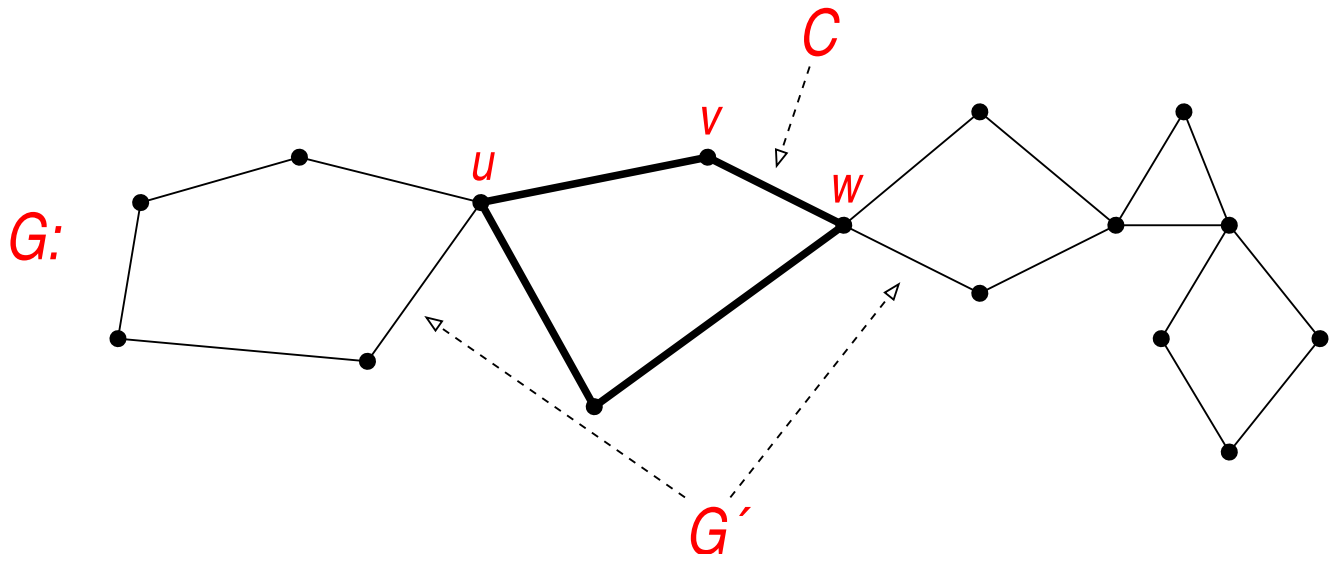
Circuito Euleriano (11)

Prova (continuação):

PASSO 3B:

– Escolha qualquer vértice w comum a ambos C e G'.

[Deve haver pelo menos um vértice deste tipo já que G é conexo. Na figura abaixo existem dois vértices deste tipo: u e w.]



Circuito Euleriano (12)

Prova (continuação):

PASSO 3C:

- Escolha uma seqüência qualquer de vértices e arestas adjacentes, começando e terminando em w, sem repetir arestas. Chame o circuito resultante de C'.

[Este passo pode ser executado já que o grau de cada vértice de G' é par e G' é finito. Veja a justificativa para o passo 2.]

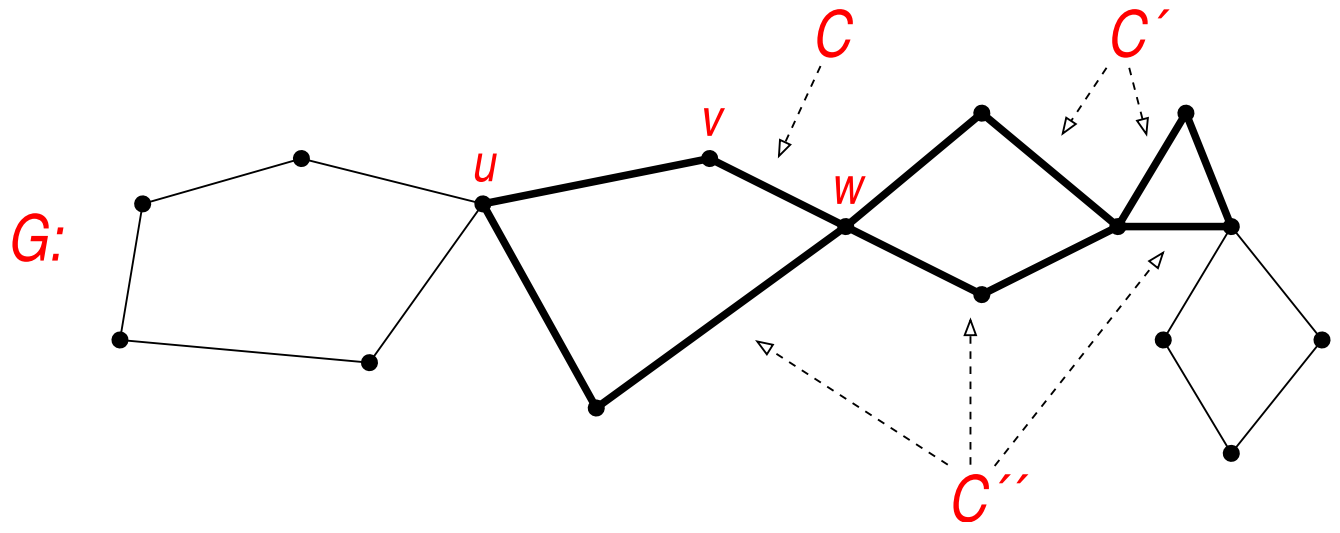
Circuito Euleriano (13)

Prova (continuação):

PASSO 3D:

- Agrupe C e C' para criar um novo circuito C'' como segue:
  - Comece em v e siga em direção a w.
  - Percorra todo o circuito C' e volte a w.
  - Caminhe pela parte de C não percorrida ainda até o vértice v.

[O efeito de executar os passos 3C e 3D para o grafo anterior é mostrado abaixo.]



Circuito Euleriano (14)

Prova (continuação):

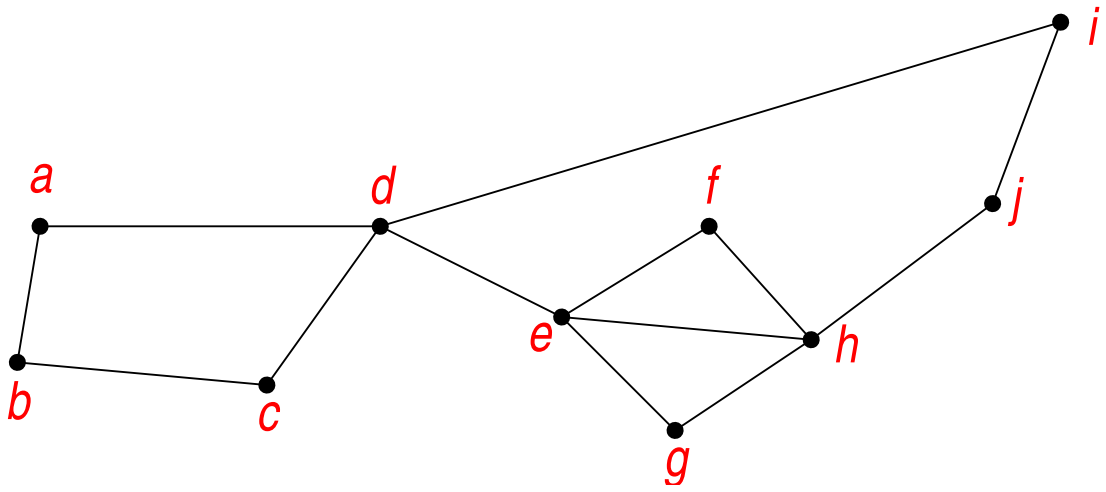
PASSO 3E:

- Seja $C \leftarrow C''$ e retorne ao passo 3.

[Como o grafo G é finito, a execução dos passos deste algoritmo termina, com a construção de um circuito Euleriano para G. Como diferentes escolhas podem ser feitas, diferentes circuitos podem ser gerados.]

Circuito Euleriano (15)

Determine se o grafo abaixo tem um circuito Euleriano. Em caso positivo ache um circuito Euleriano para o grafo.



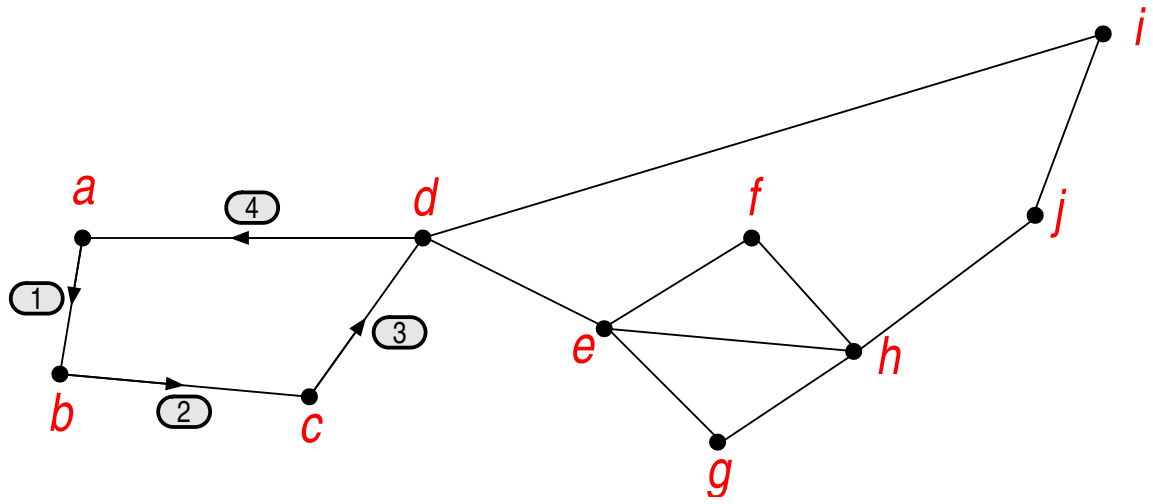
- Os vértices a, b, c, f, g, i, j têm grau 2.
- Os vértices d, e, h têm grau 4.

➜ Pelo teorema anterior, este grafo possui um circuito Euleriano.

Circuito Euleriano (16)

Seja $v = a$ e seja

$$C : abcda.$$



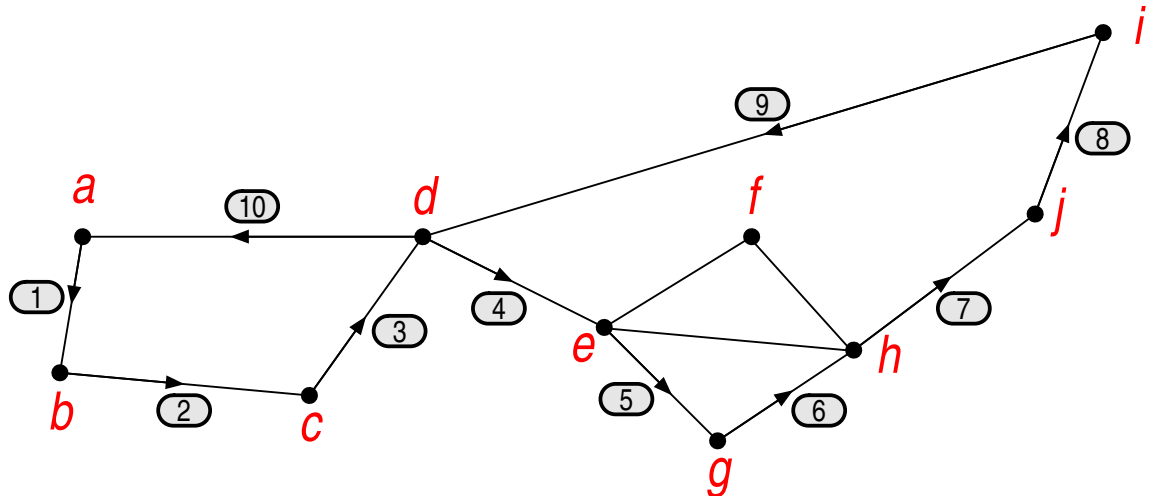
C não é um circuito Euleriano para este grafo, mas C possui uma intersecção com o restante do grafo no vértice d.

Circuito Euleriano (17)

Seja C' : $deghjid$. Agrupe C' a C para obter

$$C'' : abcdeghjida.$$

Seja $C \leftarrow C''$. Então C pode ser representado pelas arestas rotuladas no grafo abaixo:



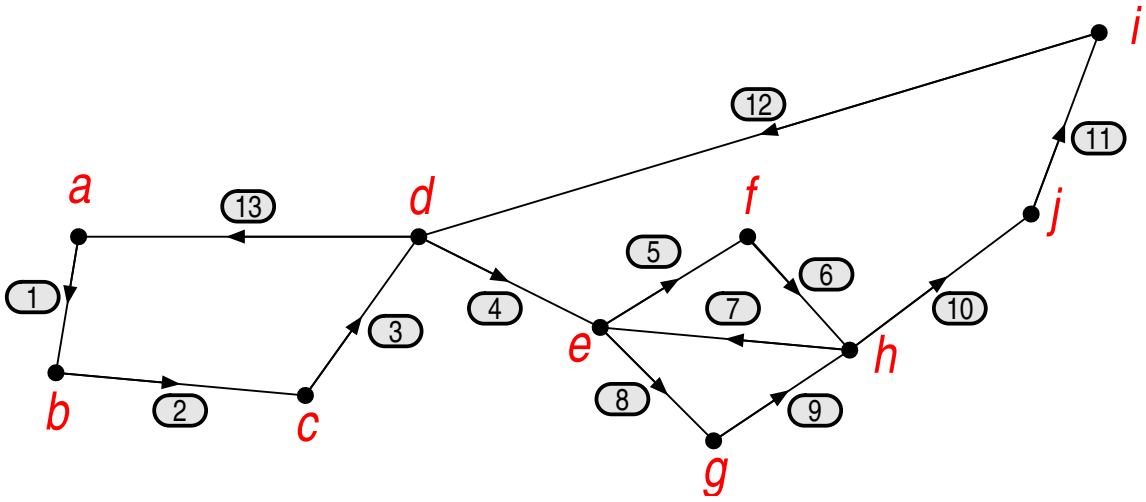
C não é um circuito Euleriano para este grafo, mas C possui uma intersecção com o restante do grafo no vértice e.

Circuito Euleriano (18)

Seja C' : $efhe$. Agrupe C' a C para obter

$$C'' : abcdefheghjida.$$

Seja $C \leftarrow C''$. Então C pode ser representado pelas arestas rotuladas no grafo abaixo:



C inclui cada aresta do grafo exatamente uma única vez e, assim, C é um circuito Euleriano para este grafo.

Circuito Euleriano (19)

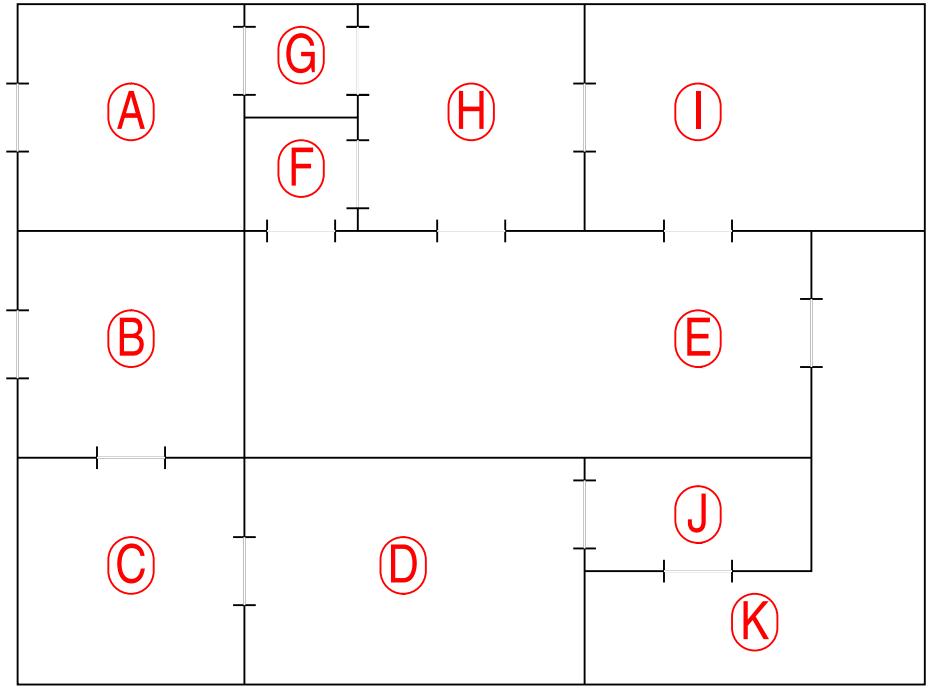
Teorema: Um grafo G tem um circuito Euleriano sse G é conexo e cada vértice de G tem grau par.

Definição: Seja G um grafo e seja v e w dois vértices de G. Um **Trajeto Euleriano** de v para w é uma seqüência de arestas e vértices adjacentes que começa em v, termina em w e passa por cada vértice de G pelo menos uma vez e passa por cada aresta de G exatamente uma única vez.

Corolário: Seja G um grafo e dois vértices v e w de G. Existe um trajeto Euleriano de v para w sse G é conexo e v e w têm grau ímpar e todos os outros vértices de G têm grau par.

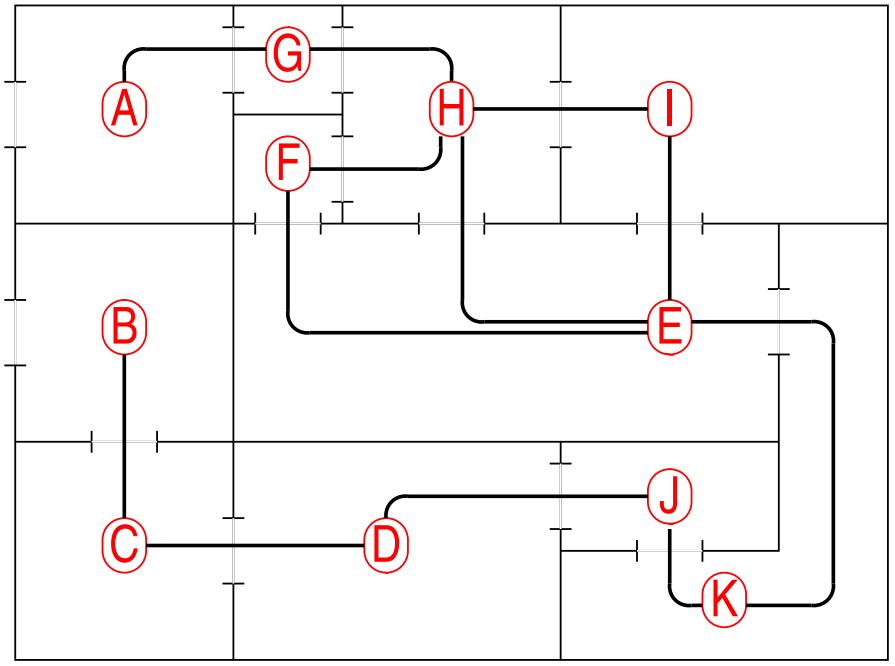
Trajeto Euleriano (1)

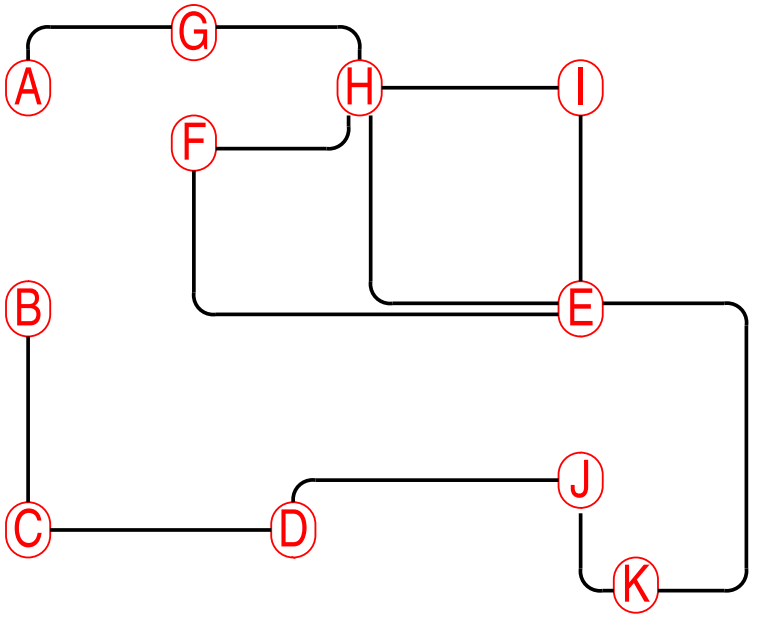
Uma casa possui uma divisão representada pela planta abaixo. É possível uma pessoa sair do cômodo A, terminar no cômodo B e passar por todas as portas da casa exatamente uma única vez? Se sim, apresente um possível trajeto.



Trajeto Euleriano (2)

A planta da casa pode ser representada pelo grafo abaixo:

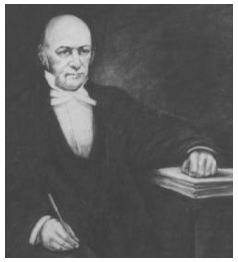




Cada vértice deste grafo tem um grau par, exceto os vértices A e B que têm grau 1. Assim, pelo corolário anterior, existe um trajeto Euleriano de A para B.

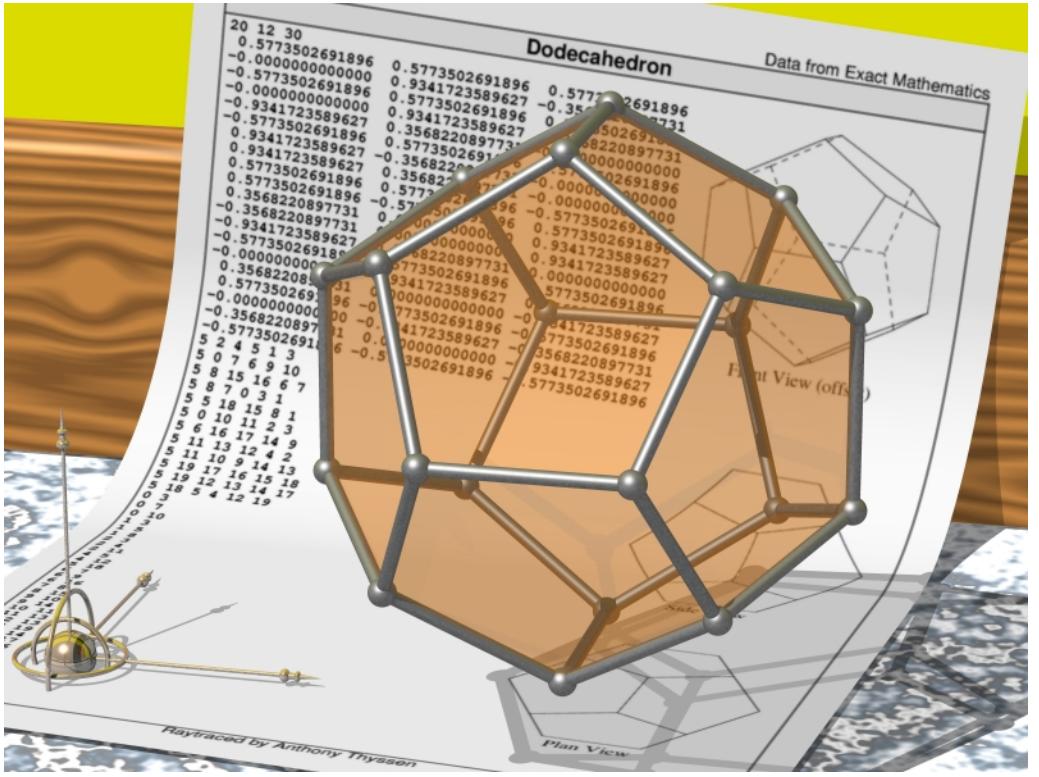
➜ AGHFEIHEKJDCB

Circuito Hamiltoniano (1)



William Hamilton (1805–1865), matemático irlandês. Contribuiu para o desenvolvimento da óptica, dinâmica e álgebra. Em particular, descobriu a álgebra dos *quaternions*. Seu trabalho provou ser significante para o desenvolvimento da mecânica quântica.

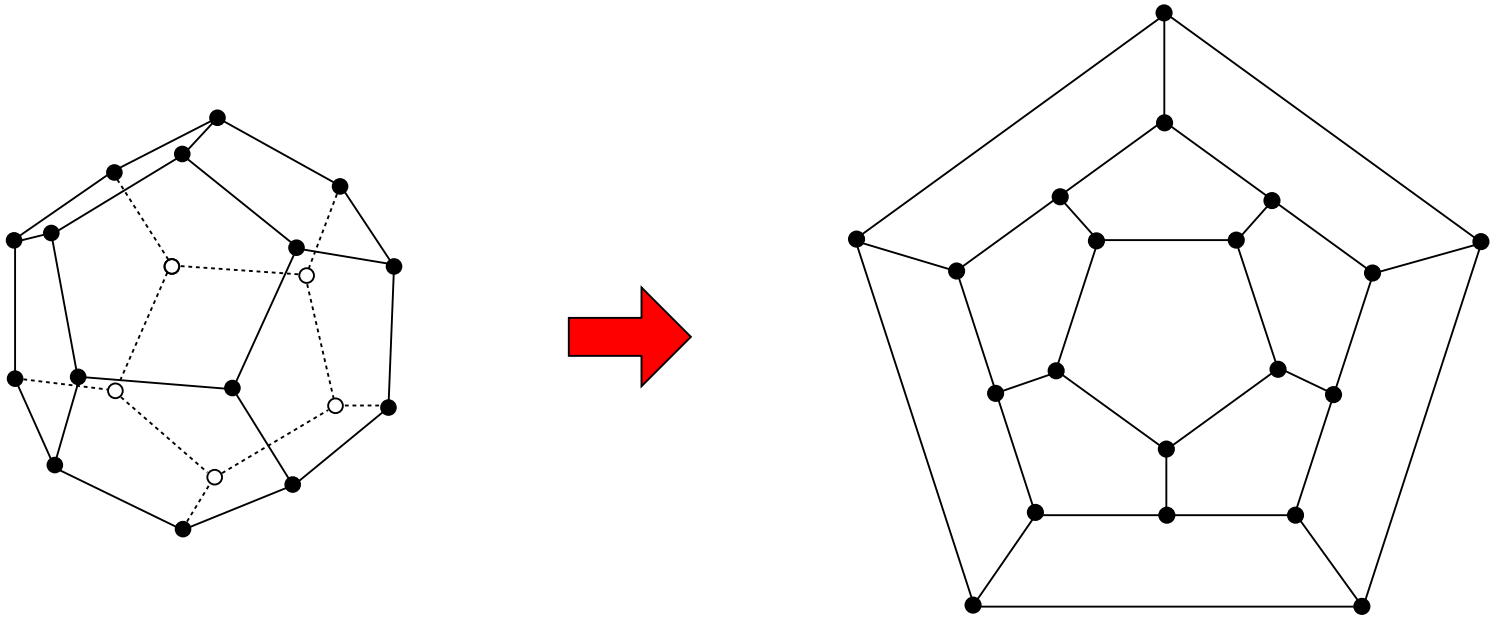
Em 1859, propôs um jogo na forma de um dodecaedro (sólido de 12 faces).



Circuito Hamiltoniano (2)
Jogo proposto por Hamilton

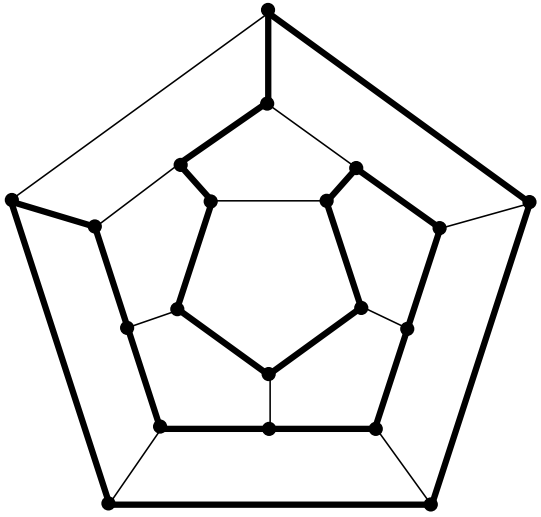
Cada vértice recebeu o nome de uma cidade: Londres, Paris, Hong Kong, New York, etc. O problema era: É possível começar em uma cidade e visitar todas as outras cidades exatamente uma única vez e retornar à cidade de partida?

O jogo é mais fácil de ser imaginado projetando o dodecaedro no plano:



Circuito Hamiltoniano (3)
Jogo proposto por Hamilton

Uma possível solução para este grafo é:



Definição: Dado um grafo G, um **Circuito Hamiltoniano** para G é um circuito simples que inclui cada vértice de G, ou seja, uma seqüência de vértices adjacentes e arestas distintas tal que cada vértice de G aparece exatamente uma única vez.

Comentários sobre circuitos Euleriano e Hamiltoniano (1)

- Circuito Euleriano:
  - Inclui todas as arestas uma única vez.
  - ➜ Inclui todos os vértices, mas que podem ser repetidos, ou seja, pode não gerar um circuito Hamiltoniano.

- Circuito Hamiltoniano:
  - Inclui todas os vértices uma única vez (exceto o inicial = final).
  - ➜ Pode não incluir todas as arestas, ou seja, pode não gerar um circuito Euleriano.

Comentários sobre circuitos Euleriano e Hamiltoniano (2)

- É possível determinar a priori se um grafo G possui um circuito Euleriano.
- Não existe um teorema que indique se um grafo possui um circuito Hamiltoniano nem se conhece um algoritmo eficiente (polinomial) para achar um circuito Hamiltoniano.
- No entanto, existe uma técnica simples que pode ser usada em muitos casos para mostrar que um grafo não possui um circuito Hamiltoniano.

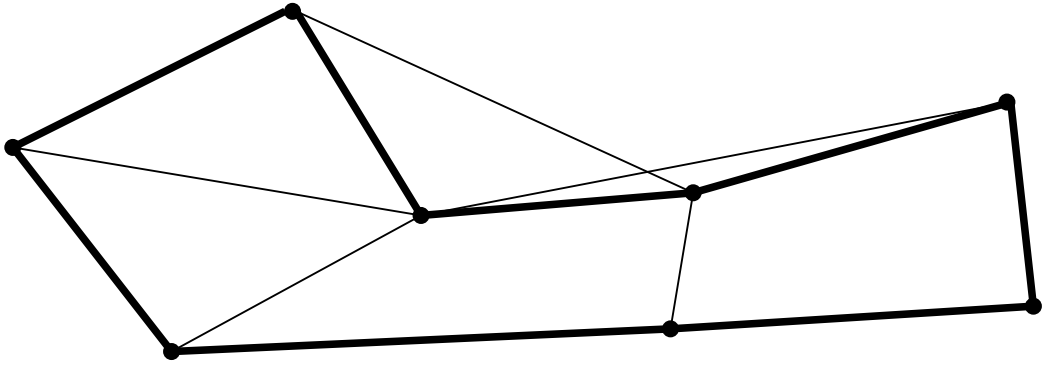
Determinando se um grafo não possui um circuito Hamiltoniano (1)

Suponha que um grafo G tenha um circuito Hamiltoniano C dado por:

$$C : v_0 e_1 v_1 e_2 \ldots v_{n-1} e_n v_n$$

Como C é um circuito simples, todas as arestas e_i são distintas e todos os vértices são distintos, exceto $v_0 = v_n$.

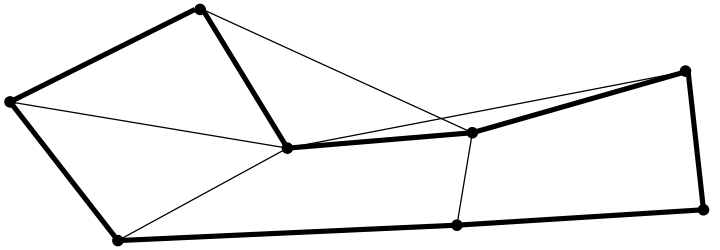
Seja H um subgrafo de G que é formado pelos vértices e arestas de C, como mostrado na figura abaixo (H é o subgrafo com as linhas grossas).



Determinando se um grafo não possui um circuito Hamiltoniano (2)

Se um grafo G tem um circuito Hamiltoniano então G tem um subgrafo H com as seguintes propriedades:

1. H contém cada vértice de G;
2. H é conexo;
3. H tem o mesmo número de arestas e de vértices;
4. Cada vértice de H tem grau 2.

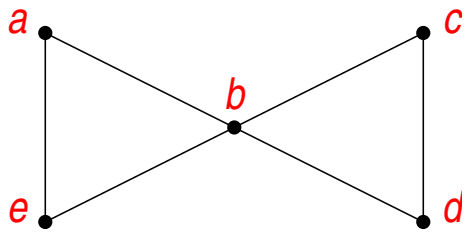


Contrapositivo desta afirmação:

➜ Se um grafo G não tem um subgrafo H com propriedades (1)–(4) então G não possui um circuito Hamiltoniano.

Determinando se um grafo não possui um circuito Hamiltoniano (3)

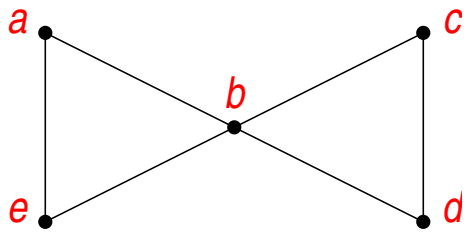
Prove que o grafo G abaixo não tem um circuito Hamiltoniano.



Se G tem um circuito Hamiltoniano, então G tem um subgrafo H que:

1. H contém cada vértice de G;
2. H é conexo;
3. H tem o mesmo número de arestas e de vértices;
4. Cada vértice de H tem grau 2.

Determinando se um grafo não possui um circuito Hamiltoniano (4)

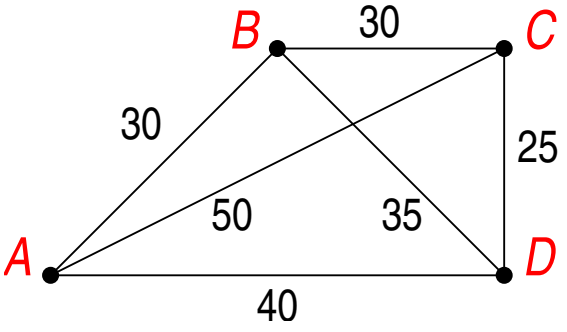


- Em G, *grau*$(b) = 4$ e cada vértice de H tem grau 2;
- Duas arestas incidentes a b devem ser removidas de G para criar H;
- Qualquer aresta incidente a b que seja removida fará com que os outros vértices restantes tenham grau menor que 2;

➜ Conseqüentemente, não existe um subgrafo H com as quatro propriedades acima e, assim, G não possui um circuito Hamiltoniano.

O Problema do Caixeiro Viajante (1)

- Em inglês, *Traveling Salesman Problem*, ou TSP.
- Suponha o mapa abaixo mostrando quatro cidades (A, B, C, D) e as distâncias em km entre elas.



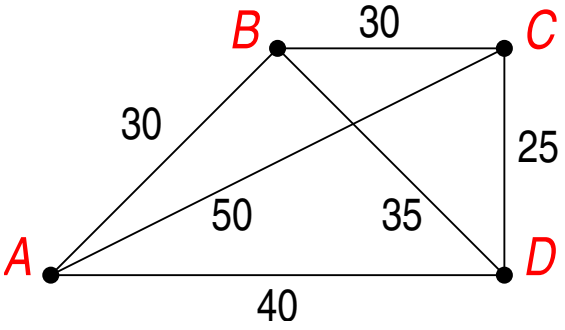
- Um caixeiro viajante deve percorrer um circuito Hamiltoniano, ou seja, visitar cada cidade exatamente uma única vez e voltar a cidade inicial.

➜ Que rota deve ser escolhida para minimizar o total da distância percorrida?

O Problema do Caixeiro Viajante (2)

- Possível solução:
  - Enumere todos os possíveis circuitos Hamiltonianos começando e terminando em A;
  - Calcule a distância de cada um deles;
  - Determine o menor deles.

Rota	Distância (km)
$ABCDA$	$30 + 30 + 25 + 40 = 125$
$ABDCA$	$30 + 35 + 25 + 50 = 140$
$ACBDA$	$50 + 30 + 35 + 40 = 155$
$ACDBA$	$50 + 25 + 35 + 30 = 140$
$ADBCA$	$40 + 35 + 30 + 50 = 155$
$ADCBA$	$40 + 25 + 30 + 30 = 125$



➜ Assim, tanto a rota $ABCDA$ ou $ADCBA$ tem uma distância total de 125 km.

O Problema do Caixeiro Viajante (3)

- A solução do TSP é um circuito Hamiltoniano que minimiza a distância total percorrida para um grafo valorado arbitrário G com n vértices, onde uma distância é atribuída a cada aresta.

- Algoritmo para resolver o TSP:
  - Atualmente, força bruta, como feito no exemplo anterior.
  - ➜ Problema da classe NP-Completo.

- Exemplo: para o grafo K_{30} existem

$$29! \approx 8,84 \times 10^{30}$$

  circuitos Hamiltonianos diferentes começando e terminando num determinado vértice.

- Mesmo se cada circuito puder ser achado e calculado em apenas 1μs, seria necessário aproximadamente $2,8 \times 10^{17}$ anos para terminar a computação nesse computador.

Representação de um grafo

- Dado um grafo $G = (V, E)$:
  - V = conjunto de vértices.
  - E = conjunto de arestas, que pode ser representado pelo subconjunto de $V \times V$.

- O tamanho da entrada de dados é medido em termos do:
  - Número de vértices $|V|$.
  - Número de arestas $|E|$.

- Se G é conexo então $|E| \geq |V| - 1$.

Representação de um grafo
Convenções

- Convenção I (Notação):
  - Dentro e somente dentro da notação assintótica os símbolos V e E significam respectivamente $|V|$ e $|E|$.
  - ➜ Se um algoritmo "executa em tempo $O(V + E)$" é equivalente a dizer que "executa em tempo $O(|V| + |E|)$".

- Convenção II (Em pseudo-código):
  - O conjunto V de vértices de G é representado por $V[G]$.
  - O conjunto E de arestas de G é representado por $E[G]$.
  - ➜ Os conjuntos V e E são vistos como atributos de G.

Representação de um grafo
Estruturas de dados

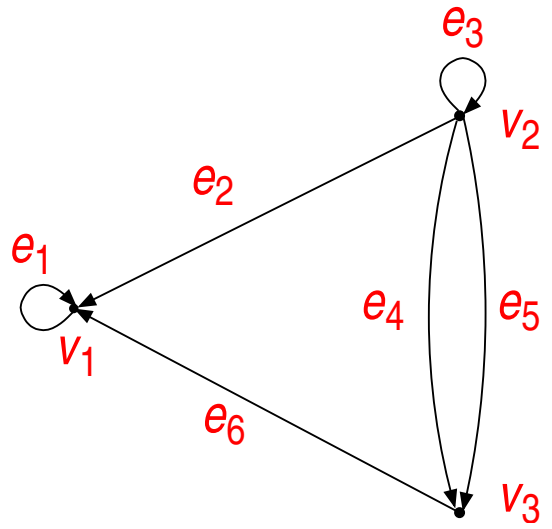
- Matriz de adjacência:
  - Forma preferida de representar grafos densos ($|E| \approx |V|^2$).
  - Indica rapidamente ($O(1)$) se existe uma aresta conectando dois vértices.

- Lista de adjacência:
  - Representação normalmente preferida.
  - Provê uma forma compacta de representar grafos esparsos ($|E| \ll |V|^2$).

➜ As duas formas principais de representar um grafo.

Representação de um grafo
Matriz de adjacência e grafo dirigido (1)

Seja o grafo dirigido abaixo:



Este grafo pode ser representado por uma matriz $A = (a_{ij})$, onde (a_{ij}) representa o número de arestas de v_i para v_j.

➜ Matriz de Adjacência

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 0 & 0 \end{bmatrix} \end{array}$$

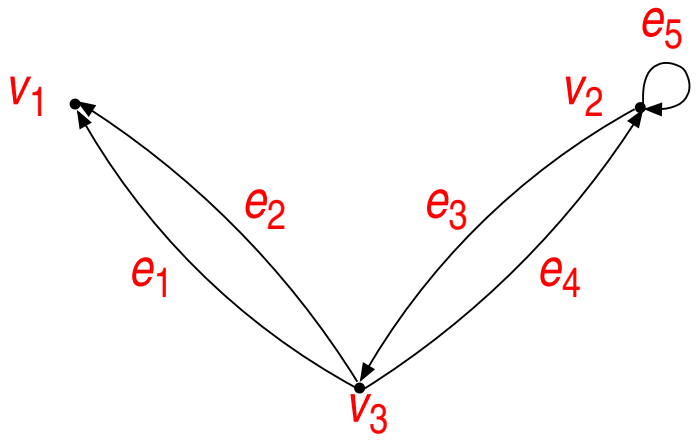
Representação de um grafo
Matriz de adjacência e grafo dirigido (2)

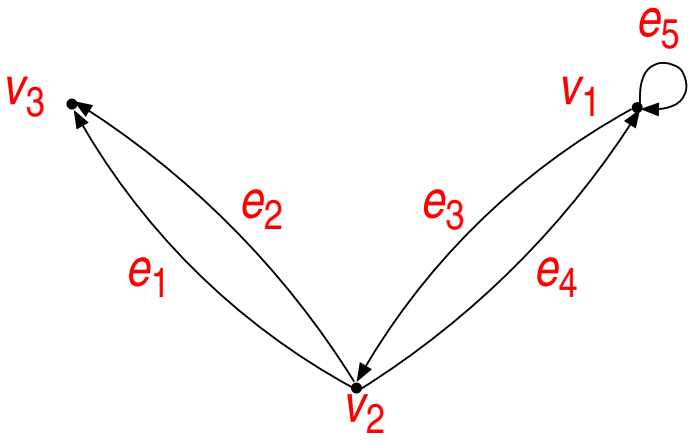
Definição: Seja G um grafo dirigido com vértices $v_1, v_2, \ldots, v_n$. A matriz de adjacência de G é a matriz $A = (a_{ij})$ ($A[1 \ldots n, 1 \ldots n]$) é definida como:

$$a_{ij} = \text{ \# de arestas de } v_i \text{ para } v_j, \forall i, j = 1, 2, \ldots, n.$$

- Valor diferente de zero na diagonal principal: laço.
- Valor igual a 1 na entrada (a_{ij}): uma única aresta de v_i a v_j.
- Valores maiores que 1 na entrada (a_{ij}): arestas paralelas de v_i a v_j.
- Espaço: $O(V^2)$.

Representação de um grafo
Matriz de adjacência e grafo dirigido (3)





$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 2 & 1 & 0 \end{bmatrix} \end{array}$$

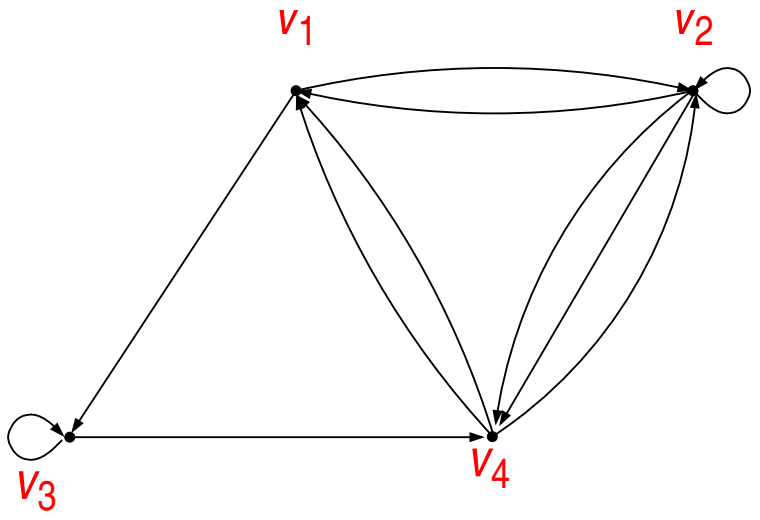
$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

Representação de um grafo
Matriz de adjacência e grafo dirigido (4)

Dada a matriz de adjacência de um grafo:

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 2 \\ 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Um possível desenho deste grafo é:

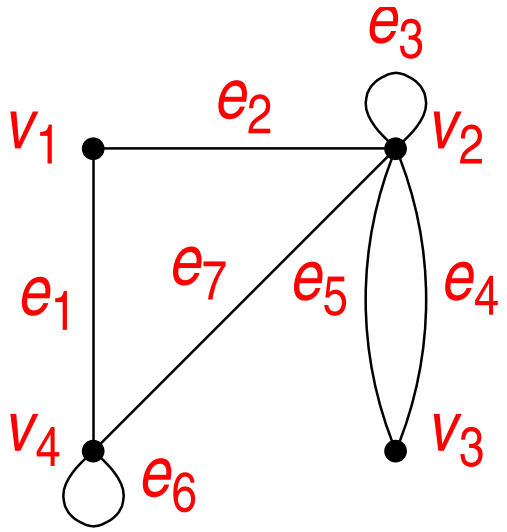


Representação de um grafo
Matriz de adjacência e grafo não dirigido

Definição: Seja G um grafo não dirigido com vértices $v_1, v_2, \ldots, v_n$. A matriz de adjacência de G é a matriz $A = (a_{ij})$ sobre o conjunto dos inteiros não negativos tal que

$$a_{ij} = \text{ \# de arestas conectando } v_i \text{ a } v_j, \forall i, j = 1, 2, \ldots, n.$$

Dado o grafo:

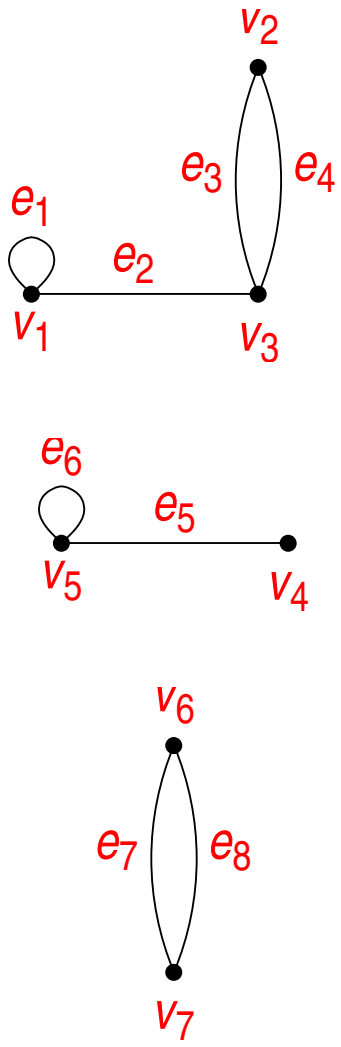


A matriz de adjacência correspondente é:

$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 2 & 1 \\ 0 & 2 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \end{array}$$

Representação de um grafo
Matriz de adjacência e componentes conexos

Dado o grafo:



A matriz de adjacência correspondente é:

$$A = \left[\begin{array}{ccc|cc|cc} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \end{array}\right]$$

A matriz A consiste de "blocos" de diferentes tamanhos ao longo da diagonal principal, já que o conjunto de vértices é disjunto.

Representação de um grafo
Matriz de adjacência: Análise

- Deve ser utilizada para grafos densos, onde $|E|$ é próximo de $|V|^2$ ($|E| \approx |V|^2$).
- O tempo necessário para acessar um elemento é independente de $|V|$ ou $|E|$.
- É muito útil para algoritmos em que necessitamos saber com rapidez se existe uma aresta ligando dois vértices.
- A maior desvantagem é que a matriz necessita $O(V^2)$ de espaço.
- Ler ou examinar a matriz tem complexidade de tempo $O(V^2)$.

Representação de um grafo
Uso de matriz de adjacência

- Quando usada, a maior parte dos algoritmos requer tempo $O(V^2)$, mas existem exceções.

- Seja um grafo dirigido que contém um vértice *sink*, ou seja, um vértice com:
  - Grau de entrada (*in-degree*) = $|V| - 1$
  - Grau de saída (*out-degree*) = 0
  - Não existe uma aresta *loop*

- Apresente um algoritmo para determinar se um grafo dirigido possui um vértice *sink* em tempo $O(V)$ usando uma matriz de adjacência.

Representação de um grafo
Número de vértices *sink* num grafo dirigido

Quantos vértices *sink* um grafo dirigido $G = (V, E)$ possui no máximo?

➜ No máximo 1.

Prova por contradição:

- Suponha que s_i e s_j sejam vértices *sink*.
- Deve existir uma aresta de todos os nós do grafo G para s_i e s_j, exceto *loops*.
- Em particular deve existir uma aresta (s_i, s_j) e uma aresta (s_j, s_i) já que s_i e s_j são vértices *sink*.
- Isto não pode ocorrer já que o grau de saída de um vértice *sink* é 0.

Logo, se existir um vértice *sink* no grafo G é no máximo 1.

Representação de um grafo
Lista de adjacência

- Vetor *Adj* de $|V|$ listas, uma para cada vértice de V.

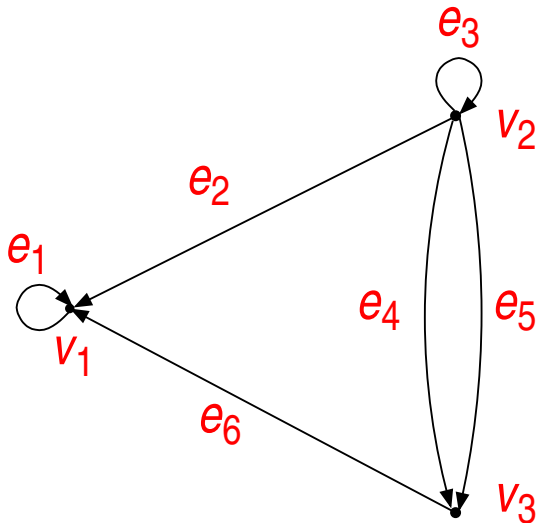
- Para cada vértice $u \in V$, a lista *Adj*[u] contém apontadores para todos os vértices v tal que a aresta $(u, v) \in E$ (todos os vértices adjacentes a u em G).
  ➜ Definição vale para grafos não dirigidos e dirigidos.

- $\sum_{i=1}^{|V|}$ "comprimento da lista de adjacência", vale:
  - Grafo dirigido = $|E|$, cada aresta aparece uma única vez na lista.
  - Grafo não dirigido = $2|E|$, cada aresta aparece duas vezes na lista (entrada de u e entrada de v).

- Espaço: $O(V + E)$.

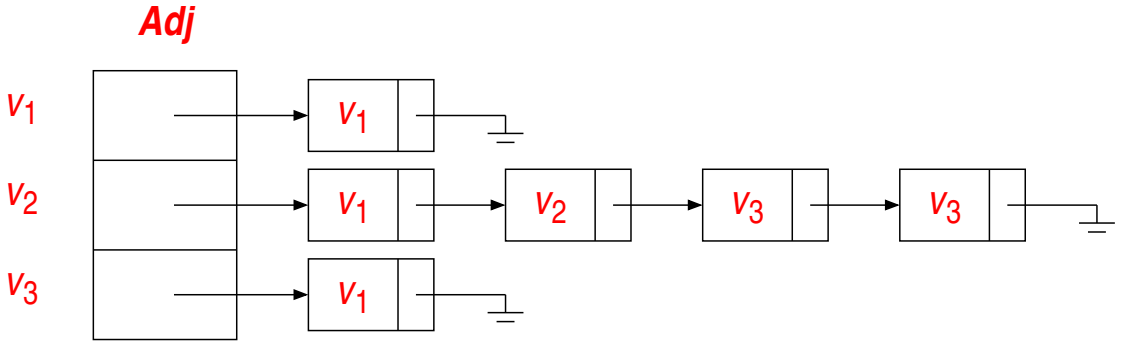
Representação de um grafo
Lista de adjacência e grafo dirigido (1)

Seja o grafo dirigido abaixo:

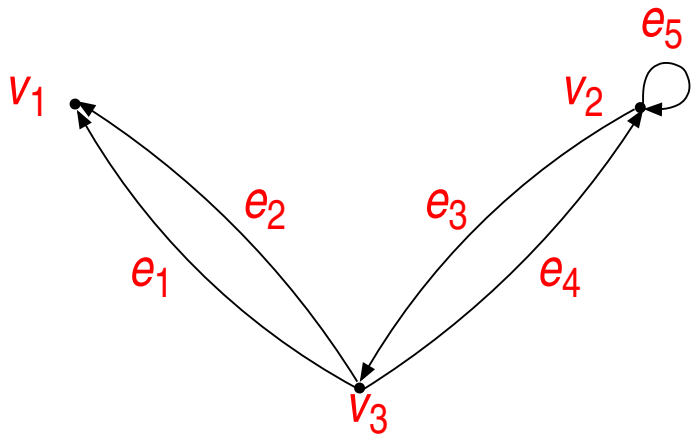


Este grafo pode ser representado por uma lista de adjacência *Adj*:

$$
\begin{aligned}
Adj[v_1] &= [v_1] \\
Adj[v_2] &= [v_1, v_2, v_3, v_3] \\
Adj[v_3] &= [v_1]
\end{aligned}
$$

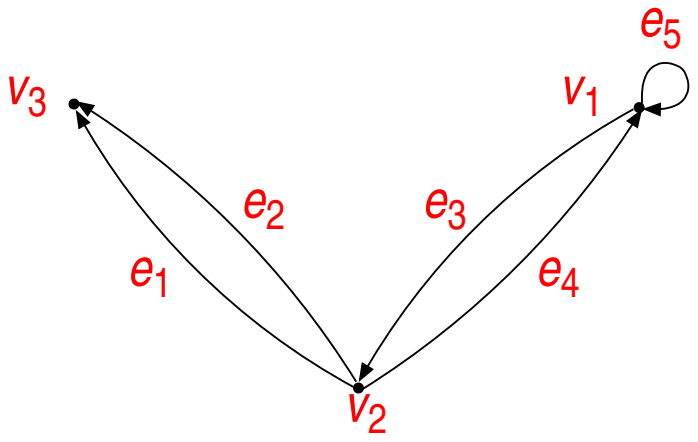


Representação de um grafo
Lista de adjacência e grafo dirigido (2)



Este grafo pode ser representado pela lista de adjacência:

$$
\begin{aligned}
Adj[v_1] &= [] \\
Adj[v_2] &= [v_2, v_3] \\
Adj[v_3] &= [v_1, v_1, v_2]
\end{aligned}
$$

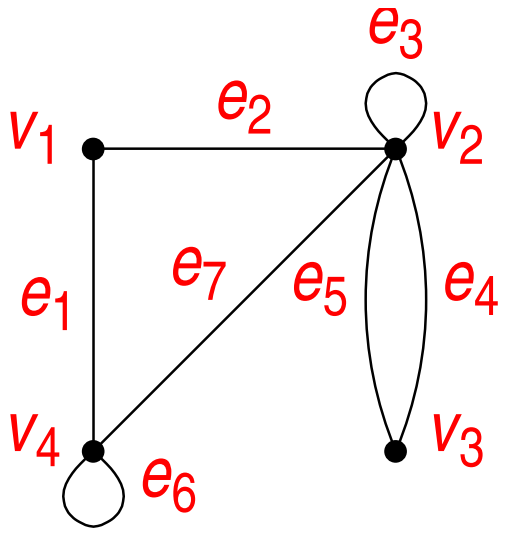


Este grafo pode ser representado pela lista de adjacência:

$$
\begin{aligned}
Adj[v_1] &= [v_1, v_2] \\
Adj[v_2] &= [v_1, v_3, v_3] \\
Adj[v_3] &= []
\end{aligned}
$$

Representação de um grafo
Lista de adjacência e grafo não dirigido

Dado o grafo:



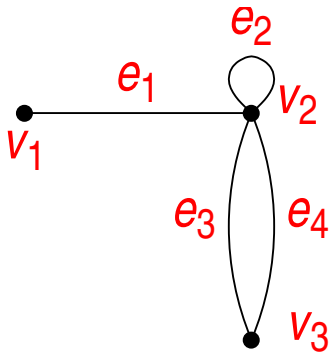
Uma lista de adjacência correspondente é:

$$
\begin{aligned}
Adj[v_1] &= [v_2, v_4] \\
Adj[v_2] &= [v_1, v_2, v_3, v_3, v_4] \\
Adj[v_3] &= [v_2, v_2] \\
Adj[v_4] &= [v_1, v_2, v_4]
\end{aligned}
$$

Contando caminhos de tamanho n (1)

O tamanho (comprimento) de um caminho é o número de arestas do mesmo, ou seja, número de vértices menos um.

Dado o grafo



O caminho

$$v_2 e_3 v_3 e_4 v_2 e_2 v_2 e_3 v_3$$

tem tamanho 4.

Quantos caminhos distintos de tamanho 2 existem conectando v_2 a v_2?

$$v_2 e_1 v_1 e_1 v_2$$
$$v_2 e_2 v_2 e_2 v_2$$
$$v_2 e_3 v_3 e_4 v_2$$
$$v_2 e_4 v_3 e_3 v_2$$
$$v_2 e_3 v_3 e_3 v_2$$
$$v_2 e_4 v_3 e_4 v_2$$

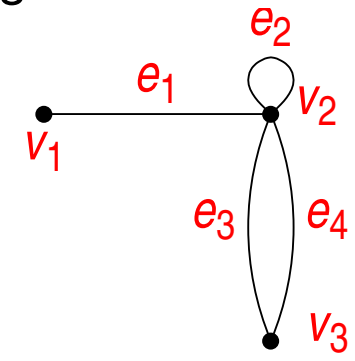
➜ Existem seis caminhos distintos.

Contando caminhos de tamanho n (2)

Quantos caminhos distintos de tamanho n existem conectando dois vértices de um dado grafo G?

➜ Este valor pode ser computado usando multiplicação de matrizes.

Seja o grafo:



A matriz de adjacência correspondente é:

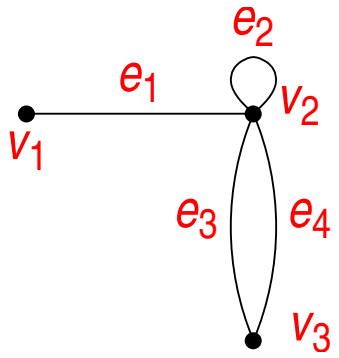
$$A = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} \end{array}$$

Contando caminhos de tamanho n (3)

O valor de A^2 é dado por:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 2 \\ 0 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 6 & 2 \\ 2 & 2 & 4 \end{bmatrix}$$

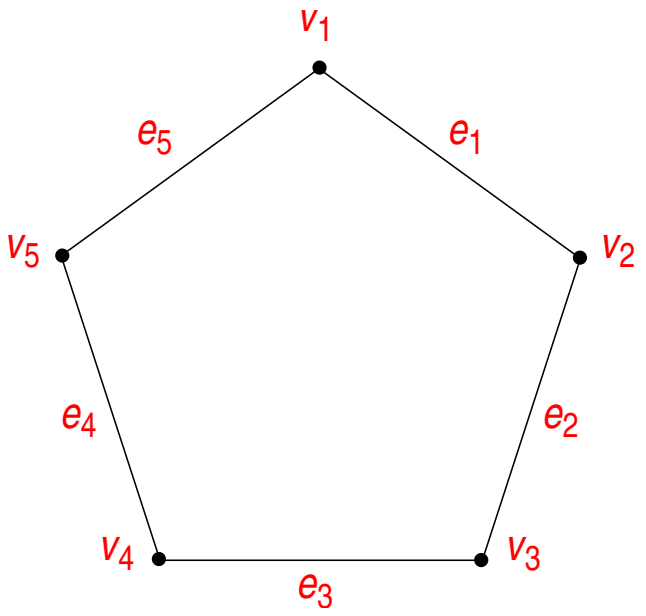
Observe que $a_{22} = 6$, que é o número de caminhos de tamanho 2 de v_2 para v_2.

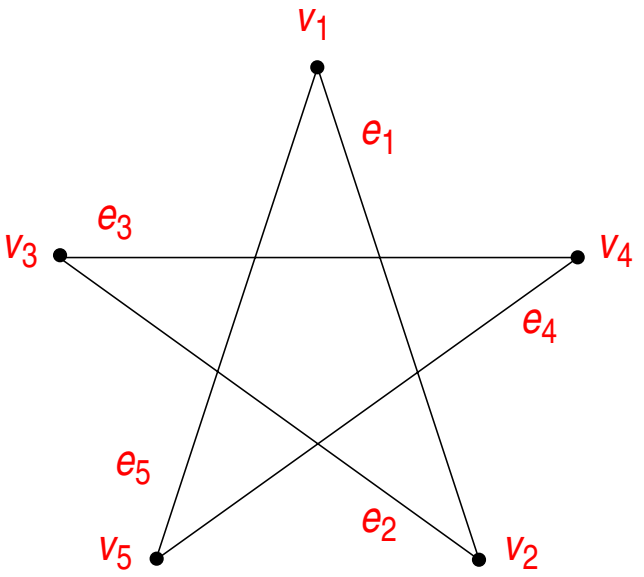


- Se A é a matriz de adjacência de um grafo G, a entrada a_{ij} da matriz A^2 indica a quantidade de caminhos de tamanho 2 conectando v_i a v_j no grafo G.
- Este resultado é válido para caminhos de tamanho n calculando A^n.

Isomorfismo de grafos (1)

Os desenhos abaixo



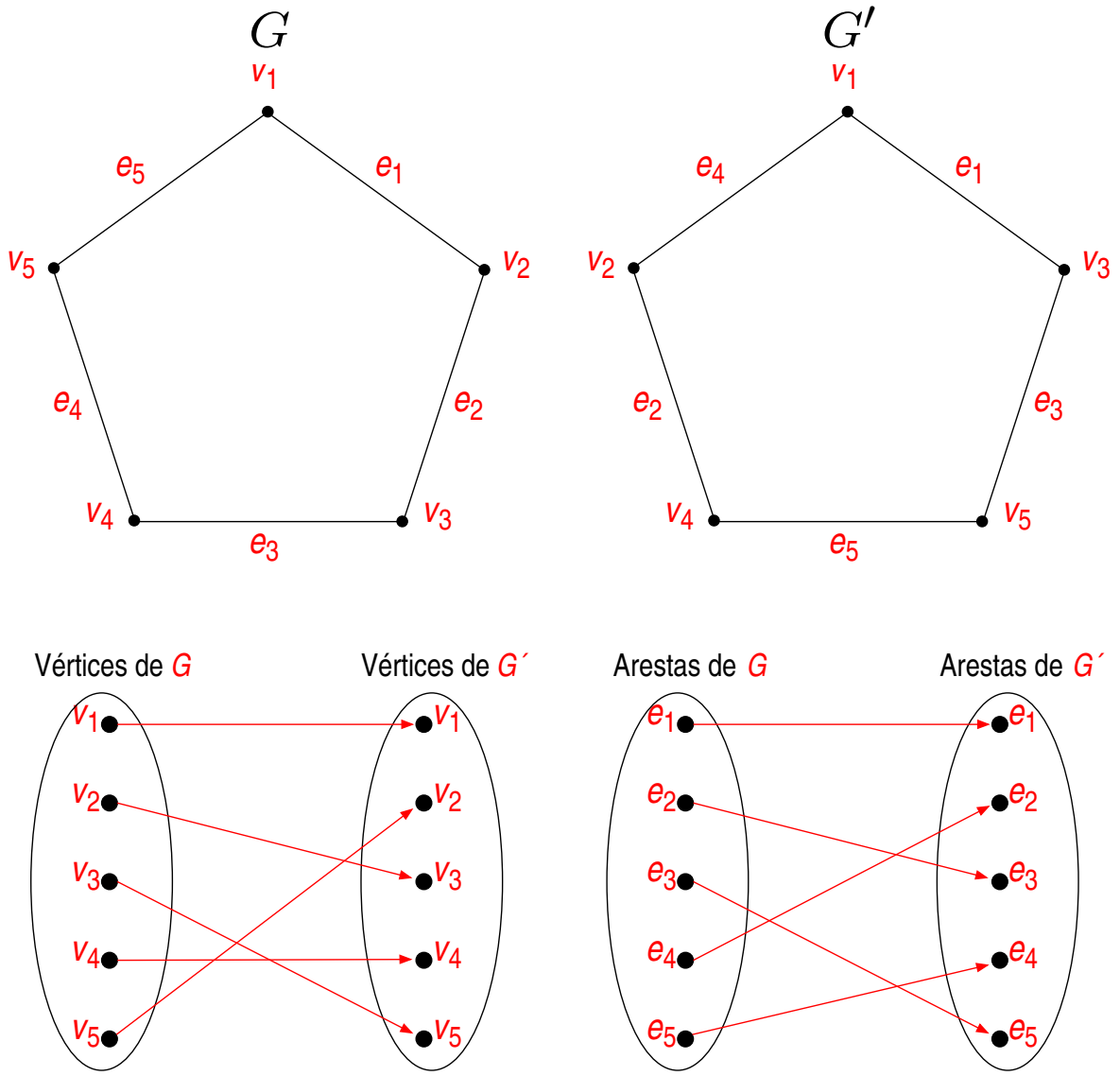


representam o mesmo grafo G:

- Conjuntos de vértices e arestas são idênticos;
- Funções aresta–vértice são as mesmas.

➜ Grafos isomorfos (do grego, o que significa a mesma forma).

Isomorfismo de grafos (2)



Estes grafos são diferentes apesar de possuírem os mesmos conjuntos de vértices e arestas.

As funções aresta–vértice não são as mesmas.

Isomorfismo de grafos (3)

Definição: Sejam os grafos G e G' com conjuntos de vértices $V(G)$ e $V(G')$ e com conjuntos de arestas $E(G)$ e $E(G')$, respectivamente. O grafo G é isomorfo ao grafo G' sse existem correspondências um-para-um

$$g : V(G) \rightarrow V(G')$$

$$h : E(G) \rightarrow E(G')$$

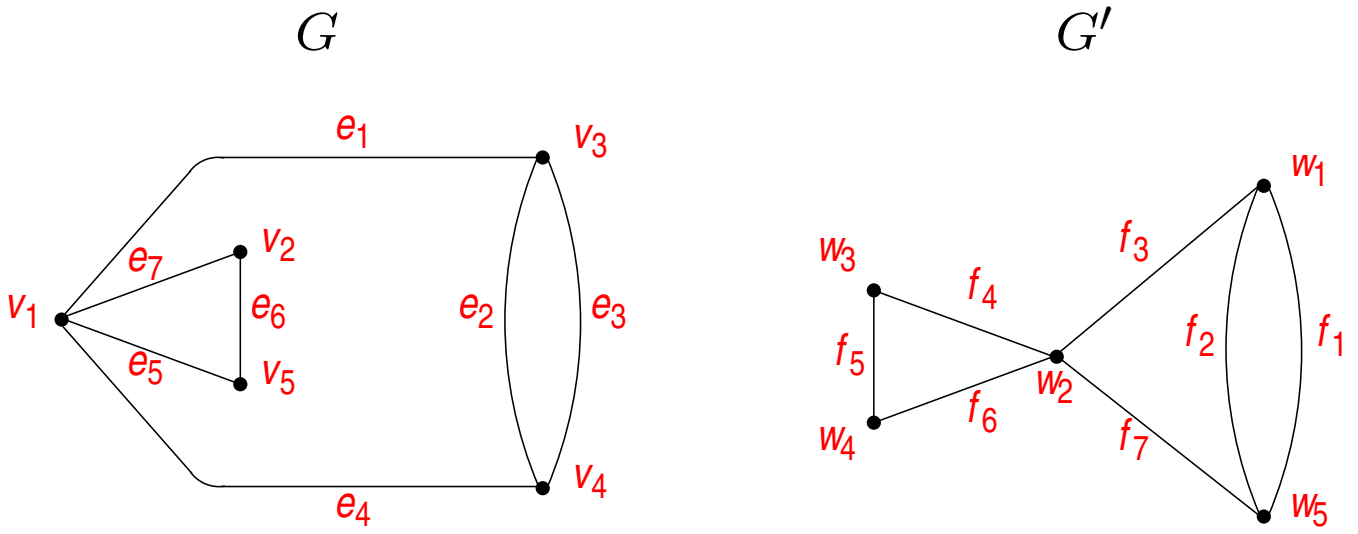
que preservam as funções aresta-vértice de G e G' no sentido que

$$\forall v \in V(G) \wedge e \in E(G)$$

v é um nó terminal de $e \Leftrightarrow g(v)$ é um nó terminal de $h(e)$.

Isomorfismo de grafos (4)

Os grafos



são isomorfos?

Isomorfismo de grafos (5)

Para resolver este problema, devemos achar funções

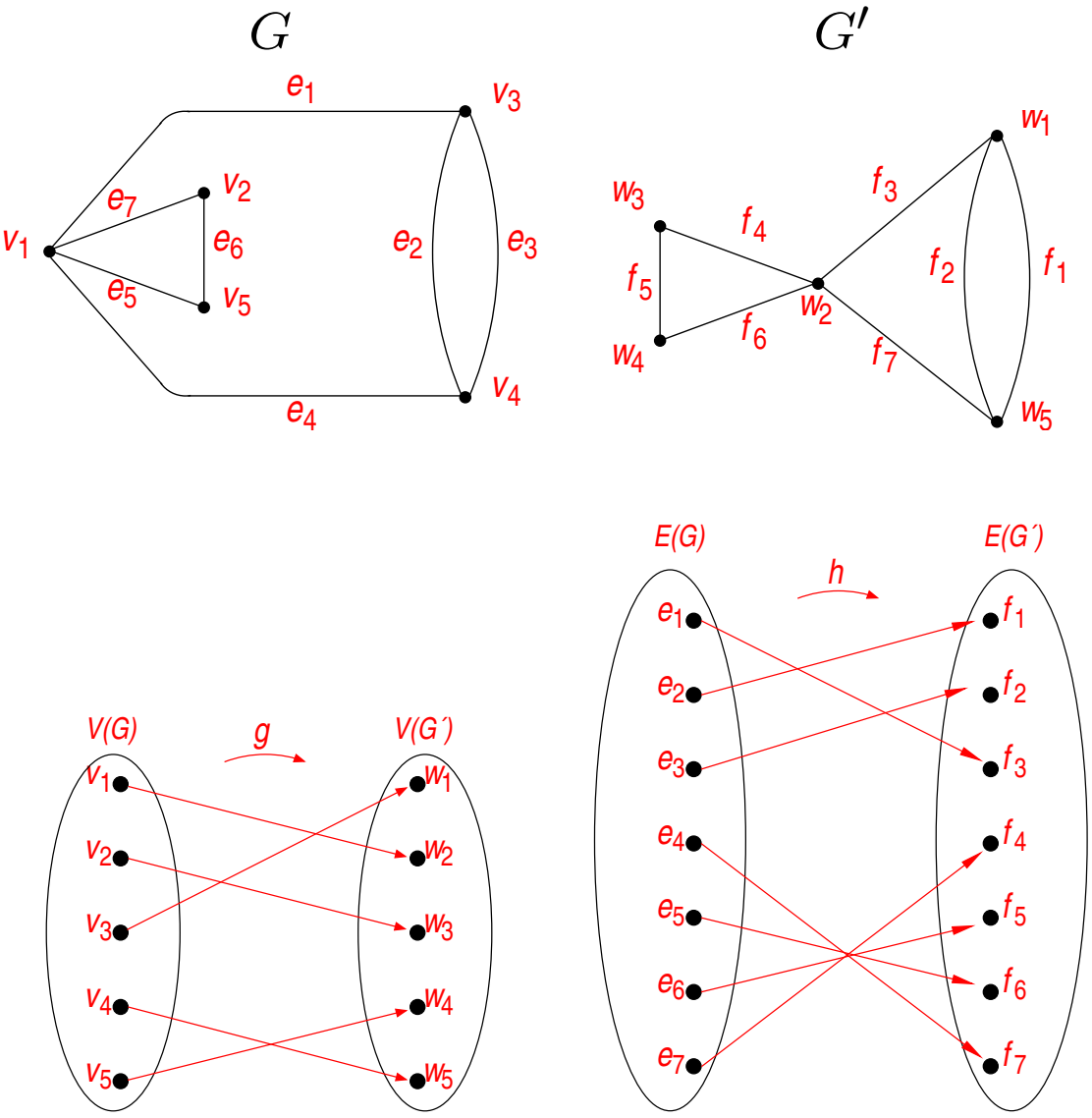
$$g : V(G) \rightarrow V(G')$$

e

$$h : E(G) \rightarrow E(G')$$

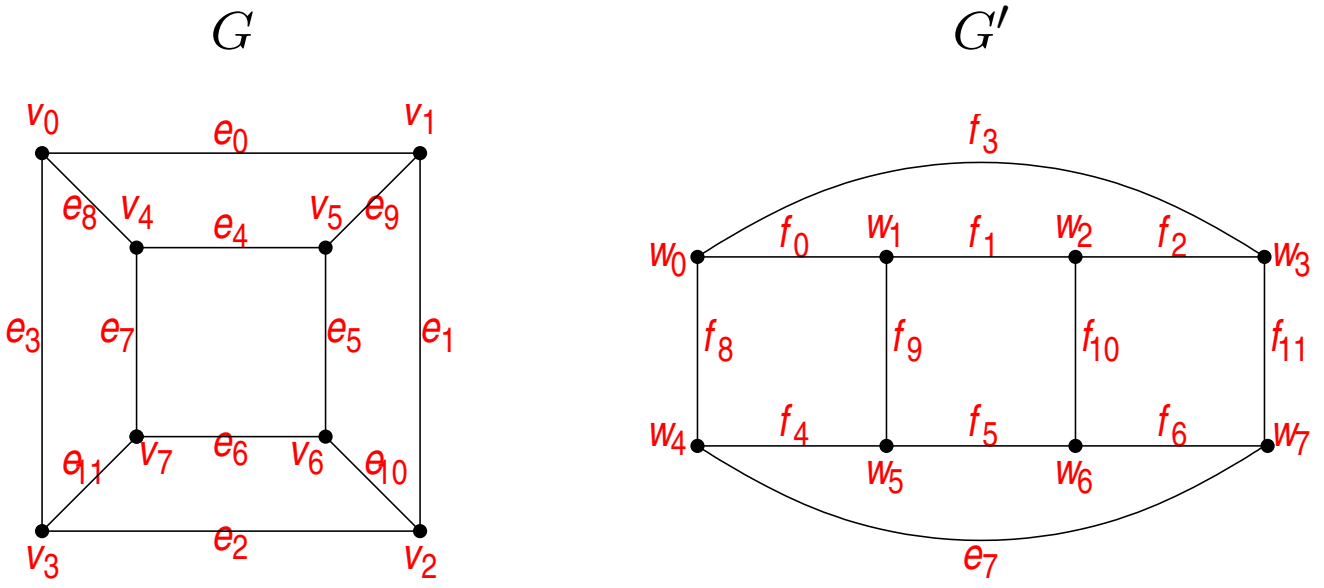
tal que exista a correspondência como mencionado anteriormente.

➜ Grafos G e G' são isomorfos.



Isomorfismo de grafos (6)

Os grafos



são isomorfos?

Isomorfismo de grafos (7)

Para resolver este problema, devemos achar funções

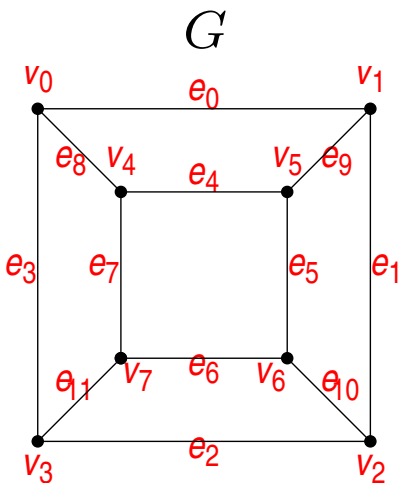
$$g : V(G) \rightarrow V(G')$$

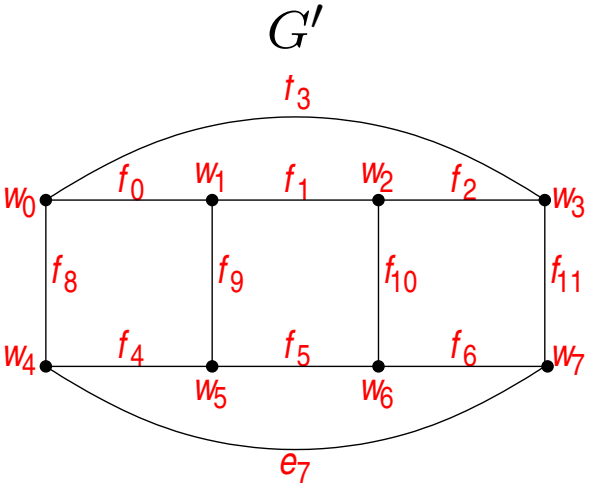
e

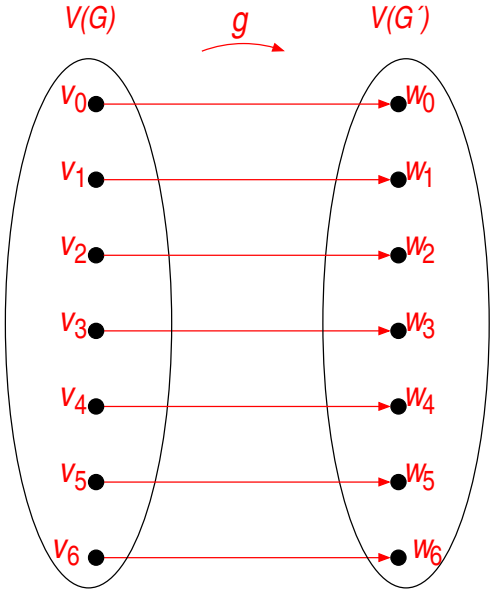
$$h : E(G) \rightarrow E(G')$$

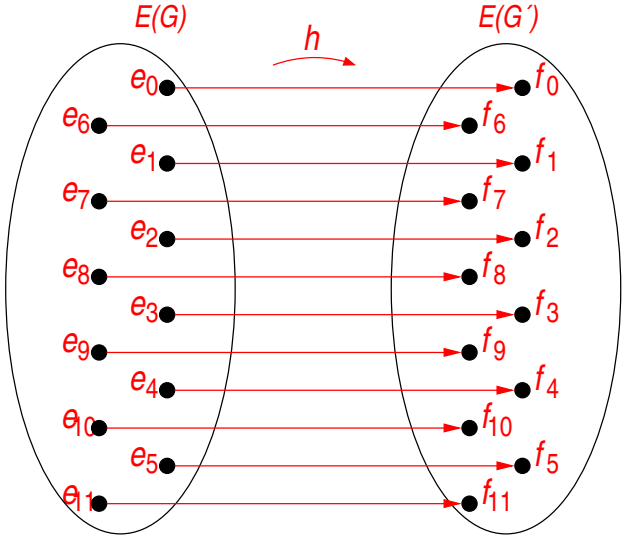
tal que exista a correspondência como mencionado anteriormente.

➜ Grafos G e G' são isomorfos.









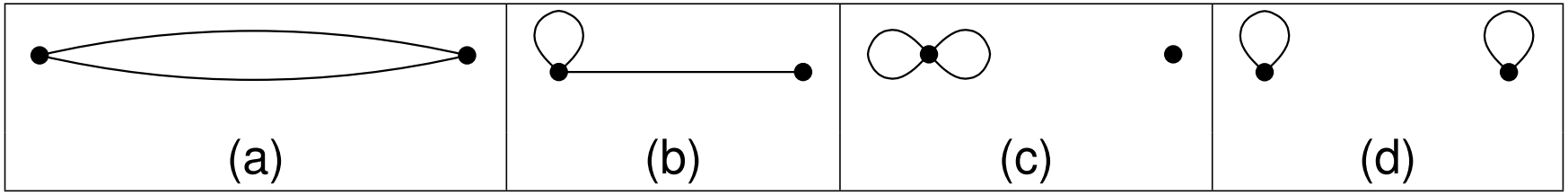
Isomorfismo de grafos (8)

- Isomorfismo de grafos é uma relação de equivalência no conjunto de grafos.

- Informalmente, temos que esta propriedade é:
  - Reflexiva: Um grafo é isomorfo a si próprio.
  - Simétrica: Se um grafo G é isomorfo a um grafo G' então G' é isomorfo a G.
  - Transitiva: Se um grafo G é isomorfo a um grafo G' e G' é isomorfo a G'' então G é isomorfo a G''.

Representantes de classes de isomorfismo (1)

Ache todos os grafos não isomorfos que têm dois vértices e duas arestas.



- Existe um algoritmo que aceita como entrada os grafos G e G' e produz como resultado uma afirmação se estes grafos são isomorfos ou não?
  - Sim. Gere todas as funções g e h e determine se elas preservam as funções aresta–vértice de G e G'.

Representantes de classes de isomorfismo (2)

- Se G e G' têm cada um n vértices e m arestas, o número de funções g é $n!$ e o número de funções h é $m!$, o que dá um número total de $n! \cdot m!$ funções.

- Exemplo para $n = m = 20$.
  - Temos $20! \cdot 20! \approx 5,9 \times 10^{36}$ pares a verificar.
  - Assumindo que cada combinação possa ser achada e calculada em apenas 1μs, seria necessário aproximadamente $1,9 \times 10^{23}$ anos para terminar a computação nesse computador.

Invariantes para isomorfismo de grafos (1)

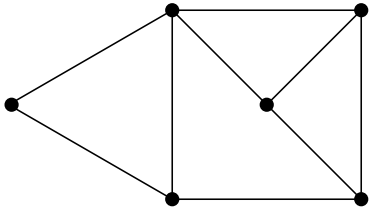
Teorema: Cada uma das seguintes propriedades é uma invariante para isomorfismo de dois grafos G e G', onde n, m e k são inteiros não negativos:

1. Tem n vértices;
2. Tem m arestas;
3. Tem um vértice de grau k;
4. Tem m vértices de grau k;
5. Tem um circuito de tamanho k;
6. Tem um circuito simples de tamanho k;
7. Tem m circuitos simples de tamanho k;
8. É conexo;
9. Tem um circuito Euleriano;
10. Tem um circuito Hamiltoniano.

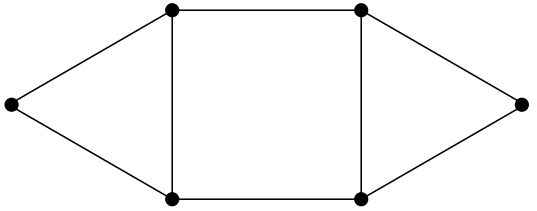
Isto significa que se G é isomorfo a G' então se G tem uma destas propriedades G' também tem.

Invariantes para isomorfismo de grafos (2)

Os grafos



G



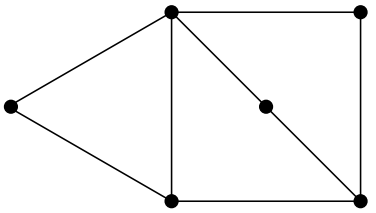
G´

são isomorfos?

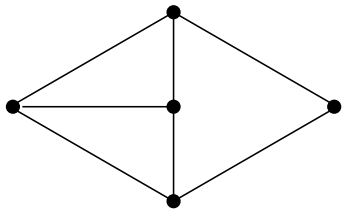
Não. G tem nove arestas e G' tem oito arestas.

Invariantes para isomorfismo de grafos (3)

Os grafos



H



H´

são isomorfos?

Não. H tem um vértice de grau 4 e H' não tem.

Isomorfismo de grafos simples (1)

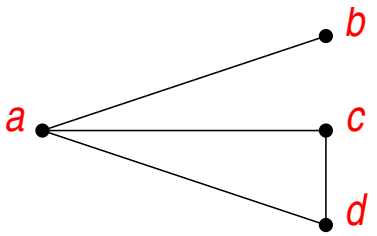
Definição: Se G e G' são grafos simples (sem arestas paralelas e sem laços) então G é isomorfo a G' sse existe uma correspondência g um-para-um do conjunto de vértices $V(G)$ de G para o conjunto de vértices $V(G')$ de G' que preserva a função aresta–vértice de G e de G' no sentido que

$$\forall \text{ vértices } u, v \in G$$

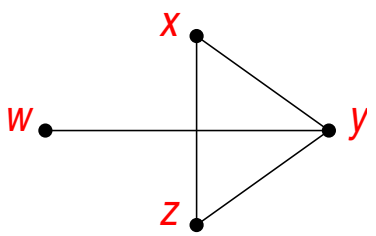
$$uv \text{ é uma aresta de } G \Leftrightarrow \{g(u), g(v)\} \text{ é uma aresta de } G'.$$

Isomorfismo de grafos simples (2)

Os grafos



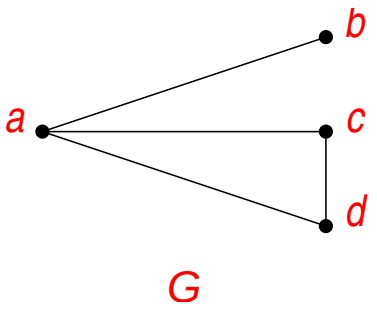
G

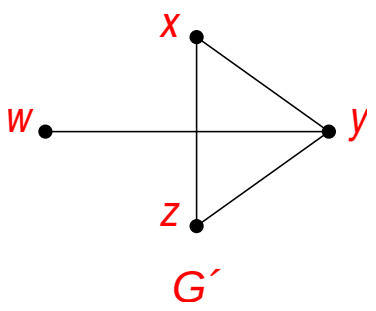


G´

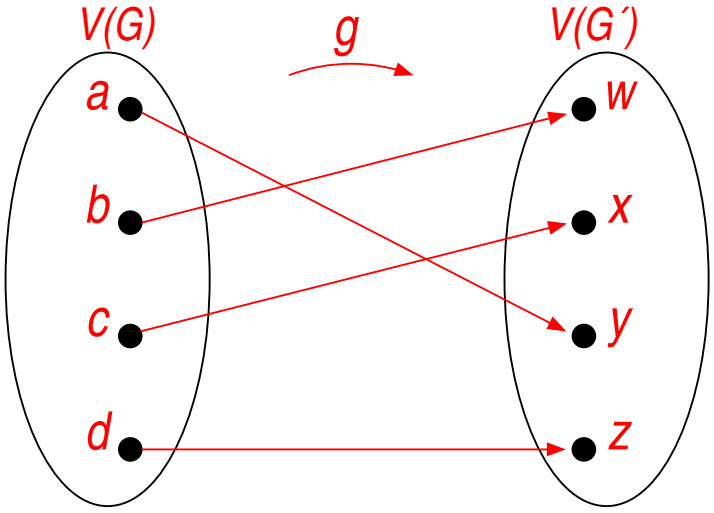
são isomorfos?

Isomorfismo de grafos simples (3)





Sim, são isomorfos.

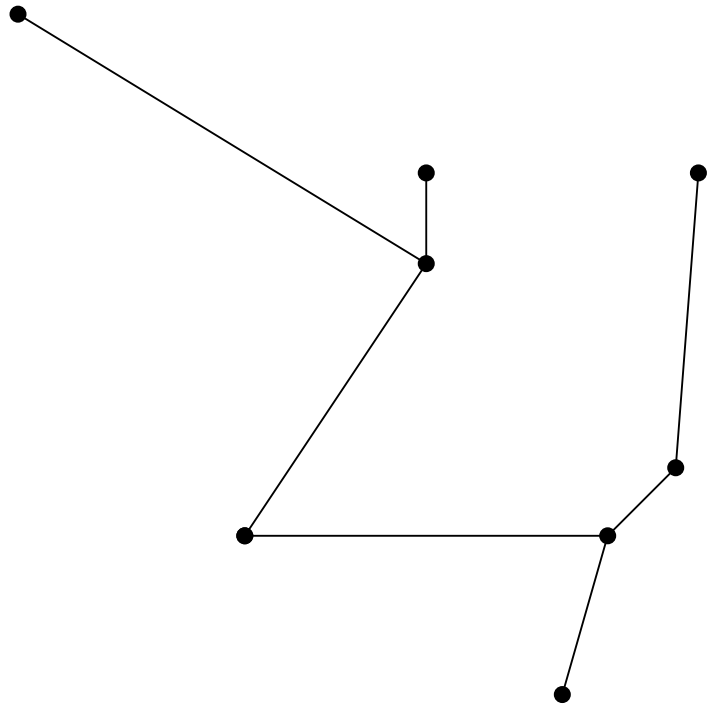


A função g preserva a função aresta–vértice de G e de G':

Arestas de G	**Arestas de G'**
ab	$yw = \{g(a), g(b)\}$
ac	$yx = \{g(a), g(c)\}$
ad	$yz = \{g(a), g(d)\}$
cd	$xz = \{g(c), g(d)\}$

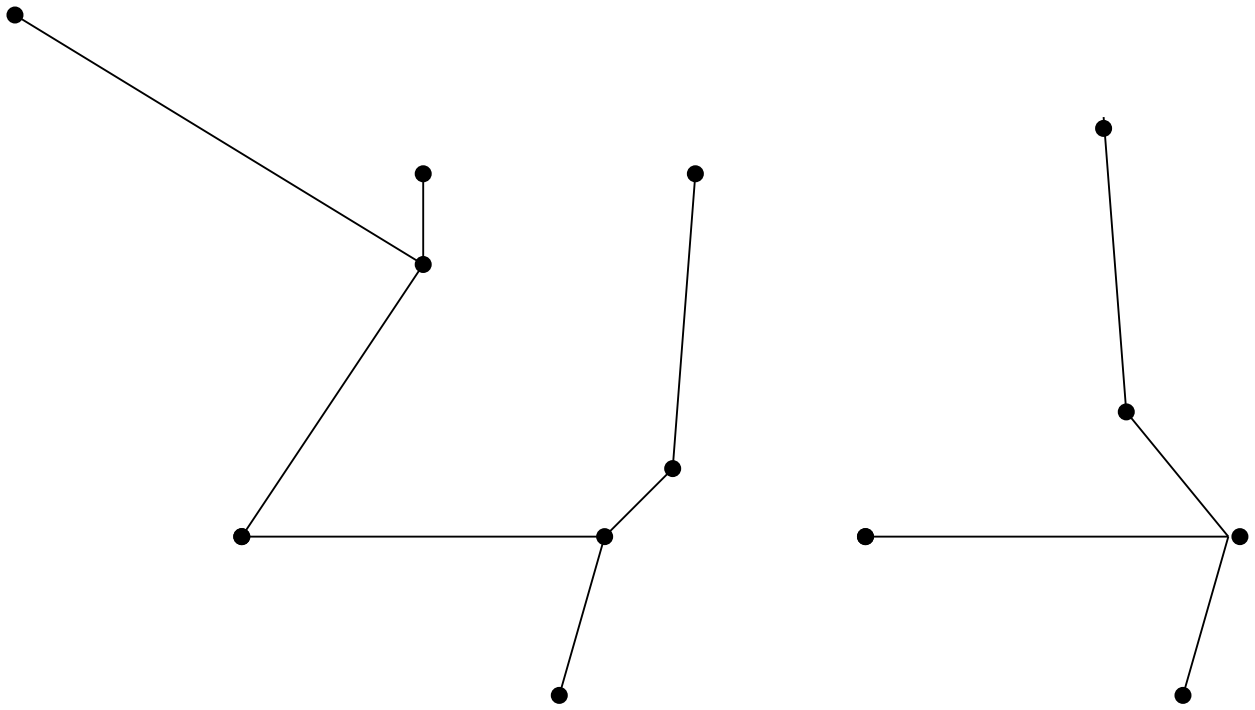
Árvore

Definição: Uma árvore (também chamada de árvore livre) é um grafo não dirigido acíclico e conexo.



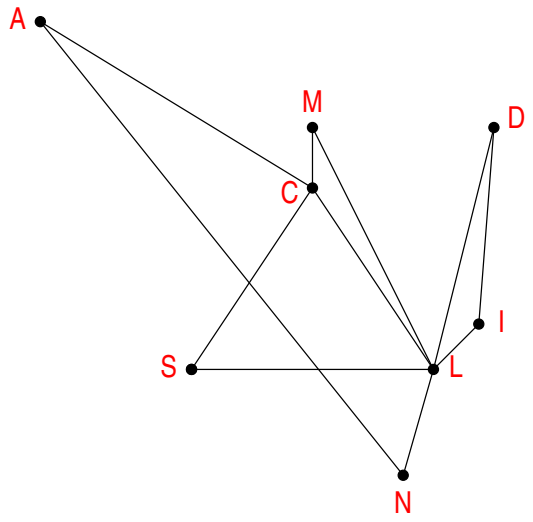
Floresta

Definição: Uma floresta é um grafo não dirigido acíclico podendo ou não ser conexo.

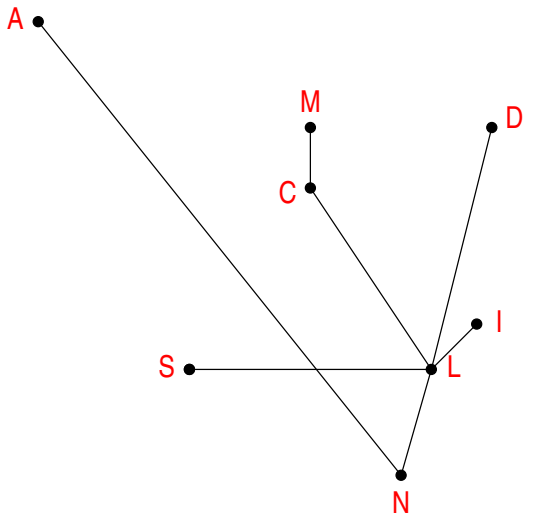


Árvore geradora (1)

Suponha que uma companhia aérea recebeu permissão para voar nas seguintes rotas:



No entanto, por questões de economia, esta empresa irá operar apenas nas seguintes rotas:



Este conjunto de rotas interconecta todas as cidades.

➜ Este conjunto de rotas é mínimo?

– Sim! Qualquer árvore deste grafo possui oito vértices e sete arestas.

Árvore geradora (2)

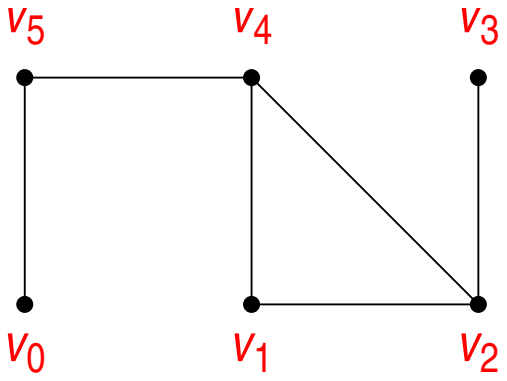
Definição: Uma **árvore geradora** de um grafo G é um grafo que contém cada vértice de G e é uma árvore.

Esta definição pode ser estendida para floresta geradora.

- Proposição:
  - Cada grafo conexo tem uma árvore geradora.
  - Duas árvores geradores quaisquer de um grafo têm a mesma quantidade de arestas.

Árvore geradora (3)

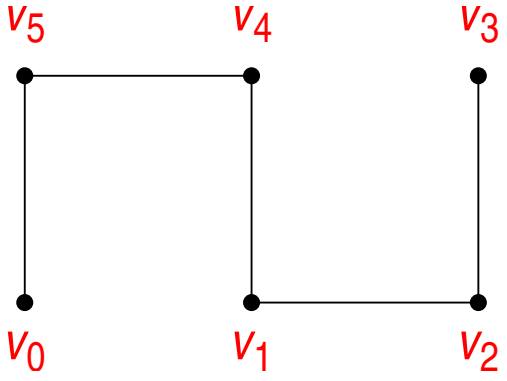
Seja o grafo G abaixo

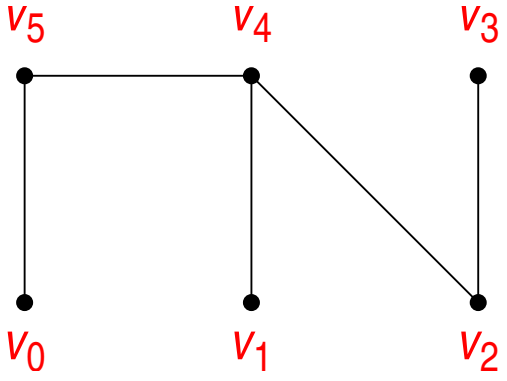


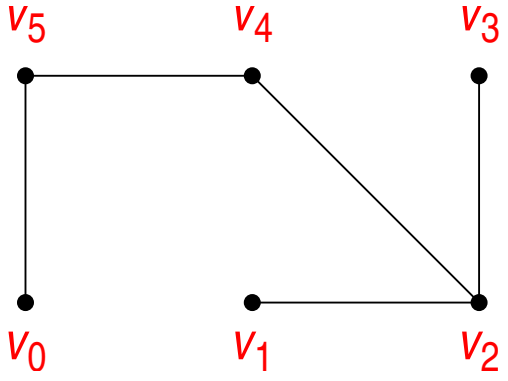
Este grafo possui o circuito $v_2v_1v_4v_2$.

A remoção de qualquer uma das três arestas do circuito leva a uma árvore.

Assim, todas as três árvores geradoras são:

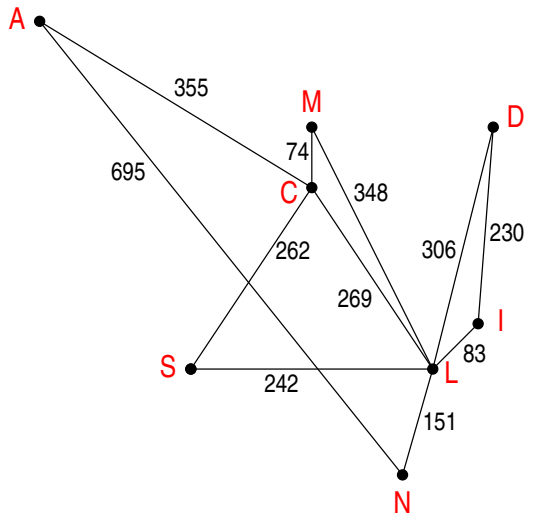




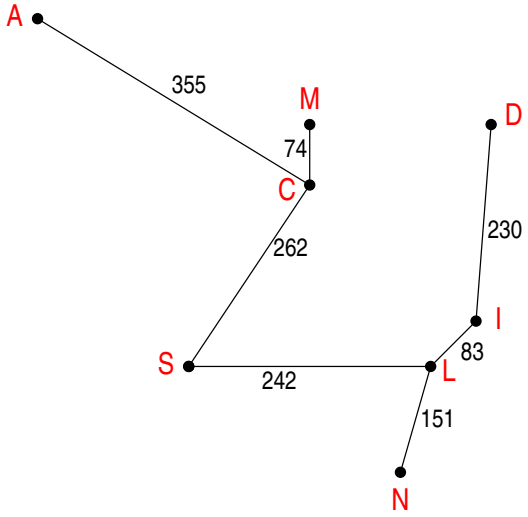


Árvore geradora mínima ou *Minimal Spanning Tree* (1)

O grafo de rotas da companhia aérea que recebeu permissão para voar pode ser "rotulado" com as distâncias entre as cidades:



Suponha que a companhia deseja voar para todas as cidades mas usando um conjunto de rotas que minimiza o total de distâncias percorridas:



Este conjunto de rotas interconecta todas as cidades.

Árvore geradora mínima (2)

Definição: Um **grafo com peso** é um grafo onde cada aresta possui um peso representado por um número real. A soma de todos os pesos de todas as arestas é o peso total do grafo. Uma **árvore geradora mínima** para um grafo com peso é uma árvore geradora que tem o menor peso total possível dentre todas as possíveis árvores geradoras do grafo.

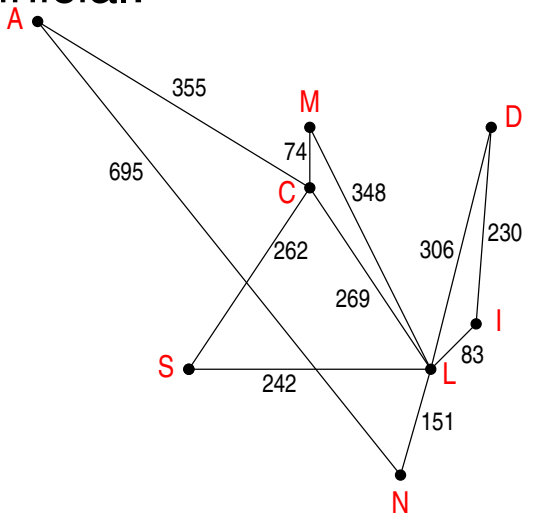
Se G é um grafo com peso e e uma aresta de G então:

- $w(e)$ indica o peso da aresta e, e
- $w(G)$ indica o peso total do grafo G.

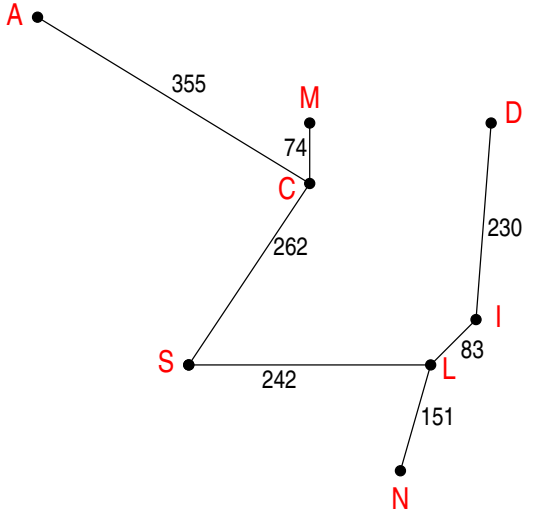
Algoritmos para obter a árvore geradora mínima

- Algoritmo de Kruskal.
- Algoritmo de Prim.

Grafo inicial:



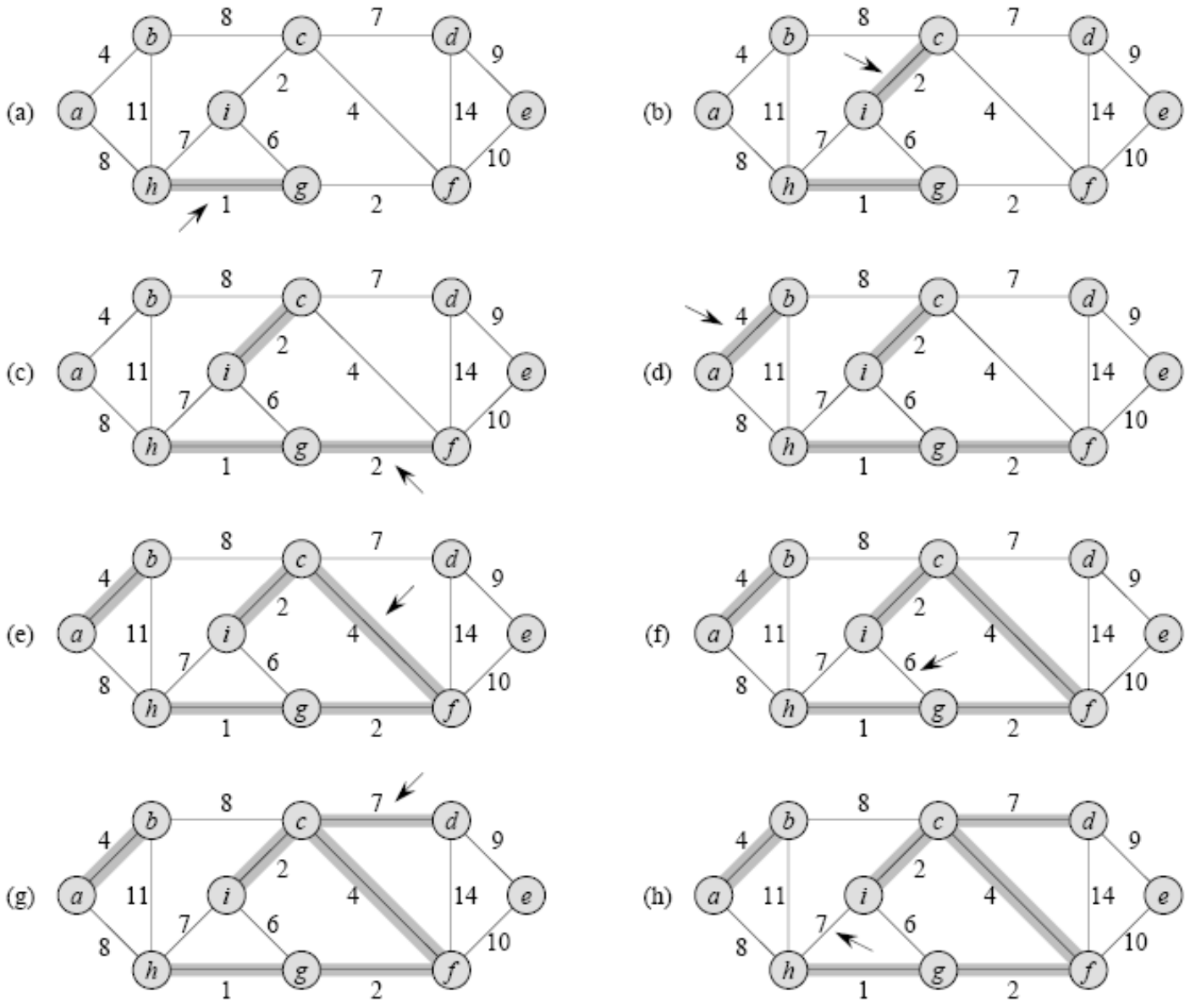
Árvore geradora mínima:



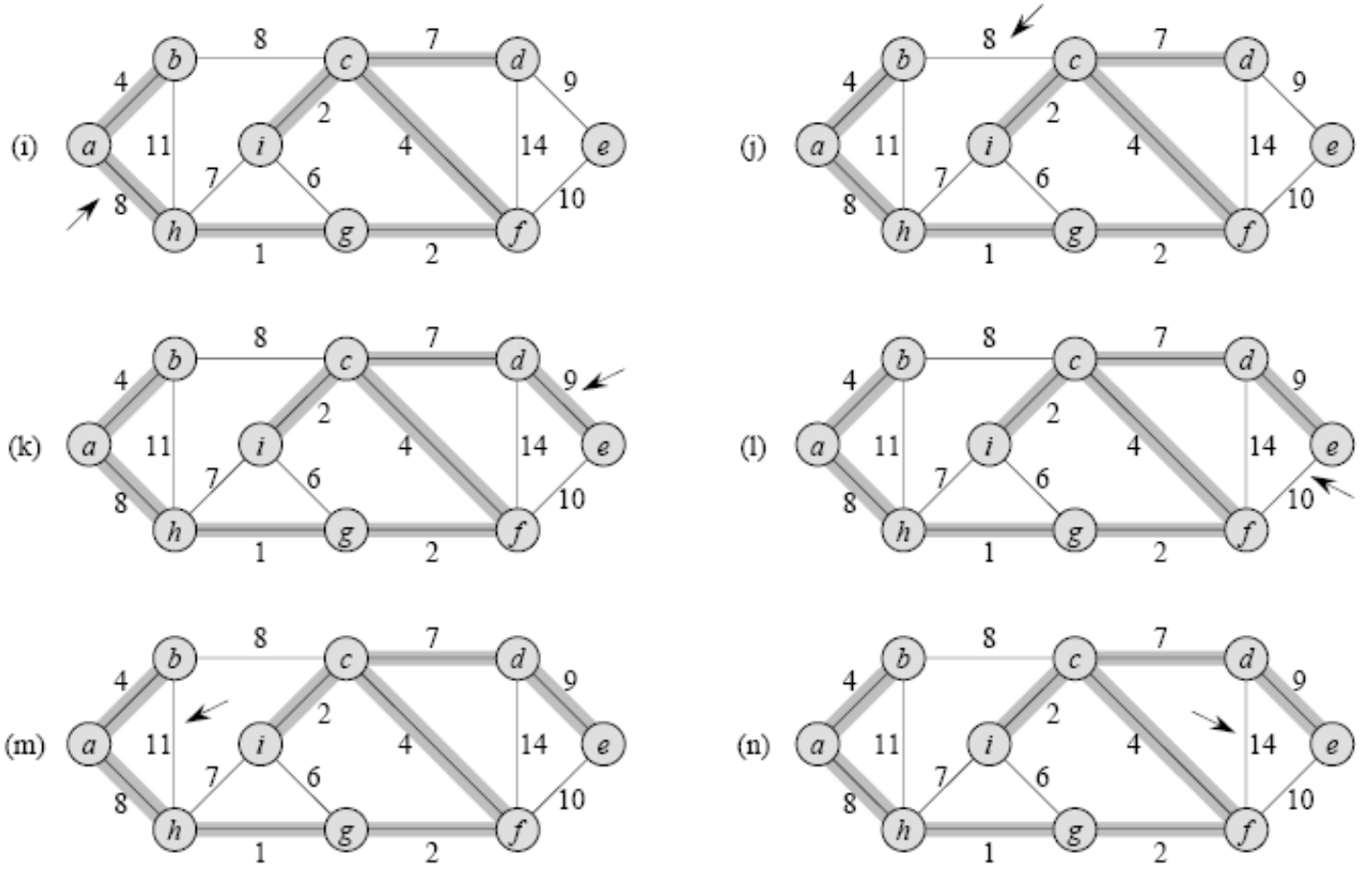
Algoritmo de Kruskal (1)

- Idéia básica:
  - Seleciona a aresta de menor peso que conecta duas árvores de uma floresta.
  - Repita o processo até que todos os vértices estejam conectados sempre preservando a invariante de se ter uma árvore.

Algoritmo de Kruskal (2)

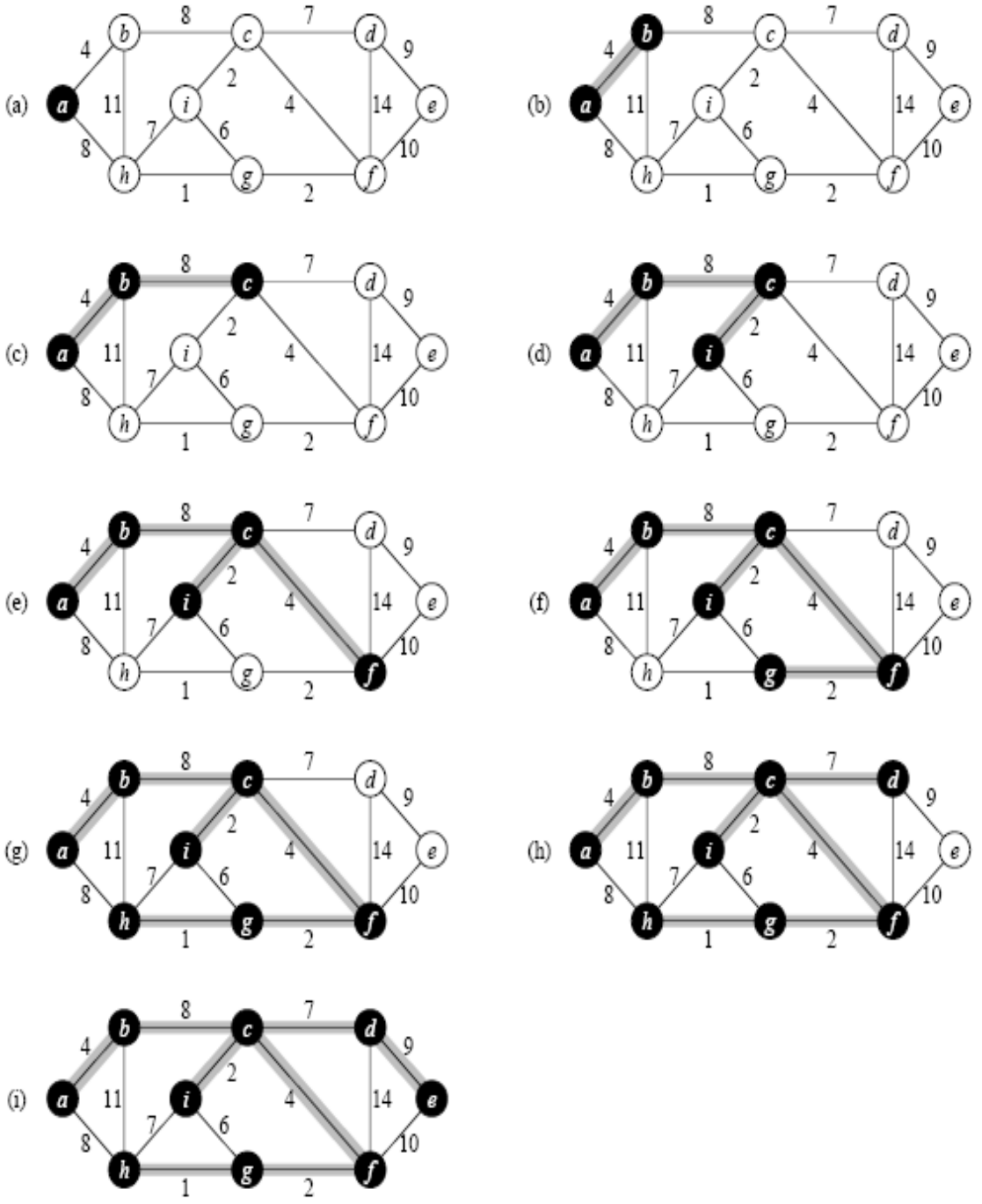


Algoritmo de Kruskal (3)



Algoritmo de Prim

- Idéia básica:
  - Tomando como vértice inicial A, crie uma fila de prioridades classificada pelos pesos das arestas conectando A.
  - Repita o processo até que todos os vértices tenham sido visitados.



Algoritmos de pesquisa em grafo

- Objetivo:
  - Pesquisa sistemática de cada aresta e vértice de um grafo.
- Grafo $G = (V, E)$ pode ser tanto dirigido quanto não dirigido.
- Os algoritmos apresentados assumem que a estrutura de dados utilizada é uma lista de adjacência.
- Exemplos de algoritmos de pesquisa em grafo:
  - Pesquisa em profundidade (*Depth-first search – DFS*).
  - Pesquisa em largura (*Breadth-first search – BFS*).
- Aplicações:
  - Computação gráfica.
  - Técnicas de verificação formal.
  - Compiladores.
  - Resolução de problemas.
  - ...

Pesquisa em largura (1)

- Seja um grafo $G = (V, E)$ e um vértice origem s.

- Pesquisa em largura:
  - Descobre as arestas em G que são alcançáveis a partir de s.
  - Computa a distância (em nº de arestas) de s para os vértices que são alcançáveis.
  - Produz uma árvore em largura com raiz s e seus vértices alcançáveis.
  - Se v é um vértice alcançável a partir de s, então o caminho entre s e v na árvore corresponde ao caminho mais curto entre s e v no grafo G.

Pesquisa em largura (2)

- Expande a fronteira entre vértices descobertos e não descobertos uniformemente através da extensão (largura) da fronteira.

- Pesquisa descobre todos os vértices que estão a distância k antes de descobrir os vértices a distância $k + 1$.

Algoritmo para calcular BFS (1)

- Estruturas de dados:
  - Grafo representado como uma lista de adjacência.
  - Vetor *color*[u]: cor de cada vértice $u \in V$.
  - Vetor $\pi[u]$: predecessor de cada vértice $u \in V$. Se u não tem predecessor ou ainda não foi descoberto então $\pi[u]$ = nil.
  - Vetor *d*[u]: distância de cada vértice $u \in V$ ao vértice s.
  - Fila Q: contém os vértices já descobertos em largura.

- Cores dos vértices:
  - *white*: não visitados ainda.
  - *gray*: vértice descoberto mas que não teve a sua lista de adjacência totalmente examinada.
  - *black*: vértice descoberto que já teve a sua lista de adjacência totalmente examinada.
    ➜ garantir que a pesquisa caminha em largura

- Funções:
  - Enqueue e Dequeue: operações sobre uma fila FIFO.

Algoritmo para calcular BFS (2)

```
BFS(G, s)
 1  for each vertex u ∈ V[G] − {s}
 2  do color[u] ← white
 3       d[u] ← ∞
 4       π[u] ← nil
 5  color[s] ← gray
 6  d[s] ← 0
 7  π[s] ← nil
 8  Q ← {s}
 9  while Q ≠ ∅
10  do u ← head[Q]
11       for each v ∈ Adj[u]
12       do if color[v] = white
13            then color[v] ← gray
14                 d[v] ← d[u] + 1
15                 π[v] ← u
16                 Enqueue(Q, v)
17       Dequeue(Q)
18       color[u] ← black
```

Comentários e análise do algoritmo (1)

1 **for** each vertex $u \in V[G] - \{s\}$
2 **do** $color[u] \leftarrow$ white
3 $\quad d[u] \leftarrow \infty$
4 $\quad \pi[u] \leftarrow$ **nil**

Linhas 1–4: Inicialização I
- Inicialização de cada vértice para *white* (não descoberto).
- Distância para o vértice s como ∞.
- Predecessor do vértice desconhecido.

Análise:
- Linhas 1–4: $O(V)$.
- Cada vértice (exceto s) é inicializado como *white* e nenhum vértice volta a ser *white*.

---

5 $color[s] \leftarrow$ gray
6 $d[s] \leftarrow 0$
7 $\pi[s] \leftarrow$ **nil**
8 $Q \leftarrow \{s\}$

Linhas 5–8: Inicialização II
- Inicialização de s com *gray* (é considerado descoberto).
- Distância para o próprio vértice s é 0.
- Não possui predecessor.
- A fila Q contém inicialmente apenas s (irá conter apenas os vértices *gray*, ou seja, os vértices descobertos em "largura").

Análise:
- Linhas 5–8: $O(1)$.

Comentários e análise do algoritmo (2)

```
 9  while Q ≠ ∅
10  do u ← head[Q]
11      for each v ∈ Adj[u]
12      do if color[v] = white
13          then color[v] ← gray
14               d[v] ← d[u] + 1
15               π[v] ← u
16               Enqueue(Q, v)
17      Dequeue(Q)
18      color[u] ← black
```

Linhas 9–18: *Loop*

- *Loop* é executado até que a fila esteja vazia, ou seja, não hajam vértices *gray*.
- Cada vértice é colocado e retirado da fila somente uma única vez.
- Vértice *gray* é um vértice que foi descoberto mas a sua lista de adjacência ainda não foi totalmente descoberta.
- Vértice *black* já foi totalmente examinado.
- Vértice u contém o primeiro elemento da fila Q.
- Linhas 11–16 examinam cada vértice v adjacente a u que não foi descoberto ainda (*white* (12)), marca como descoberto (*gray* (13)), calcula a sua distância até s (14), marca o seu predecessor como u (15), e o coloca na fila Q (16).
- Quando todos os vértices adjacentes a u forem examinados, u é retirado da fila Q (17) e passa a ser *black* (18).

Comentários e análise do algoritmo (3)

```
 9  while Q ≠ ∅
10  do u ← head[Q]
11      for each v ∈ Adj[u]
12      do if color[v] = white
13          then color[v] ← gray
14               d[v] ← d[u] + 1
15               π[v] ← u
16               Enqueue(Q, v)
17      Dequeue(Q)
18      color[u] ← black
```

Análise:

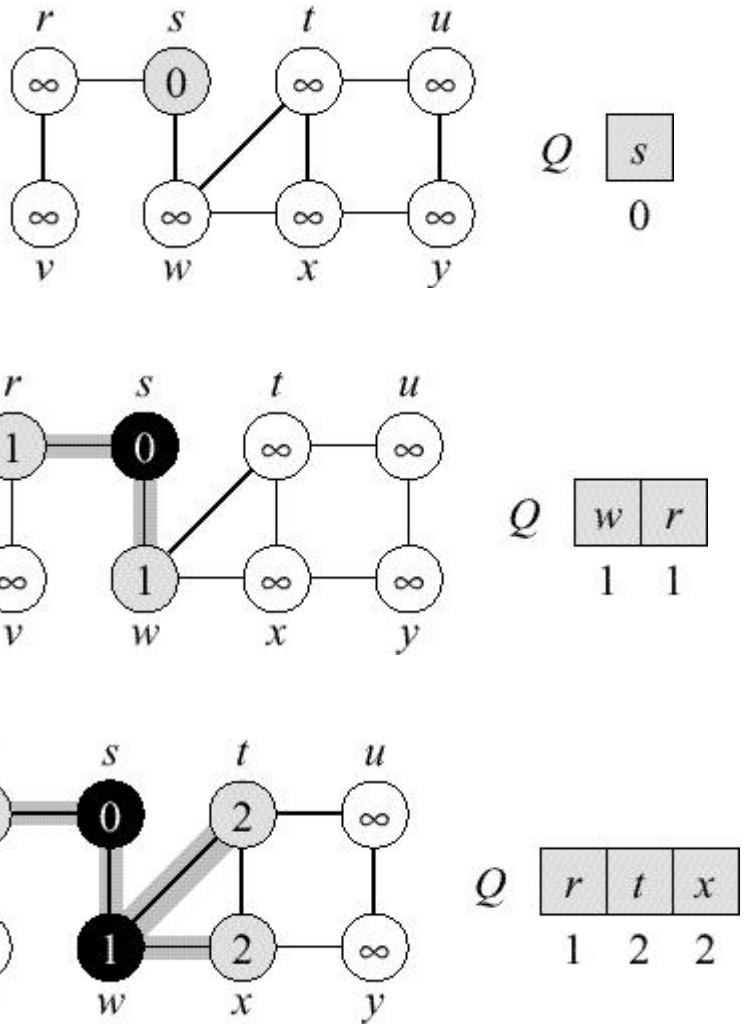
- Operações de colocar e retirar da fila cada vértice é $O(1)$ e o tempo total relacionado com as operações de fila é $O(V)$.
- A lista de adjacência de cada vértice u é percorrida somente uma única vez quando esse vértice será retirado de Q.
- A soma dos comprimentos das filas de adjacência é $O(E)$, assim como o tempo para percorrê-la.
- Linhas 9–18: $O(V + E)$, que é o custo das operações associadas à fila e a percorrer as listas de adjacência.

Análise de todo algoritmo:

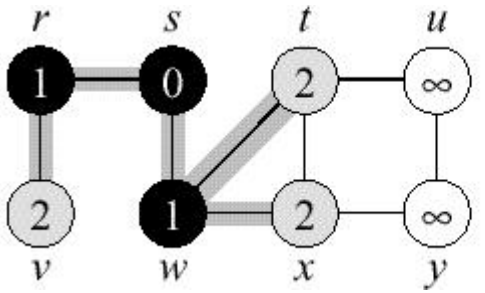
- Linhas 1–4: $O(V)$.
- Linhas 5–8: $O(1)$.
- Linhas 9–18: $O(V + E)$.

➜ Custo total: $O(V + E)$.

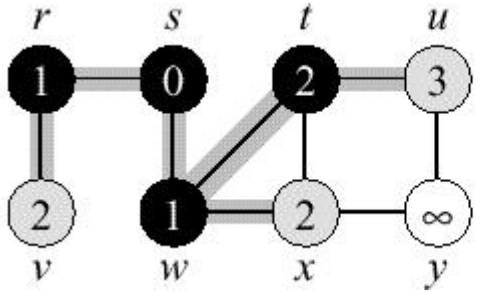
Execução do algoritmo BFS (1)



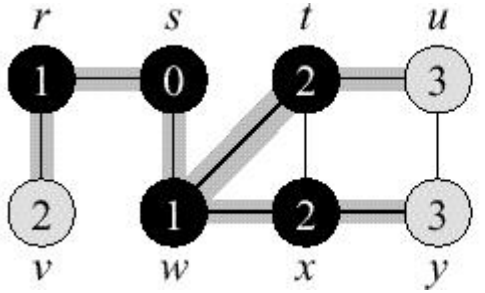
Execução do algoritmo BFS (2)



Q | t | x | v
2 2 2

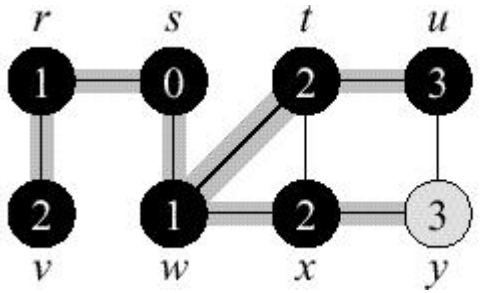


Q | x | v | u
2 2 3



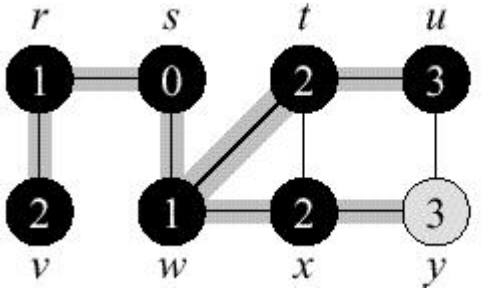
Q | v | u | y
2 3 3

Execução do algoritmo BFS (3)



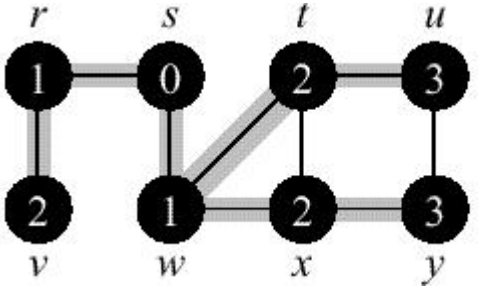
Q y

3



Q y

3



Q $\varnothing$

Caminho mais curto

- BFS calcula a distância (em nº de arestas) de $s \in V$ para os vértices que são alcançáveis em $G = (V, E)$.

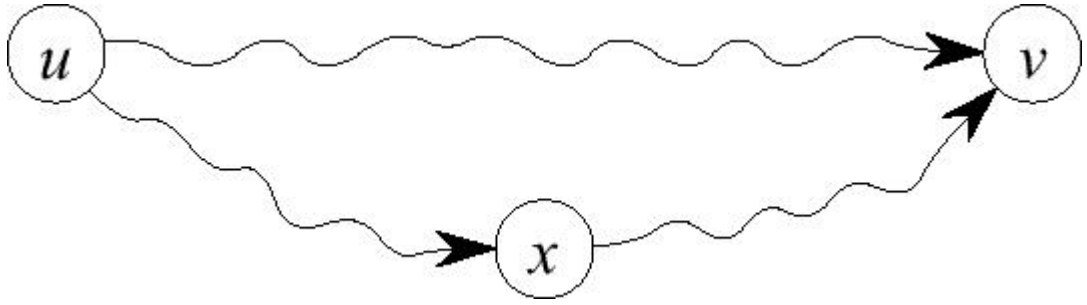
- Caminho mais curto $\delta(s, v)$:
  - Número mínimo de arestas em qualquer caminho de s para v, no caso de ser alcançável, ou ∞ se não for.
  - $d[v] = \delta(s, v)$, para todo $v \in V$.
  - ➜ Caminho mais curto entre s e v.

- Teorema: Sub-caminhos de caminhos mais curtos são caminhos mais curtos.

  Prova: Se algum sub-caminho não gerar o caminho mais curto, ele poderia ser substituído por um sub-caminho atalho e gerar um caminho total mais curto.

Inequação triangular

- Teorema: $\delta(u, v) \leq \delta(u, x) + \delta(x, v)$.
  - Caminho mais curto $u \rightsquigarrow v$ não é mais longo que qualquer outro caminho $u \rightsquigarrow v$ — em particular o caminho concatenando o caminho mais curto $u \rightsquigarrow x$ com o caminho mais curto $x \rightsquigarrow v$.



Árvore em largura (1)

- BFS produz uma árvore em largura com raiz s e seus vértices alcançáveis.
  - Árvore representada pelo vetor π.

- Seja o grafo $G = (V, E)$ e o vértice origem s. O sub-grafo predecessor de G é definido como $G_\pi = (V_\pi, E_\pi)$, onde

$$V_\pi = \{v \in V \mid \pi[v] \neq \textbf{nil}\} \cup \{s\}$$

e

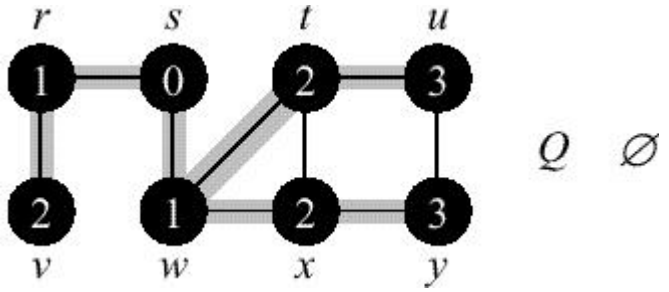
$$E_\pi = \{(\pi[v], v) \in E \mid v \in V_\pi - \{s\}\}$$

Árvore em largura (2)

- O sub-grafo predecessor G_π é uma árvore em largura se:
  - V_π consiste dos vértices alcançáveis a partir de s, $\forall v \in V_\pi$.
  - $\exists$ um caminho simples único de s a v em G_π que também é o caminho mais curto de s a v em G.

  ➜ $|E_\pi| = |V_\pi| - 1$.

- BFS constrói o vetor π tal que sub-grafo predecessor G_π é uma árvore em largura.



Imprime caminho mais curto

Print-Path(G, s, v)

```
1  if v = s
2  then print s
3  else if π[v] = nil
4          then print "no path from" s "to" v "exists"
5          else Print-Path(G, s, π[v])
6                  print v
```

Análise:

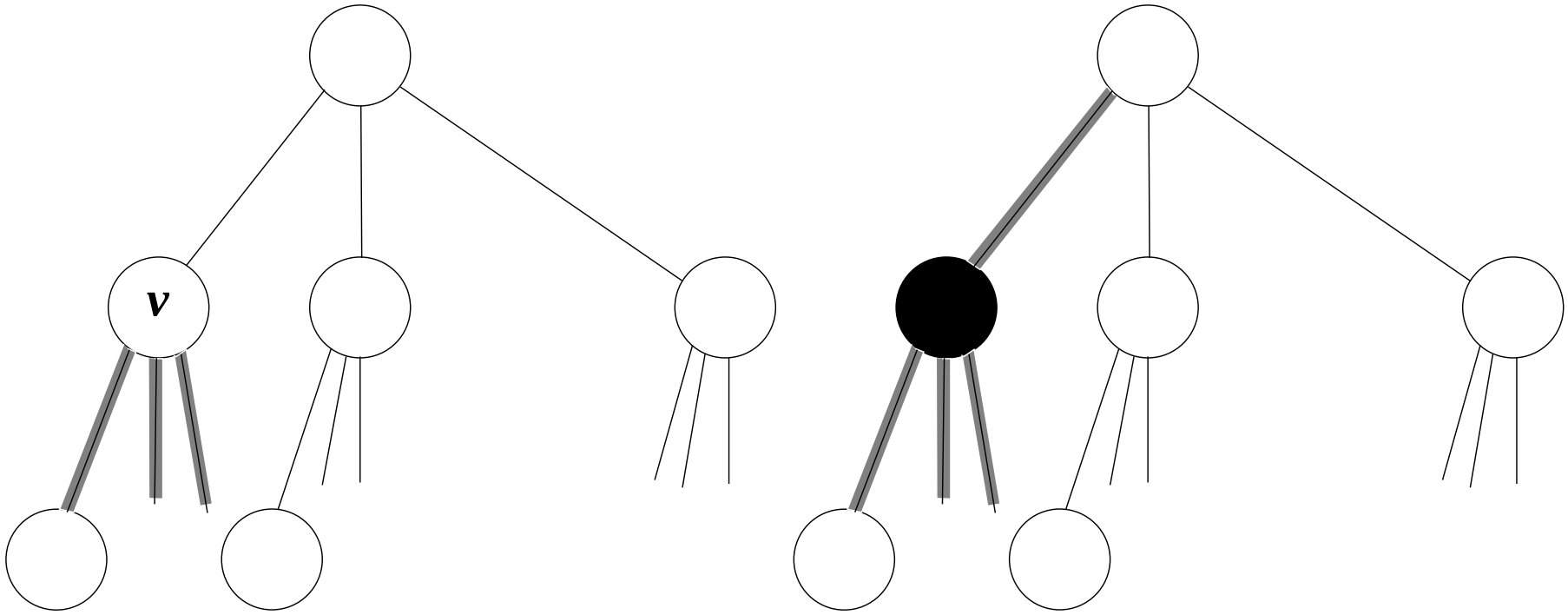
- Executa em tempo $O(V)$.

Pesquisa em profundidade (1)

- Seja um grafo $G = (V, E)$

- Pesquisa em profundidade:
  - Explora os vértices do grafo a partir de arestas não exploradas ainda, mais fundo no grafo quanto possível.
  - Quando todas as arestas de v tiverem sido exploradas, a pesquisa volta para explorar outras arestas do vértice do qual v foi descoberto.
  - Processo continua até que todos os vértices alcançáveis a partir de uma origem tenham sido descobertos.
  - Se existe vértice não descoberto ainda então um novo vértice é selecionado e o processo começa todo novamente.

Pesquisa em profundidade (2)



Algoritmo para calcular DFS (1)

- Estruturas de dados:
  - Grafo representado como uma lista de adjacência.
  - Vetor *color*$[u]$: cor de cada vértice $u \in V$.
  - Vetor $\pi[u]$: predecessor de cada vértice $u \in V$. Se u não tem predecessor ou ainda não foi descoberto então $\pi[u]$ = nil.
  - Vetor $d[1 \ldots |V|]$: marca quando o vértice é descoberto (*white* $\rightarrow$ *gray*).
  - Vetor $f[1 \ldots |V|]$: marca quando o vértice é terminado (*gray* $\rightarrow$ *black*).
  - Variável global time: indica o instante em que o vértice é descoberto e terminado, ou seja, o *timestamp*.

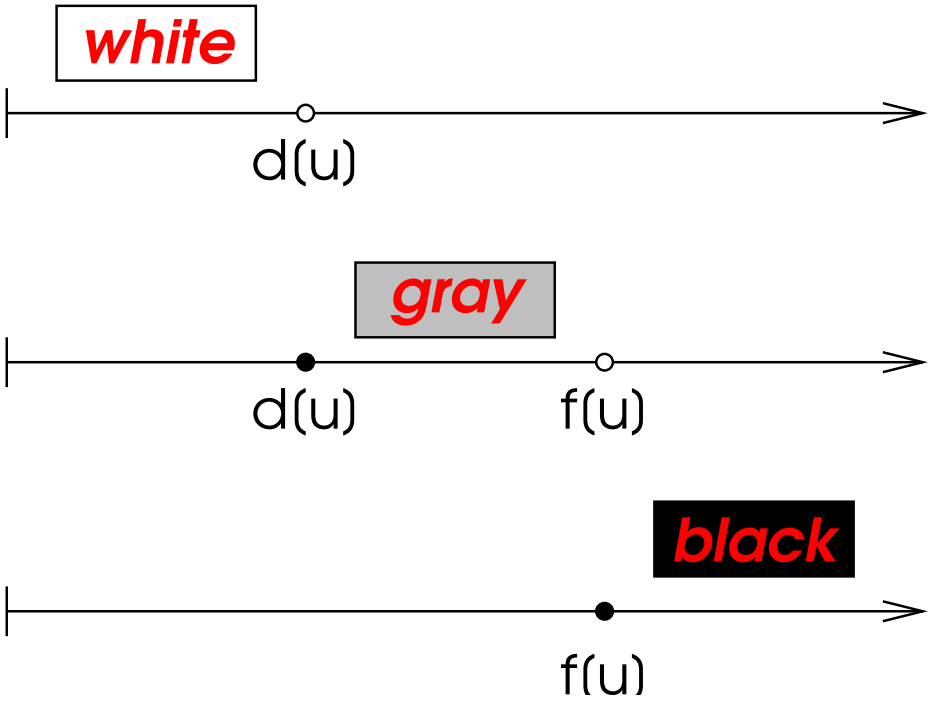
Algoritmo para calcular DFS (2)

- Cores dos vértices:
  - *white*: não visitados ainda.
  - *gray*: vértice descoberto mas que não teve a sua lista de adjacência totalmente examinada.
  - *black*: vértice descoberto que já teve a sua lista de adjacência totalmente examinada e está terminado.

- Função:
  - DFS-Visit: Percorre recursivamente o grafo em profundidade.

Algoritmo para calcular DFS (3)

- Vetores d e f indicam os *timestamps* de "descoberta" e "término" do vértice.
- Valores dos *timestamps* estão entre 1 e $2|V|$
  - $\forall u \in V, \exists$ um único evento de descoberta e um único evento de término.
  - $\forall u \in V, d[u] \prec f[u] \Rightarrow d[u] < f[u]$
- $\forall u \in V$, tempo lógico:



Algoritmo para calcular DFS (4)

DFS(G)

```
1  for each vertex u ∈ V[G]
2  do color[u] ← white
3       π[u] ← nil
4  time ← 0
5  for each vertex u ∈ V[G]
6  do if color[u] = white
7       then DFS-Visit(u)
```

DFS-Visit(u)

```
1  color[u] ← gray
2  d[u] ← time ← time +1
3  for each v ∈ Adj[u]
4       do if color[v] = white
5            then π[v] ← u
6                 DFS-Visit(v)
7  color[u] ← black
8  f[u] ← time ← time +1
```

Comentários e análise do algoritmo (1)

DFS(G)
1 **for** each vertex $u \in V[G]$
2 **do** *color*$[u] \leftarrow$ white
3 $\pi[u] \leftarrow$ **nil**
4 *time* $\leftarrow 0$

Linhas 1–4: Inicialização I
- Inicialização de cada vértice para *white* (não descoberto).
- Predecessor do vértice desconhecido.

Linha 4: Inicialização II
- Variável global usada para indicar o *timestamp*.

Análise:
- Linhas 1–3: $O(V)$.
- Cada vértice é inicializado como *white* e nenhum vértice volta a ser *white*.
- Linha 4: $O(1)$.

Comentários e análise do algoritmo (2)

DFS(G) (continuação)
5 **for** each vertex $u \in V[G]$
6 **do if** *color*$[u]$ = white
7 **then** DFS-Visit(u)

Linhas 5–7: *Loop*

- Para cada vértice ainda não descoberto (linha 6) faz a pesquisa em profundidade (linha 7).

Análise:

- Linhas 5–7: $O(V + E)$.
- As linhas 5–7 da pesquisa DFS gastam $O(V)$ sem contar o tempo do procedimento DFS-Visit, que só é chamado uma única vez para cada vértice branco.
- O procedimento DFS-Visit executa em $O(E)$.

Comentários e análise do algoritmo (3)

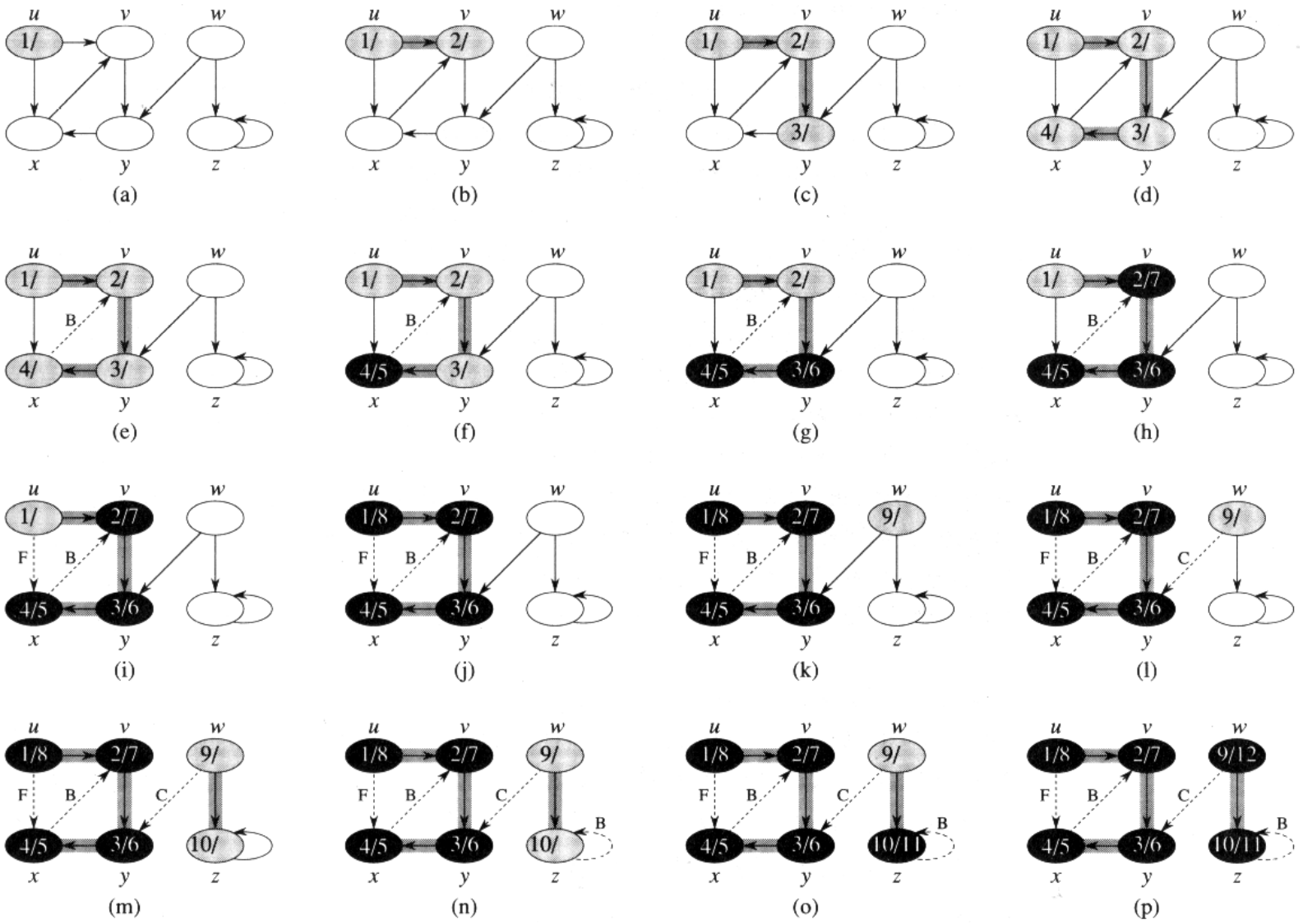
DFS-Visit(u)
1 *color*$[u] \leftarrow$ gray
2 $d[u] \leftarrow$ *time* $\leftarrow$ *time* $+1$
3 **for** each $v \in$ *Adj*$[u]$
4 **do if** *color*$[v]$ = white
5 **then** $\pi[v] \leftarrow u$
6 DFS-Visit(v)
7 *color*$[u] \leftarrow$ black
8 $f[u] \leftarrow$ *time* $\leftarrow$ *time* $+1$

Linhas 1, 2, 7, 8: Atribuições
- Inicialização de cada vértice para *white* (não descoberto).
- Predecessor do vértice desconhecido.

Linha 4: Inicialização II
- Variável global usada para indicar o *timestamp*.

Análise:
- Linhas 1–3: $O(V)$.
- Cada vértice é inicializado como *white* e nenhum vértice volta a ser *white*.
- Linha 4: $O(1)$.

Execução do algoritmo DFS



O TAD Grafo

- Importante considerar os algoritmos em grafos como tipos abstratos de dados.
- Conjunto de operações associado a uma estrutura de dados.
- Independência de implementação para as operações.

Operadores do TAD Grafo

1. *FGVazio*(*Grafo*): Cria um grafo vazio.
2. *InsereAresta*(*V1, V2, Peso, Grafo*): Insere uma aresta no grafo.
3. *ExisteAresta*(*V1, V2, Grafo*): Verifica se existe uma determinada aresta.
4. Obtém a lista de vértices adjacentes a um determinado vértice.
5. *RetiraAresta*(*V1, V2, Peso, Grafo*): Retira uma aresta do grafo.
6. *LiberaGrafo*(*Grafo*): Libera o espaço ocupado por um grafo.
7. *ImprimeGrafo*(*Grafo*): Imprime um grafo.
8. *GrafoTransposto*(*Grafo, GrafoT*): Obtém o transposto de um grafo dirigido.
9. *RetiraMin*(*A*): Obtém a aresta de menor peso de um grafo com peso nas arestas.

Operação “Obter Lista de Adjacentes”

1. *ListaAdjVazia*(*v, Grafo*): retorna *true* se a lista de adjacentes de v está vazia.
2. *PrimeiroListaAdj*(*v, Grafo*): retorna o endereço do primeiro vértice na lista de adjacentes de v.
3. *ProxAdj*(*v, Grafo, u, Peso, Aux, FimListaAdj)*: retorna o vértice u (apontado por *Aux*) da lista de adjacentes de v, bem como o peso da aresta (v, u). Ao retornar, *Aux* aponta para o próximo vértice da lista de adjacentes de v, e *FimListaAdj* retorna *true* se o final da lista de adjacentes foi encontrado.

Implementação da operação "Obter Lista de Adjacentes"

É comum encontrar um pseudo comando do tipo:

for $u \in$ *ListaAdjacentes*(v) **do** {faz algo com u}

O trecho de programa abaixo apresenta um possível refinamento do pseudo-comando acima.

```
if not ListaAdjVazia(v, Grafo)
then begin
       Aux := PrimeiroListaAdj(v, Grafo);
       FimListaAdj := false;
       while not FimListaAdj
       do ProxAdj(v, Grafo, u, Peso, Aux, FimListaAdj);
     end;
```

Matriz de adjacência: Implementação

A inserção de um novo vértice ou retirada de um vértice já existente pode ser realizada com custo constante.

```
const
  MaxNumVertices   = 100;
  MaxNumArestas    = 4500;
type
  TipoValorVertice = 0..MaxNumVertices;
  TipoPeso         = integer;
  TipoGrafo        = record
                       Mat: array[TipoValorVertice, TipoValorVertice] of TipoPeso;
                       NumVertices: 0..MaxNumVertices;
                       NumArestas : 0..MaxNumArestas;
                     end;
  Apontador        = TipoValorVertice;

procedure FGVazio(var Grafo: TipoGrafo);
var i, j: integer;
begin
  for i := 0 to Grafo.NumVertices do
    for j := 0 to Grafo.NumVertices do
      Grafo.mat[i, j] := 0;
end;  {FGVazio}
```

Matriz de adjacência: Implementação

```
procedure InsereAresta (var V1, V2: TipoValorVertice;
                        var Peso  : TipoPeso;
                        var Grafo : TipoGrafo);
begin
  Grafo.Mat[V1, V2] := peso;
end;   {InsereAresta}
```

```
function ExisteAresta (    Vertice1,
                           Vertice2: TipoValorVertice;
                       var Grafo:    TipoGrafo) : boolean;
begin
  ExisteAresta := Grafo.Mat[Vertice1, Vertice2] > 0;
end;   {ExisteAresta}
```

Matriz de adjacência: Implementação

```
{Operador para obter a lista de adjacentes}
function ListaAdjVazia (var Vertice: TipoValorVertice;
                        var Grafo:   TipoGrafo): boolean;
var Aux:         Apontador;
    ListaVazia: boolean;
begin
  ListaVazia := true;
  Aux := 0;
  while (Aux < Grafo.NumVertices) and ListaVazia do
    if Grafo.Mat[Vertice, Aux] > 0
    then ListaVazia := false
    else Aux := Aux + 1;
  ListaAdjVazia := ListaVazia = true;
end;   {ListaAdjVazia}
```

Matriz de adjacência: Implementação

```
{Operador para obter a lista de adjacentes}
function PrimeiroListaAdj (var Vertice: TipoValorVertice;
                           var Grafo:   TipoGrafo): Apontador;
var Aux:         Apontador;
    Listavazia: boolean;
begin
  ListaVazia := true;  Aux := 0;
  while (Aux < Grafo.NumVertices) and ListaVazia do
    if Grafo.Mat[Vertice, Aux] > 0
    then begin
           PrimeiroListaAdj := Aux;
           ListaVazia := false;
         end
    else Aux := Aux + 1;
  if Aux = Grafo.NumVertices
  then writeln('Erro: Lista adjacência vazia (PrimeiroListaAdj)');
end;  {PrimeiroListaAdj}
```

Matriz de adjacência: Implementação

```
{Operador para obter a lista de adjacentes}
procedure ProxAdj (var Vertice:     TipoValorVertice;
                   var Grafo:       TipoGrafo;
                   var Adj:         TipoValorVertice;
                   var Peso:        TipoPeso;
                   var Prox:        Apontador;
                   var FimListaAdj: boolean);
{Retorna Adj apontado por Prox}
begin
  Adj  := Prox;
  Peso := Grafo.Mat[Vertice, Prox];
  Prox := Prox + 1;
  while (Prox < Grafo.NumVertices) and
        (Grafo.Mat[Vertice, Prox] = 0) do
    Prox := Prox + 1;
  if Prox = Grafo.NumVertices then FimListaAdj := true;
end;   {ProxAdj}
```

Matriz de adjacência: Implementação

```
procedure RetiraAresta (var V1, V2: TipoValorVertice;
                        var Peso  : TipoPeso;
                        var Grafo : TipoGrafo);
begin
  if Grafo.Mat[V1, V2] = 0
  then writeln('Aresta não existe')
  else begin
         Peso := Grafo.Mat[V1, V2];
         Grafo.Mat[V1, V2] := 0;
       end;
end;   {RetiraAresta}
```

```
procedure LiberaGrafo (var Grafo: TipoGrafo); begin
  {Não faz nada no caso de matrizes de adjacência}
end;   {LiberaGrafo}
```

Matriz de adjacência: Implementação

```
procedure ImprimeGrafo (var Grafo : TipoGrafo);
var i, j: integer;
begin
  write('   ');
  for i := 0 to Grafo.NumVertices-1 do write(i:3); writeln;
  for i := 0 to Grafo.NumVertices-1 do
  begin
    write(i:3);
    for j := 0 to Grafo.NumVertices-1 do write(Grafo.mat[i,j]:3);
    writeln;
  end;
end;   {ImprimeGrafo}
```