



Teoria dos Grafos e Computabilidade

— Preliminary concepts —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Motivation —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Porque estudar grafos?

- ▶ Arcabouço matemático com aplicação em diversas áreas do conhecimento
- ▶ Utilizados na definição e/ou resolução de problemas
- ▶ Estudar grafos é mais uma forma de solucionar problemas computáveis
- ▶ Os estudos teóricos em grafos buscam o desenvolvimento de algoritmos mais eficientes.
- ▶ Abstração matemática que representa situações reais através de um diagrama.

Porque estudar grafos?

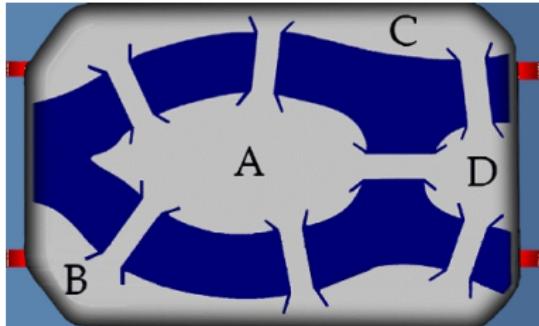
- ▶ Arcabouço matemático com aplicação em diversas áreas do conhecimento
- ▶ Utilizados na definição e/ou resolução de problemas
- ▶ Estudar grafos é mais uma forma de solucionar problemas computáveis
- ▶ Os estudos teóricos em grafos buscam o desenvolvimento de algoritmos mais eficientes.
- ▶ Abstração matemática que representa situações reais através de um diagrama.

Áreas de conhecimento

Genética, química, pesquisa operacional, telecomunicações, engenharia elétrica, redes de computadores, conexão de vôos aéreos, restrições de precedência, fluxo de programas, dentre outros

Motivação – Pontes de Königsberg

Pontes de Königsberg O rio Pregel divide o centro da cidade de Königsberg (Prússia no século XVII, atual Kaliningrado, Rússia) em quatro regiões. Essas regiões são ligadas por um complexo de sete (7) pontes, conforme mostra a figura. Discutia-se nas ruas da cidade a possibilidade de atravessar todas as pontes, voltando ao lugar de onde se saiu, sem repetir alguma. Havia-se tornado uma lenda popular a possibilidade da façanha quando Euler , em 1736, provou que não existia caminho que possibilitasse tais restrições.

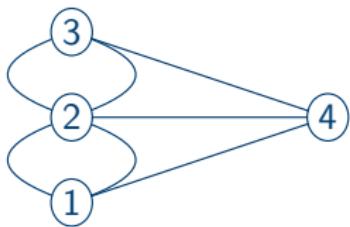


Motivação – Pontes de Königsberg

- ▶ Resolvido em 1736 por Leonhard Euler
- ▶ Necessário um modelo para representar o problema
- ▶ Abstração de detalhes irrelevantes:
 - ▶ Área de cada ilha
 - ▶ Formato de cada ilha
 - ▶ Tipo da ponte, etc.

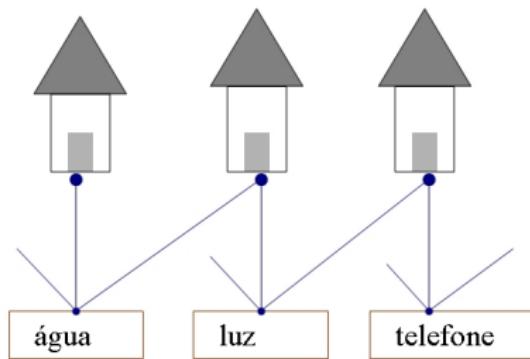
Motivação – Pontes de Königsberg

- Resolvido em 1736 por Leonhard Euler
- Necessário um modelo para representar o problema
- Abstração de detalhes irrelevantes:
 - Área de cada ilha
 - Formato de cada ilha
 - Tipo da ponte, etc.
- Euler generalizou o problema por meio de um modelo de grafos



Problemas das 3 casas

É possível conectar os 3 serviços às 3 casas sem haver cruzamento de tubulação?



Colorir um mapa

Quantas cores são necessárias para colorir o mapa do Brasil, sendo que estados adjacentes não podem ter a mesma cor?



Caminho mínimo

De forma a reduzir seus custos operacionais, uma empresa de transporte de cargas deseja oferecer aos motoristas de sua frota um mecanismo que os auxilie a selecionar o melhor caminho (o de menor distância) entre quaisquer duas cidades por ela servidas, de forma a que sejam minimizados os custos de transporte.



Questions?

Preliminary concepts
– Motivation –

Teoria dos Grafos e Computabilidade

— Graph definition —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Grafo coleção de vértices e arestas

Vértices objeto simples que pode ter nomes e outros atributos

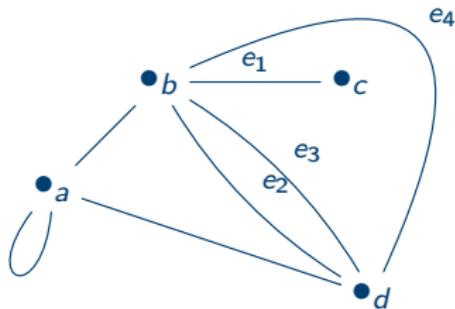
Arestas conexão entre dois vértices

Conceitos

Grafo coleção de vértices e arestas

Vértices objeto simples que pode ter nomes e outros atributos

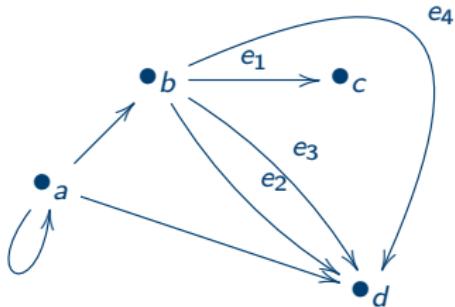
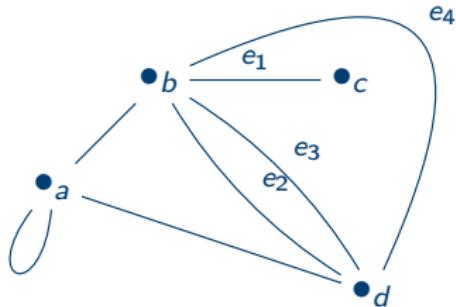
Arestas conexão entre dois vértices



Grafo coleção de vértices e arestas

Vértices objeto simples que pode ter nomes e outros atributos

Arestas conexão entre dois vértices



Grafo coleção de vértices e arestas

Vértices objeto simples que pode ter nomes e outros atributos

Arestas conexão entre dois vértices

Um grafo $G = (V, E)$ em que V é o conjunto de vértices e E o conjunto de arestas de forma que

Grafo coleção de vértices e arestas

Vértices objeto simples que pode ter nomes e outros atributos

Arestas conexão entre dois vértices

Um grafo $G = (V, E)$ em que V é o conjunto de vértices e E o conjunto de arestas de forma que

- $E = \{(u, v) \mid u, v \in V\}$ – grafo direcionado

Grafo coleção de vértices e arestas

Vértices objeto simples que pode ter nomes e outros atributos

Arestas conexão entre dois vértices

Um grafo $G = (V, E)$ em que V é o conjunto de vértices e E o conjunto de arestas de forma que

- $E = \{(u, v) \mid u, v \in V\}$ – grafo direcionado
- $E = \{\{u, v\} \mid u, v \in V\}$ – grafo não-direcionado

Modelagem de grafo

- ▶ No problema das casas
 - ▶ Vértices são casas e serviços
 - ▶ Areias são as tubulações entre casas e serviços
- ▶ No problema da coloração de mapas
 - ▶ Vértices são estados
 - ▶ Areias relacionam estados vizinhos
- ▶ No problema do caminho mais curto
 - ▶ Vértices são as cidades
 - ▶ Areias são as ligações entre as cidades

Problemas interessantes

Problema das 4 cores

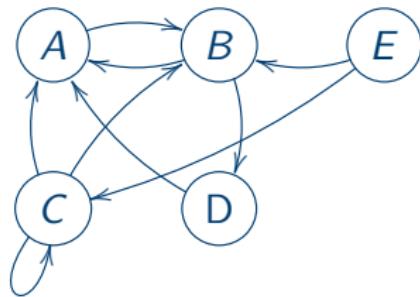
Qual a quantidade mínima de cores para colorir um mapa de tal forma que países fronteiriços possuam cores diferentes? Apresenta-se um exemplo em que 3 cores não são suficientes. Uma prova de que 5 cores é suficiente foi formulada. Conjeturou-se então que 4 cores seriam suficientes. Esta questão ficou em aberto até 1976 quando Appel e Haken provaram para 4 cores

Problema do ciclo Hamiltoniano (Hamilton 1859)

Existem n cidades. Cada par de cidades pode ser adjacente ou não arbitrariamente. Partindo de uma cidade qualquer, o problema consiste em determinar um trajeto que passe exatamente uma vez em cada cidade e retorne ao ponto de partida.

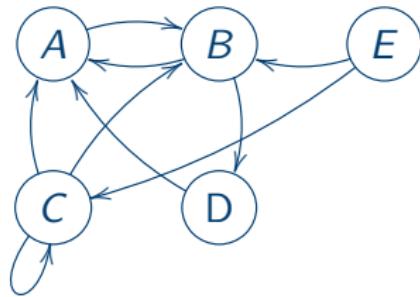
Conceitos

Um **grafo direcionado** é um par $G=(V,E)$, em que V é um conjunto finito e E é uma relação binária em V .

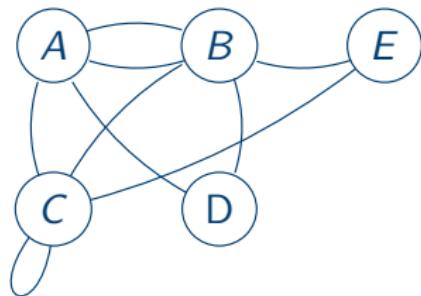


Conceitos

Um **grafo direcionado** é um par $G=(V,E)$, em que V é um conjunto finito e E é uma relação binária em V .



Um **grafo não direcionado** é um par $G=(V,E)$ em que o conjunto de arestas E consiste em pares de vértices não orientados. As arestas (v_i, v_j) e (v_j, v_i) são consideradas a mesma aresta.



Questions?

Preliminary concepts
– Graph definition –



Teoria dos Grafos e Computabilidade

— Terminology —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Terminologia

Loop

uma aresta associada ao par de vértices (v_i, v_i)



Terminologia

Loop

uma aresta associada ao par de vértices (v_i, v_i)



Arestas paralelas

quando mais de uma aresta está associada ao mesmo par de vértices



Terminologia

Loop

uma aresta associada ao par de vértices (v_i, v_i)



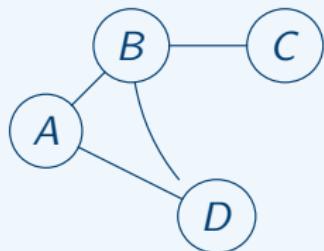
Arestas paralelas

quando mais de uma aresta está associada ao mesmo par de vértices



Grafo simples

um grafo que não possui loops e nem arestas paralelas



Terminologia

Loop

uma aresta associada ao par de vértices (v_i, v_i)



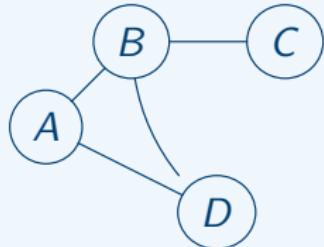
Arestas paralelas

quando mais de uma aresta está associada ao mesmo par de vértices



Grafo simples

um grafo que não possui loops e nem arestas paralelas



Vértices adjacentes

Dois vértices são ditos adjacentes se eles são pontos finais de uma mesma aresta

Grafo não direcionado

- ▶ grau $d(v)$ - número de arestas que incidem em v .

Grafo direcionado

- ▶ grau de entrada $d^-(v)$ - número de arestas que chegam em v
- ▶ grau de saída $d^+(v)$ - número de arestas que saem em v

Terminologia

Grafo não direcionado

- ▶ grau $d(v)$ - número de arestas que incidem em v .

Grafo direcionado

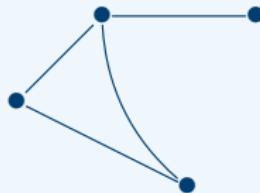
- ▶ grau de entrada $d^-(v)$ - número de arestas que chegam em v
- ▶ grau de saída $d^+(v)$ - número de arestas que saem em v

Um laço conta duas vezes para o grau de um vértice

Terminologia

Grafo não direcionado

- ▶ grau $d(v)$ - número de arestas que incidem em v .



Grafo direcionado

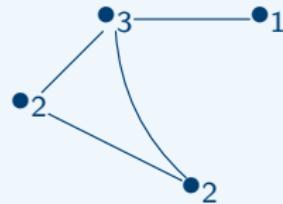
- ▶ grau de entrada $d^-(v)$ - número de arestas que chegam em v
- ▶ grau de saída $d^+(v)$ - número de arestas que saem em v

Um laço conta duas vezes para o grau de um vértice

Terminologia

Grafo não direcionado

- ▶ grau $d(v)$ - número de arestas que incidem em v .



Grafo direcionado

- ▶ grau de entrada $d^-(v)$ - número de arestas que chegam em v
- ▶ grau de saída $d^+(v)$ - número de arestas que saem em v

Um laço conta duas vezes para o grau de um vértice

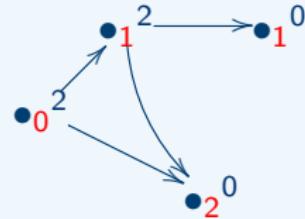
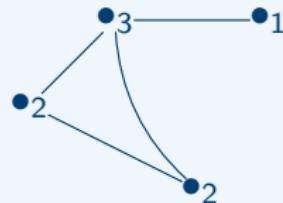
Terminologia

Grafo não direcionado

- ▶ grau $d(v)$ - número de arestas que incidem em v .

Grafo direcionado

- ▶ grau de entrada $d^-(v)$ - número de arestas que chegam em v
- ▶ grau de saída $d^+(v)$ - número de arestas que saem em v



Um laço conta duas vezes para o grau de um vértice

Terminologia

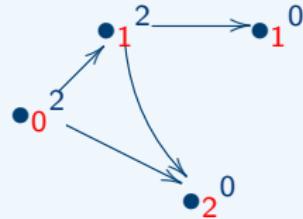
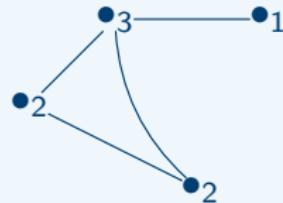
Grafo não direcionado

- ▶ grau $d(v)$ - número de arestas que incidem em v .

Grafo direcionado

- ▶ grau de entrada $d^-(v)$ - número de arestas que chegam em v
- ▶ grau de saída $d^+(v)$ - número de arestas que saem em v

Um laço conta duas vezes para o grau de um vértice

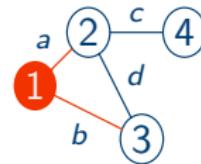
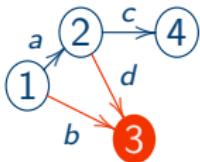


Seqüência de graus

Escrever o grau de todos os vértices em ordem não-decrescente

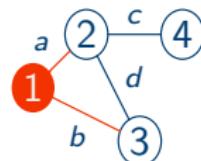
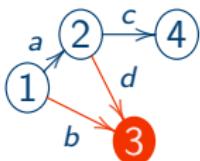
Terminologia

- Duas arestas não paralelas são **adjacentes** se elas são incidentes a um vértice comum

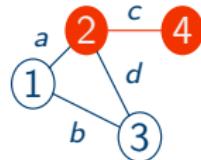
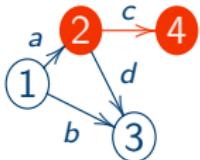


Terminologia

- Duas arestas não paralelas são **adjacentes** se elas são incidentes a um vértice comum

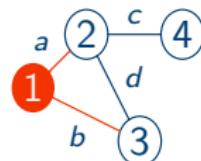
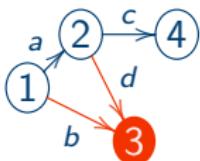


- Quando um vértice v é o vértice final de alguma aresta $e = uv$, é dito que e é **incidente** em v a partir de u

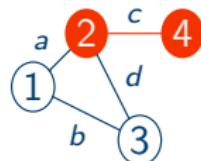
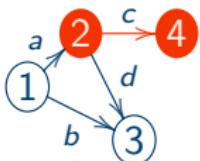


Terminologia

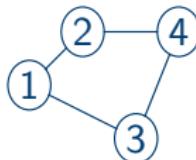
- Duas arestas não paralelas são **adjacentes** se elas são incidentes a um vértice comum



- Quando um vértice v é o vértice final de alguma aresta $e = uv$, é dito que e é **incidente** em v a partir de u

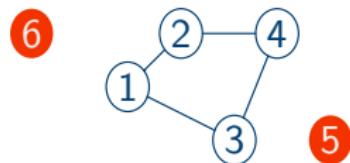


- Um grafo no qual todos os vértices possuem o mesmo grau é chamado de **grafo regular**.



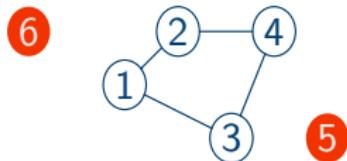
Terminologia

- ▶ Um vértice com nenhuma aresta incidente é chamado de **vértice isolado**.

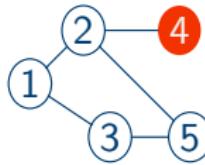


Terminologia

- Um vértice com nenhuma aresta incidente é chamado de **vértice isolado**.

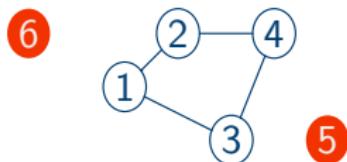


- Um vértice com grau 1 é chamado de **vértice pendente**

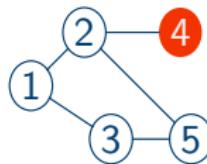


Terminologia

- Um vértice com nenhuma aresta incidente é chamado de **vértice isolado**.



- Um vértice com grau 1 é chamado de **vértice pendente**

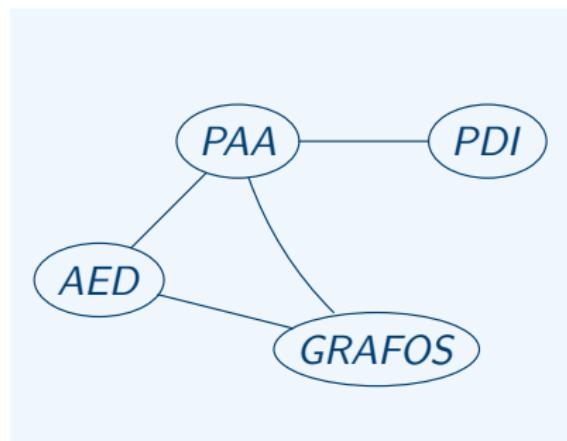


- Um grafo sem nenhuma aresta é chamado de **grafo nulo**. Todos os vértices em um grafo nulo são vértices isolados



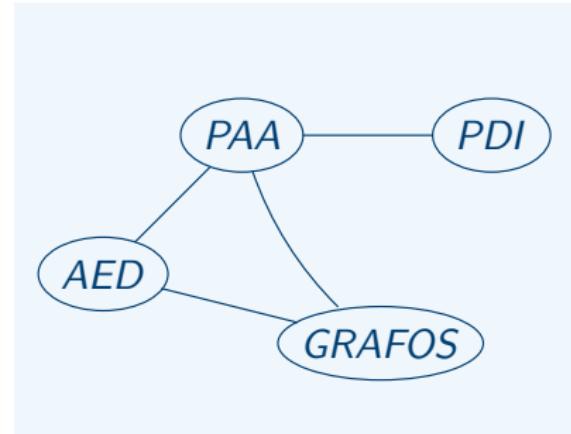
Grafos valorado e rotulado

Um grafo $G=(V,A)$ é dito ser rotulado em vértices (ou arestas) quando a cada vértice (ou aresta) estiver associado um rótulo

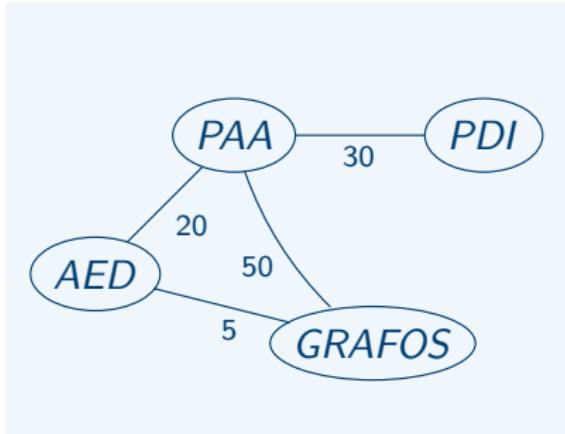


Grafos valorado e rotulado

Um grafo $G=(V,A)$ é dito ser rotulado em vértices (ou arestas) quando a cada vértice (ou aresta) estiver associado um rótulo



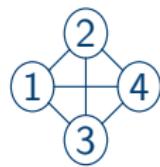
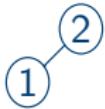
Um grafo $G=(V,A)$ é dito ser valorado quando existe uma ou mais funções relacionando V e/ou A com um conjunto de números.



Terminologia

Um grafo $G=(V,E)$ é **completo** se para cada par de vértices v_i e v_j existe uma aresta entre v_i e v_j . Em um grafo completo quaisquer dois vértices distintos são adjacentes (K_n)

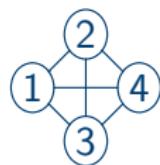
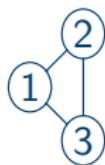
①



Terminologia

Um grafo $G=(V,E)$ é **completo** se para cada par de vértices v_i e v_j existe uma aresta entre v_i e v_j . Em um grafo completo quaisquer dois vértices distintos são adjacentes (K_n)

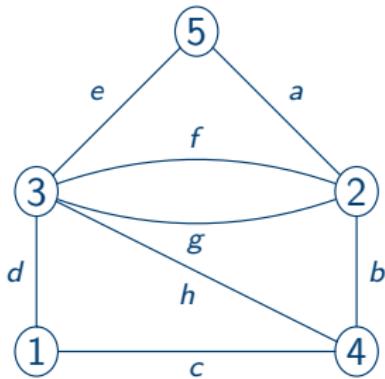
①



Seja K_n um grafo completo com n vértices. O **número de arestas** de um grafo completo é :

$$|E| = \frac{(n - 1) \times n}{2}$$

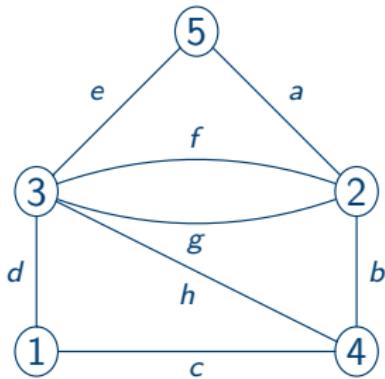
Terminology



Walk A walk in $G = (V, E)$ is a finite non-null sequence

$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

Terminology



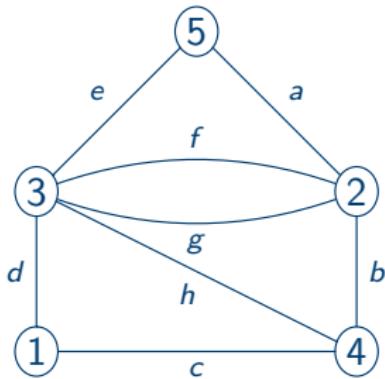
walk

5a2f3f2g3h4b2

Walk A walk in $G = (V, E)$ is a finite non-null sequence

$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

Terminology



walk

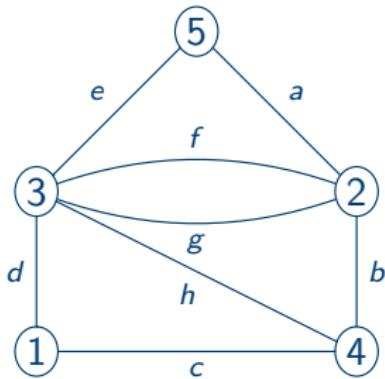
5a2f3f2g3h4b2

Walk A walk in $G = (V, E)$ is a finite non-null sequence

$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

Trail A trail is a walk in $G = (V, E)$ if the edges of W are distinct.

Terminology



walk
trail

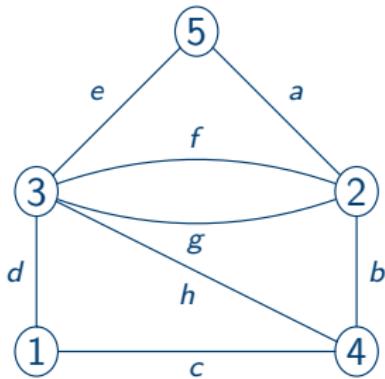
5a2f3f2g3h4b2
4c1d3h4b2g3

Walk A walk in $G = (V, E)$ is a finite non-null sequence

$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

Trail A trail is a walk in $G = (V, E)$ if the edges of W are distinct.

Terminology



walk
trail

5a2f3f2g3h4b2
4c1d3h4b2g3

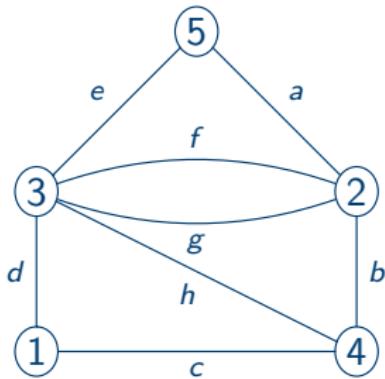
Walk A walk in $G = (V, E)$ is a finite non-null sequence

$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

Trail A trail is a walk in $G = (V, E)$ if the edges of W are distinct.

Path A path is a trail in $G = (V, E)$ if the vertices of W are distinct.

Terminology



walk
trail
path

5a2f3f2g3h4b2
4c1d3h4b2g3
1c4h3e5a2

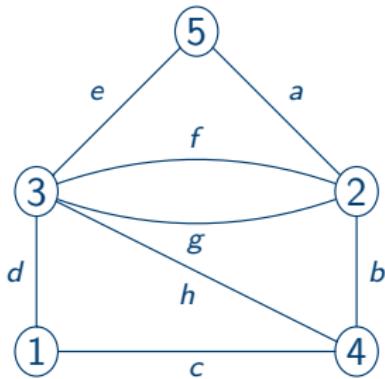
Walk A walk in $G = (V, E)$ is a finite non-null sequence

$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

Trail A trail is a walk in $G = (V, E)$ if the edges of W are distinct.

Path A path is a trail in $G = (V, E)$ if the vertices of W are distinct.

Terminology



walk
trail
path

5a2f3f2g3h4b2
4c1d3h4b2g3
1c4h3e5a2

Walk A walk in $G = (V, E)$ is a finite non-null sequence

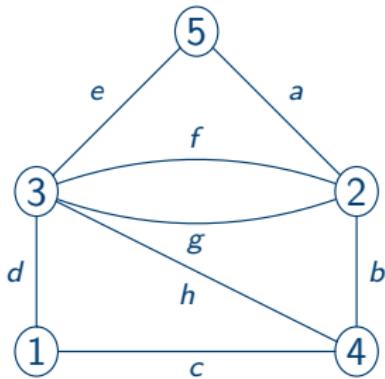
$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

Trail A trail is a walk in $G = (V, E)$ if the edges of W are distinct.

Path A path is a trail in $G = (V, E)$ if the vertices of W are distinct.

Cycle A cycle is a closed path – origin and terminus are the same.

Terminology



walk	5a2f3f2g3h4b2
trail	4c1d3h4b2g3
path	1c4h3e5a2
cycle	1c4b2a5e3d1

Walk A walk in $G = (V, E)$ is a finite non-null sequence

$W = v_0 e_1 v_1 e_2 \dots e_k v_k$ terms are alternately vertices and edges, such that, for $1 \leq i \leq k$, the ends of e_i are v_{i-1} and v_i . We say that W is a walk from v_0 (origin) to v_k (terminus).

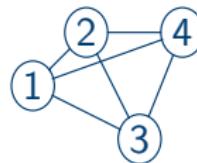
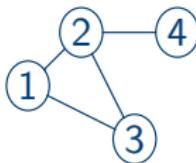
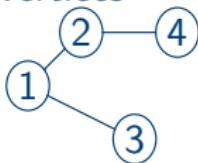
Trail A trail is a walk in $G = (V, E)$ if the edges of W are distinct.

Path A path is a trail in $G = (V, E)$ if the vertices of W are distinct.

Cycle A cycle is a closed path – origin and terminus are the same.

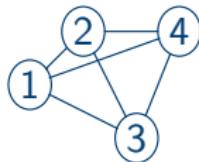
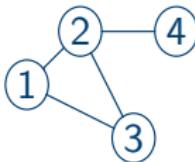
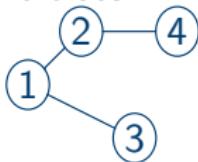
Terminologia

Grafo conexo – existe pelo menos um caminho entre todos os pares de vértices

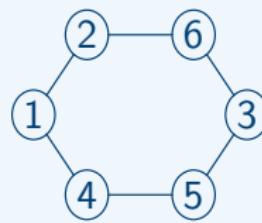
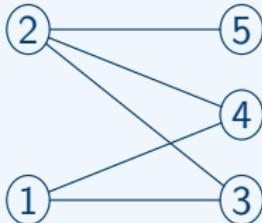


Terminologia

Grafo conexo – existe pelo menos um caminho entre todos os pares de vértices

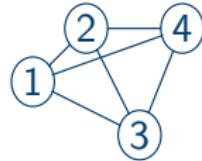
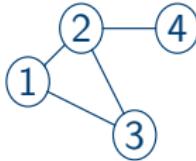
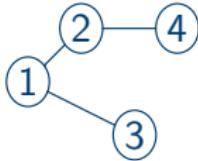


Um grafo é dito ser **bipartido** quando seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1 e V_2 , tais que toda aresta de G une um vértice de V_1 a outro de V_2 .

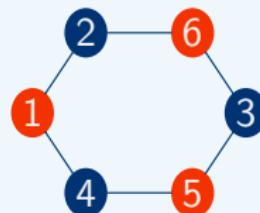
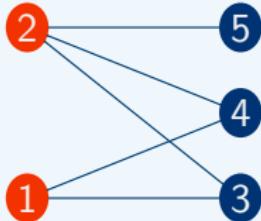


Terminologia

Grafo conexo – existe pelo menos um caminho entre todos os pares de vértices

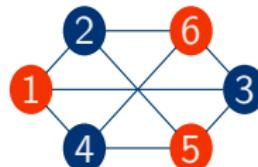
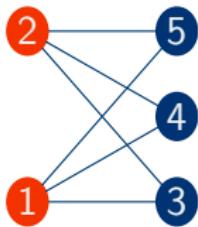


Um grafo é dito ser **bipartido** quando seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1 e V_2 , tais que toda aresta de G une um vértice de V_1 a outro de V_2 .



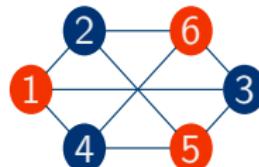
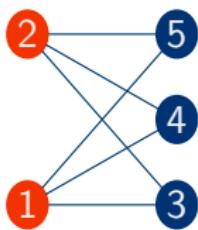
Grafo bipartido completo

Um grafo é dito ser **bipartido completo** quando seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1 e V_2 , tais que toda aresta de G une um vértice de V_1 a outro de V_2 , e que todo vértice de V_1 é **adjacente** a todo vértice de V_2 .



Grafo bipartido completo

Um grafo é dito ser **bipartido completo** quando seu conjunto de vértices V puder ser particionado em dois subconjuntos V_1 e V_2 , tais que toda aresta de G une um vértice de V_1 a outro de V_2 , e que todo vértice de V_1 é **adjacente** a todo vértice de V_2 .



Seja K_{mn} um grafo bipartido completo com n vértices em V_1 e m vértices em V_2 . O **número de arestas** de um grafo bipartido completo é:

$$|E| = n \times m$$

Propriedade de grau

O número de arestas incidentes a um vértice v_i é chamado de grau, $d(v_i)$, do vértice i . A soma dos graus de todos os vértices de um grafo G é duas vezes o número de arestas de G , e portanto é par.

$$\sum_{i=1}^n d(v_i) = 2e$$

Propriedade de grau

O número de arestas incidentes a um vértice v_i é chamado de grau, $d(v_i)$, do vértice i . A soma dos graus de todos os vértices de um grafo G é duas vezes o número de arestas de G , e portanto é par.

$$\sum_{i=1}^n d(v_i) = 2e$$

Teorema O número de vértices de grau ímpar em um grafo é par

$$\sum_{i=1}^n d(v_i) = \sum_{d(v_j) \text{par}} d(v_j) + \sum_{d(v_k) \text{ímpar}} d(v_k)$$

Operações sobre grafos

Seja $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ dois grafos. O grafo

$$G = G_1 \cup G_2$$

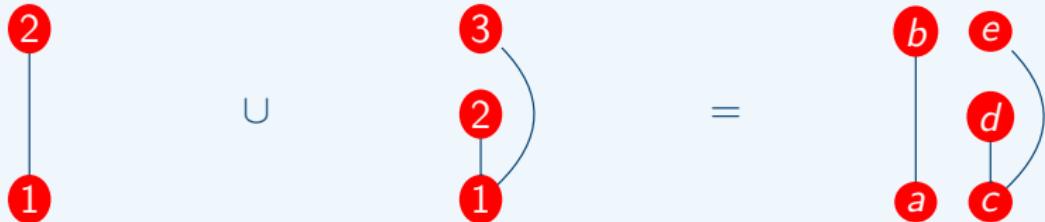
que representa a **união de dois grafos**, é formado pelo grafo com conjunto de vértices $V_1 \cup V_2$ e conjunto de arestas $E_1 \cup E_2$.

Operações sobre grafos

Seja $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ dois grafos. O grafo

$$G = G_1 \cup G_2$$

que representa a **união de dois grafos**, é formado pelo grafo com conjunto de vértices $V_1 \cup V_2$ e conjunto de arestas $E_1 \cup E_2$.



Operações sobre grafos

Seja $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ dois grafos. O grafo

$$G = G_1 + G_2$$

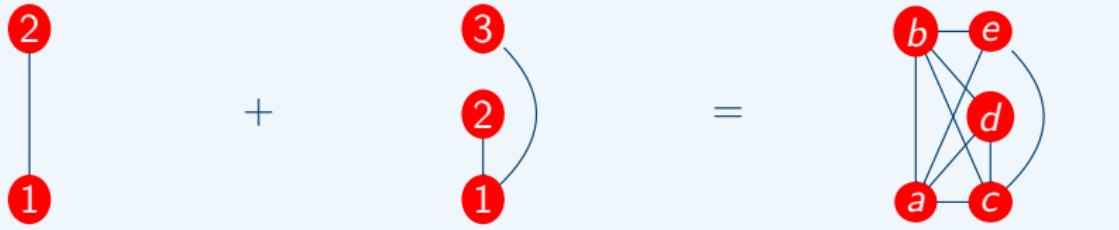
que representa a soma de dois grafos, é formado por $G_1 \cup G_2$ e de arestas ligando cada vértice de V_1 a V_2

Operações sobre grafos

Seja $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$ dois grafos. O grafo

$$G = G_1 + G_2$$

que representa a soma de dois grafos, é formado por $G_1 \cup G_2$ e de arestas ligando cada vértice de V_1 a V_2



Propriedades de soma e união

- ▶ Podem ser aplicadas a qualquer número finito de grafos
- ▶ São operações associativas
- ▶ São operações comutativas

Propriedades de soma e união

- ▶ Podem ser aplicadas a qualquer número finito de grafos
- ▶ São operações associativas
- ▶ São operações comutativas

Exemplo 1

Defina soma e união para grafos direcionados . As propriedades de associação e comutação são mantidas?

Remoção de aresta e de vértice

Se e é uma aresta de um grafo G , denota-se $G - e$ o grafo obtido de G pela remoção da aresta e . Se E é um conjunto de arestas em G , denota-se $G-E$ ao grafo obtido pela remoção das arestas em E .

Remoção de aresta e de vértice

Se e é uma aresta de um grafo G , denota-se $G - e$ o grafo obtido de G pela remoção da aresta e . Se E é um conjunto de arestas em G , denota-se $G-E$ ao grafo obtido pela remoção das arestas em E .

Se v é um vértice de um grafo G denota-se por $G - v$ o grafo obtido de G pela remoção do vértice v conjuntamente com as arestas incidentes a v . Denota-se $G - S$ ao grafo obtido pela remoção dos vértices em S , sendo S um conjunto qualquer de vértices de G .

Remoção de aresta e de vértice

Se e é uma aresta de um grafo G , denota-se $G - e$ o grafo obtido de G pela remoção da aresta e . Se E é um conjunto de arestas em G , denota-se $G-E$ ao grafo obtido pela remoção das arestas em E .

Se v é um vértice de um grafo G denota-se por $G - v$ o grafo obtido de G pela remoção do vértice v conjuntamente com as arestas incidentes a v . Denota-se $G - S$ ao grafo obtido pela remoção dos vértices em S , sendo S um conjunto qualquer de vértices de G .

Denota-se por G/e o grafo obtido pela contração da aresta e . Remova $e = (v, w)$ de G e una suas extremidades v e w de tal forma que o vértice resultante seja incidente às arestas originalmente incidentes a v e w .

Questions?

Preliminary concepts
– Terminology –

Teoria dos Grafos e Computabilidade

— Conceitos —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Introdução —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Objetivos

1

Capacitar o aluno a utilizar as linguagens **proposicional** e **de predicados** para expressar conhecimento.

Objetivos

1

Capacitar o aluno a utilizar as linguagens **proposicional** e de **predicados** para expressar conhecimento.

2

Qualificar o aluno a aplicar princípios básicos de lógica e matemática na **Análise e Validação** de Argumentos.

Objetivos

1

Capacitar o aluno a utilizar as linguagens **proposicional** e de **predicados** para expressar conhecimento.

2

Qualificar o aluno a aplicar princípios básicos de lógica e matemática na **Análise** e **Validação** de Argumentos.

3

Habilitar o aluno a compreender o **Pensamento Indutivo** e **Dedutivo**.

Objetivos

1

Capacitar o aluno a utilizar as linguagens **proposicional** e de **predicados** para expressar conhecimento.

2

Qualificar o aluno a aplicar princípios básicos de lógica e matemática na **Análise** e **Validação** de Argumentos.

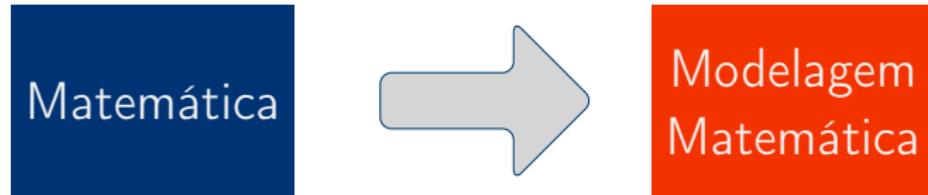
3

Habilitar o aluno a compreender o **Pensamento Indutivo** e **Dedutivo**.

4

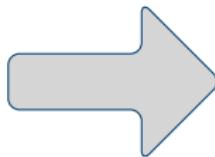
Reconhecer os mecanismos lógicos para **formalização** e **dedução** de problemas relacionados à computação.

Importância da Matemática



Importância da Matemática

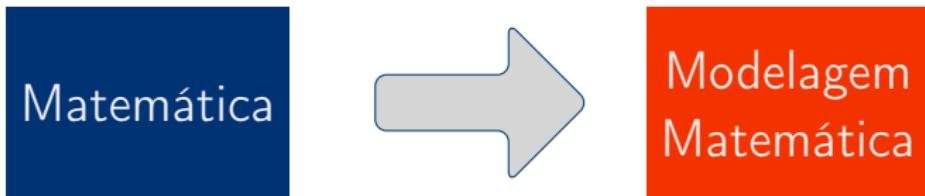
Matemática



Modelagem
Matemática

Provê métodos (estruturas) convenientes para resolver problemas.

Importância da Matemática



Provê métodos (estruturas) convenientes para resolver problemas.

MATEMÁTICA

Matemática Contínua

Matemática Discreta

Importância da Matemática

MATEMÁTICA
CONTÍNUA

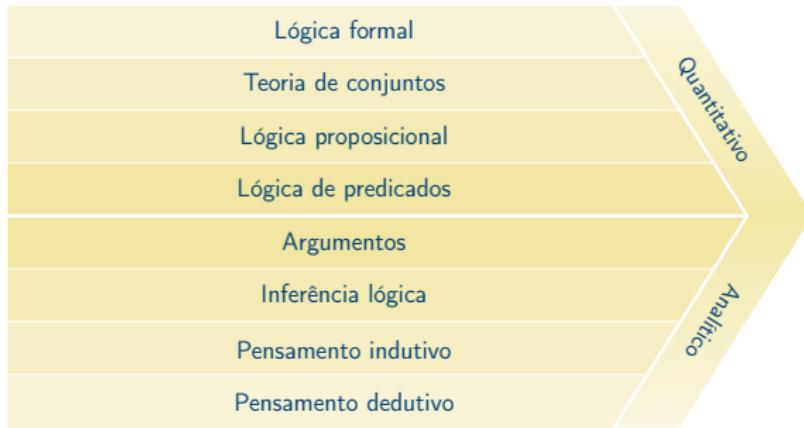
- ▶ ligada ao Cálculo Infinitesimal
- ▶ permite modelar (“prever”) fenômenos físicos
- ▶ Análise Numérica “traduz” o Cálculo para formato compatível com o computador
- ▶ Exemplos de aplicações:
 - ▶ na Computação: Análise Numérica, Computação Gráfica
 - ▶ na Engenharia: projetos detalhados de pontes, aviões, carros

MATEMÁTICA
DISCRETA

- ▶ ligada a processos “discretos” (não contínuos) realizados passo-a-passo
- ▶ só interessam os “estados” em que se pode encontrar um sistema e não os detalhes da “transição” entre estes estados
- ▶ Exemplos de aplicações na CC: grafos, estruturas de dados, máquinas de estados finitos, codificação

Importância da Matemática

Alguns elementos da Matemática Discreta relevantes para o estudo da Ciência da Computação que serão estudados em **LÓGICA COMPUTACIONAL**



Definição

Lógica vem do Grego **logos** que significa palavra, pensamento, ideia, argumento, relato, razão lógica

Definição

Lógica vem do Grego **logos** que significa palavra, pensamento, ideia, argumento, relato, razão lógica

- Ciência das **leis** do pensamento e a **arte** de aplicá-las à pesquisa e à demonstração da verdade.

Definição

Lógica vem do Grego **logos** que significa palavra, pensamento, ideia, argumento, relato, razão lógica

- ▶ Ciência das **leis** do pensamento e a **arte** de aplicá-las à pesquisa e à demonstração da verdade.
- ▶ Linguagem de **organização** do conhecimento

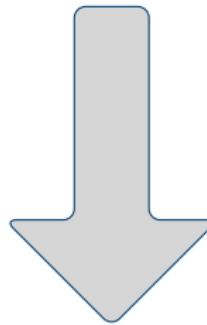
Definição

Lógica vem do Grego **logos** que significa palavra, pensamento, ideia, argumento, relato, razão lógica

- ▶ Ciência das **leis** do pensamento e a **arte** de aplicá-las à pesquisa e à demonstração da verdade.
- ▶ Linguagem de **organização** do conhecimento
- ▶ Linguagem de **representação** do raciocínio

Aristóteles é considerado o
PAI da lógica formal

Aristóteles é considerado o
PAI da lógica formal



LÓGICA é a base para a construção de algoritmos

Paradoxos

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

Paradoxo do mentiroso

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

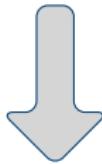
Um homem diz que está mentindo. O que ele diz é verdade ou mentira?

Paradoxo do mentiroso

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

Paradoxo do mentiroso

Um homem diz que está mentindo. O que ele diz é verdade ou mentira?



Considerando a afirmação todo o homem é mentiroso não pode ser verdadeira, porque, a sê-lo, o homem seria mentiroso e, por consequência falso tudo o que dissesse. Por outro lado, também não pode ser falsa, pois isso implicaria que os homens falam a verdade, sendo, portanto, verdadeira a afirmação do mentiroso.

Paradoxos

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

Paradoxos

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

Paradoxo do barbeiro

Paradoxos

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

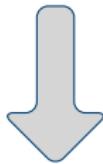
Faço a barba a todos os homens da cidade que não se barbeiam sozinhos, e só a esses.

Paradoxo do barbeiro

Paradoxos são declarações aparentemente verdadeiras que, apesar disso, contêm uma contradição lógica.

Paradoxo do barbeiro

Faço a barba a todos os homens da cidade que não se barbeiam sozinhos, e só a esses.



Se ele próprio se barbear, pertencerá ao grupo dos que se barbeiam sozinhos, como é óbvio. Mas a afirmação diz expressamente que ele nunca faz a barba de ninguém pertencente a esse conjunto. Portanto, não pode barbear-se a si próprio. Se é outra pessoa a fazer a barba ao barbeiro, então ele é um homem que não se barbeia sozinho.

Questions?

Conceitos
– Introdução –

Teoria dos Grafos e Computabilidade

— O que é lógica e para que serve? —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

O que é lógica e para que serve?

A LÓGICA é uma técnica eficiente para diversos domínios

- ▶ Organização de conhecimentos em qualquer área;

O que é lógica e para que serve?

A LÓGICA é uma técnica eficiente para diversos domínios

- ▶ Organização de conhecimentos em qualquer área;
- ▶ Inferência correta sem esforço consciente;

O que é lógica e para que serve?

A LÓGICA é uma técnica eficiente para diversos domínios

- ▶ Organização de conhecimentos em qualquer área;
- ▶ Inferência correta sem esforço consciente;
- ▶ Interpretação e análise rápida de informações;

O que é lógica e para que serve?

A LÓGICA é uma técnica eficiente para diversos domínios

- ▶ Organização de conhecimentos em qualquer área;
- ▶ Inferência correta sem esforço consciente;
- ▶ Interpretação e análise rápida de informações;
- ▶ Expressividade competência linguística (oral e escrita);

O que é lógica e para que serve?

A LÓGICA é uma técnica eficiente para diversos domínios

- ▶ Organização de conhecimentos em qualquer área;
- ▶ Inferência correta sem esforço consciente;
- ▶ Interpretação e análise rápida de informações;
- ▶ Expressividade competência linguística (oral e escrita);
- ▶ Detecção de padrões em estruturas (premissas, pressuposições, cenários, etc.)

O que é lógica e para que serve?

O brinco da Princesa

O que é lógica e para que serve?

O brinco da Princesa

Há não muito tempo atrás, num país distante, havia um velho rei que tinha três filhas, intelligentíssimas e de indescritível beleza, chamadas Guilhermina, Genoveva e Griselda. Sentindo-se perto de partir dessa para a melhor, e sem saber qual das filhas designar como sua sucessora, o velho rei resolveu submetê-las a um teste. A vencedora não apenas seria a nova soberana, como ainda receberia a senha da conta secreta do rei (num banco suíço), além de um fim de semana na Disneylândia. Chamando as filhas à sua presença, o rei mostrou-lhe cinco pares de brincos, idênticos em tudo com exceção das pedras neles engastadas: três eram de esmeralda, dois de rubi. O rei vendou então os olhos das moças e, escolhendo, ao acaso, colocou em cada uma elas um par de brincos.

Exemplo retirado de Mortari, C. A., Introdução à lógica, 6^a ed, São Paulo, 2001

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

Guilhermina foi a primeira a tentar

A venda dos olhos dos seus olhos foi removida. Guilhermina examinou os brincos de suas irmãs, mas não foi capaz de dizer que tipo de pedra estava nos seus (e retirou-se furiosa).

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

Guilhermina foi a primeira a tentar

A venda dos olhos dos seus olhos foi removida. Guilhermina examinou os brincos de suas irmãs, mas não foi capaz de dizer que tipo de pedra estava nos seus (e retirou-se furiosa).

Genoveva foi a segunda a tentar

Após examinar os brincos de Griselda, Genoveva se deu por conta de que também não sabia determinar se seus brincos eram de esmeralda ou de rubi e, saiu da mesma forma furiosa que sua irmã.

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

Griselda foi a última a tentar

Antes mesmo que o rei tirasse-lhe a venda dos olhos, anunciou corretamente, em alto e bom som, o tipo de pedra de seus brincos dizendo ainda o porquê de sua afirmação.

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

Griselda foi a última a tentar

Antes mesmo que o rei tirasse-lhe a venda dos olhos, anunciou corretamente, em alto e bom som, o tipo de pedra de seus brincos dizendo ainda o porquê de sua afirmação.



Como a Griselda descobriu?

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

OPÇÃO	GUILHERMINA	GENOVEVA	GRISELDA
C1	esmeralda	esmeralda	esmeralda
C2	esmeralda	esmeralda	rubi
C3	esmeralda	rubi	rubi
C4	esmeralda	rubi	esmeralda
C5	rubi	rubi	esmeralda
C6	rubi	esmeralda	esmeralda
C7	rubi	esmeralda	rubi

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

OPÇÃO	GUILHERMINA	GENOVEVA	GRISELDA
C1	esmeralda	esmeralda	esmeralda
C2	esmeralda	esmeralda	rubi
C3	esmeralda	rubi	rubi
C4	esmeralda	rubi	esmeralda
C5	rubi	rubi	esmeralda
C6	rubi	esmeralda	esmeralda
C7	rubi	esmeralda	rubi

Única chance de Guilhermina ser a vencedora está em C3, pois as duas irmãs estão com rubi, mas ela errou

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

OPÇÃO	GUILHERMINA	GENOVEVA	GRISELDA
C1	esmeralda	esmeralda	esmeralda
C2	esmeralda	esmeralda	rubi
C3	esmeralda	rubi	rubi
C4	esmeralda	rubi	esmeralda
C5	rubi	rubi	esmeralda
C6	rubi	esmeralda	esmeralda
C7	rubi	esmeralda	rubi

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

OPÇÃO	GUILHERMINA	GENOVEVA	GRISELDA
C1	esmeralda	esmeralda	esmeralda
C2	esmeralda	esmeralda	rubi
C3	esmeralda	rubi	rubi
C4	esmeralda	rubi	esmeralda
C5	rubi	rubi	esmeralda
C6	rubi	esmeralda	esmeralda
C7	rubi	esmeralda	rubi

C2 e C7 estão expostas as chances de Genoveva saber qual era a pedra de seu brinco, pois era necessário que Griselda estivesse usando um Rubi.

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

OPÇÃO	GUILHERMINA	GENOVEVA	GRISELDA
C1	esmeralda	esmeralda	esmeralda
C2	esmeralda	esmeralda	rubi
C3	esmeralda	rubi	rubi
C4	esmeralda	rubi	esmeralda
C5	rubi	rubi	esmeralda
C6	rubi	esmeralda	esmeralda
C7	rubi	esmeralda	rubi

O que é lógica e para que serve?

O teste consistia no seguinte: aquela que pudesse dizer, sem sombra de dúvida, qual tipo de pedra que havia em seus brincos herdaria o reino (e a conta na Suíça etc).

OPÇÃO	GUILHERMINA	GENOVEVA	GRISELDA
C1	esmeralda	esmeralda	esmeralda
C2	esmeralda	esmeralda	rubi
C3	esmeralda	rubi	rubi
C4	esmeralda	rubi	esmeralda
C5	rubi	rubi	esmeralda
C6	rubi	esmeralda	esmeralda
C7	rubi	esmeralda	rubi



Questions?

Conceitos

- O que é lógica e para que serve? –

Teoria dos Grafos e Computabilidade

— Motivação —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

A LÓGICA pode ser usada em diferentes áreas da Ciência da Computação

- ▶ inteligência artificial

A LÓGICA pode ser usada em diferentes áreas da Ciência da Computação

- ▶ inteligência artificial
- ▶ projeto de circuito lógico

A LÓGICA pode ser usada em diferentes áreas da Ciência da Computação

- ▶ inteligência artificial
- ▶ projeto de circuito lógico
- ▶ engenharia de software

A LÓGICA pode ser usada em diferentes áreas da Ciência da Computação

- ▶ inteligência artificial
- ▶ projeto de circuito lógico
- ▶ engenharia de software
- ▶ sistemas de informação

A LÓGICA pode ser usada em diferentes áreas da Ciência da Computação

- ▶ inteligência artificial
- ▶ projeto de circuito lógico
- ▶ engenharia de software
- ▶ sistemas de informação
- ▶ teoria de autômatos e computabilidade;

A LÓGICA pode ser usada em diferentes áreas da Ciência da Computação

- ▶ inteligência artificial
- ▶ projeto de circuito lógico
- ▶ engenharia de software
- ▶ sistemas de informação
- ▶ teoria de autômatos e computabilidade;
- ▶ processamento de linguagens;

A LÓGICA pode ser usada em diferentes áreas da Ciência da Computação

- ▶ inteligência artificial
- ▶ projeto de circuito lógico
- ▶ engenharia de software
- ▶ sistemas de informação
- ▶ teoria de autômatos e computabilidade;
- ▶ processamento de linguagens;
- ▶ teoria de sistemas distribuídos.

Motivação

Como usar de modelos e métodos matemáticos para analisar problemas em Ciência da Computação?

Provas possuem um papel fundamental na CC pois elas são usadas para certificar que software e hardware sempre irão se comportar corretamente.

Motivação

Como usar de modelos e métodos matemáticos para analisar problemas em Ciência da Computação?

Provas possuem um papel fundamental na CC pois elas são usadas para certificar que software e hardware sempre irão se comportar corretamente.

Uma PROVA é um método para estabelecer a verdade

Diferentes verdades



No sistema judicial, esta verdade é decidida por um juri baseado em evidências permitidas que foram coletadas.

Diferentes verdades

Verdade
legal

No sistema
judicial , esta
verdade é
decidida por um
juri baseado em
evidências
permitidas que
foram coletadas.

Verdade
autoritária

Nos negócios ,
esta verdade é
especificada
pour uma
pessoa ou
organização
confiável (ou
até mesmo por
um chefe).

Diferentes verdades

Verdade legal

No sistema judicial , esta verdade é decidida por um juri baseado em evidências permitidas que foram coletadas.

Verdade autoritária

Nos negócios , esta verdade é especificada pour uma pessoa ou organização confiável (ou até mesmo por um chefe).

Verdade científica

Na física ou biologia , esta verdade é confirmada por meio de experimentos.

Na ciência prefere-se teoria ao invés de verdade.

Diferentes verdades

Verdade legal

No sistema judicial , esta verdade é decidida por um juri baseado em evidências permitidas que foram coletadas.

Verdade autoritária

Nos negócios , esta verdade é especificada pour uma pessoa ou organização confiável (ou até mesmo por um chefe).

Verdade científica

Na física ou biologia , esta verdade é confirmada por meio de experimentos. Na ciência prefere-se teoria ao invés de verdade.

Verdade provável

No campo da estatística , esta verdade é estabelecida por meio de análise estatística de um conjunto de dados.

PROVA FILOSÓFICA

Envolve exposição cuidadosa e persuasiva tipicamente baseada em uma série de argumentos pequenos e plausíveis.

Diferentes tipos de provas

PROVA FILOSÓFICA

Envolve exposição cuidadosa e persuasiva tipicamente baseada em uma série de argumentos pequenos e plausíveis.

PROVA MATEMÁTICA

A prova matemática de uma proposição é uma cadeia de deduções lógicas conduzindo a uma proposição a partir de um conjunto base de premissas verdadeiras (axiomas).

Questions?

Conceitos
– Motivação –



Teoria dos Grafos e Computabilidade

— Teoria de Conjuntos e Funções —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Conjuntos —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

CONJUNTO

Coleção “bem-definida” (e não-ordenada) de objetos.

Definição de conjuntos

CONJUNTO

Coleção “bem-definida” (e não-ordenada) de objetos.

ELEMENTO

Cada objeto do conjunto é denominado de elemento ou membro.

Definição de conjuntos

CONJUNTO

Coleção “bem-definida” (e não-ordenada) de objetos.

ELEMENTO

Cada objeto do conjunto é denominado de elemento ou membro.

NOTAÇÃO

Letras maiúsculas são, em geral, usadas para denotar conjuntos, e minúsculas para denotar elementos destes conjuntos.

Definição de conjuntos

CONJUNTO

Coleção “bem-definida” (e não-ordenada) de objetos.

ELEMENTO

Cada objeto do conjunto é denominado de elemento ou membro.

NOTAÇÃO

$a \in A$ o elemento “a” pertence ao conjunto “A”

Definição de conjuntos

CONJUNTO

Coleção “bem-definida” (e não-ordenada) de objetos.

ELEMENTO

Cada objeto do conjunto é denominado de elemento ou membro.

NOTAÇÃO

$a \in A$ o elemento “a” pertence ao conjunto “A”

$a \notin A$ o elemento “a” não pertence ao conjunto “A”

Definição de conjuntos

CONJUNTO

Coleção “bem-definida” (e não-ordenada) de objetos.

ELEMENTO

Cada objeto do conjunto é denominado de elemento ou membro.

NOTAÇÃO

$$A = \{\text{melão, laranja, morango}\}$$

Definição de conjuntos

CONJUNTO

Coleção “bem-definida” (e não-ordenada) de objetos.

ELEMENTO

Cada objeto do conjunto é denominado de elemento ou membro.

NOTAÇÃO

$$A = \{\text{melão, laranja, morango}\}$$

$$\text{melão} \in A$$

$$\text{uva} \notin A$$

Exemplos de Conjuntos

Finito

Conjunto dos livros da biblioteca

Exemplos de Conjuntos

Finito *Conjunto dos livros da biblioteca*

Finito *Conjunto S de 2 elementos formados das vogais e dígitos*

vogais $X = \{a, e, i, o, u\}$

dígitos $Y = \{0, 1, 2, \dots, 9\}$

$S = \{X, Y\} = \{\{a, e, i, o, u\}, \{0, 1, 2, \dots, 9\}\}$

Exemplos de Conjuntos

Finito *Conjunto dos livros da biblioteca*

Finito *Conjunto S de 2 elementos formados das vogais e dígitos*

vogais $X = \{a, e, i, o, u\}$

dígitos $Y = \{0, 1, 2, \dots, 9\}$

$S = \{X, Y\} = \{\{a, e, i, o, u\}, \{0, 1, 2, \dots, 9\}\}$

Infinito *Conjunto dos números naturais*

Exemplos de Conjuntos

Finito *Conjunto dos livros da biblioteca*

Finito *Conjunto S de 2 elementos formados das vogais e dígitos*

vogais $X = \{a, e, i, o, u\}$

dígitos $Y = \{0, 1, 2, \dots, 9\}$

$S = \{X, Y\} = \{\{a, e, i, o, u\}, \{0, 1, 2, \dots, 9\}\}$

Infinito *Conjunto dos números naturais*

Conjunto vazio *Conjunto dos dinossauros vivos ($\{\}$ ou \emptyset)*

Formas para definir conjuntos

Chaves *Listar os elementos do conjunto entre chaves*

$\{ovo, carne, macarrão\}$ $\{1, 2, 3, \dots, 50\}$ $\{lua, \pi, 1\}$

Formas para definir conjuntos

Chaves *Listar os elementos do conjunto entre chaves*

$\{ovo, carne, macarrão\}$ $\{1, 2, 3, \dots, 50\}$ $\{lua, \pi, 1\}$

SOBRE A REPRESENTAÇÃO DOS CONJUNTOS

- A ordem em que os elementos são listados é irrelevante
- A repetição dos elementos é irrelevante

Formas para definir conjuntos

Chaves *Listar os elementos do conjunto entre chaves*

$$\{ovo, carne, macarrão\} \quad \{1, 2, 3, \dots, 50\} \quad \{lua, \pi, 1\}$$

Propriedade *Especificar uma propriedade para definir um conjunto, como $S = \{x \mid P(x)\}$*

$$\{x \in \mathbb{R} \mid -2 \leq x \leq 5\} \quad \{x \in \mathbb{N} \mid x \text{ é primo e } x \leq 47\}$$

Formas para definir conjuntos

Chaves Listar os elementos do conjunto entre chaves

$$\{ovo, carne, macarrão\} \quad \{1, 2, 3, \dots, 50\} \quad \{lua, \pi, 1\}$$

Propriedade Especificar uma propriedade para definir um conjunto, como $S = \{x \mid P(x)\}$

$$\{x \in \mathbb{R} \mid -2 \leq x \leq 5\} \quad \{x \in \mathbb{N} \mid x \text{ é primo e } x \leq 47\}$$

Recursão Especificar um conjunto por meio de uma função recursiva

$$\begin{cases} 1 \in A \\ \text{se } x \in A \text{ e } x + 3 < 11, \text{ então } x + 3 \in A. \end{cases}$$

Formas para definir conjuntos

Chaves Listar os elementos do conjunto entre chaves

$$\{ovo, carne, macarrão\} \quad \{1, 2, 3, \dots, 50\} \quad \{lua, \pi, 1\}$$

Propriedade Especificar uma propriedade para definir um conjunto, como $S = \{x \mid P(x)\}$

$$\{x \in \mathbb{R} \mid -2 \leq x \leq 5\} \quad \{x \in \mathbb{N} \mid x \text{ é primo e } x \leq 47\}$$

Recursão Especificar um conjunto por meio de uma função recursiva

$$\begin{cases} 1 \in A \\ \text{se } x \in A \text{ e } x + 3 < 11, \text{ então } x + 3 \in A. \end{cases}$$

Característica Especificar por meio de uma função característica

$$A = \begin{cases} 1, & \text{se } x = 0, 2, 4, 6, 8 \\ 0, & \text{caso contrário} \end{cases}$$

Exemplos de conjuntos importantes

$\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$ é o conjunto dos números naturais

$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ é o conjunto dos números inteiros

$\mathbb{Z}^+ = \{1, 2, 3, 4, \dots\}$ é o conjunto dos números inteiros positivos

\mathbb{R} é o conjunto dos números reais

\mathbb{R}^+ é o conjunto dos números reais positivos

\mathbb{C} é o conjunto dos números complexos

Igualdade de conjuntos

$$A = B$$

Dois conjuntos são ditos **iguais** sse eles possuem os mesmos elementos

Igualdade de conjuntos

$$A = B$$

Dois conjuntos são ditos **iguais** sse eles possuem os mesmos elementos

Formalmente, $A = B$ significa

$$(\forall x)[(x \in A \rightarrow x \in B) \wedge (x \in B \rightarrow x \in A)]$$

ou

$$A = B \leftrightarrow \forall x : (x \in A \leftrightarrow x \in B)$$

Igualdade de conjuntos

$$A = B$$

Dois conjuntos são ditos **iguais** sse eles possuem os mesmos elementos

Formalmente, $A = B$ significa

$$(\forall x)[(x \in A \rightarrow x \in B) \wedge (x \in B \rightarrow x \in A)]$$

ou

$$A = B \leftrightarrow \forall x : (x \in A \leftrightarrow x \in B)$$

A DEFINIÇÃO DE IGUALDADE DE CONJUNTO IMPLICA EM

$$\{a, b, c, d\} = \{d, a, c, b\}$$

$$\{a, a, a, a, b, b, b, b, c, d\} = \{d, a, c, b\}$$

Subconjuntos

$$A \subseteq B$$

O conjunto A é dito um **subconjunto** de B sse todo elemento de A é também um elemento de B .

Subconjuntos

$$A \subseteq B$$

O conjunto A é dito um **subconjunto** de B sse todo elemento de A é também um elemento de B .

Formalmente, $A \subseteq B$ significa

$$\forall x : (x \in A \rightarrow x \in B)$$

Subconjuntos

$$A \subseteq B$$

O conjunto A é dito um **subconjunto** de B sse todo elemento de A é também um elemento de B .

Formalmente, $A \subseteq B$ significa

$$\forall x : (x \in A \rightarrow x \in B)$$

$$A \subset B$$

A é um **subconjunto próprio** de B sse cada elemento de A estiver em B e pelo menos um elemento de B não está em A .

Formalmente, $A \subset B$ significa

$$\begin{aligned} A \subset B &\Leftrightarrow \forall x : (x \in A \rightarrow x \in B) \wedge \exists x : (x \in B \wedge x \notin A) \\ &\Leftrightarrow A \subseteq B \wedge A \neq B \end{aligned}$$

Subconjuntos

$$A \subseteq B$$

O conjunto A é dito um **subconjunto** de B sse todo elemento de A é também um elemento de B .

Formalmente, $A \subseteq B$ significa

$$\forall x : (x \in A \rightarrow x \in B)$$

A **está contido** em B e B **contêm** A são formas alternativas de dizer que A é um subconjunto de B.

- ▶ O conjunto dos naturais é um subconjunto dos inteiros
- ▶ O conjunto dos seres humanos contêm um subconjunto de pessoas do sexo masculino
- ▶ O subconjunto de pessoas do sexo feminino está contido no conjunto dos seres humanos.

Subconjuntos

Exemplo 1

$$A = \{1, 7, 9, 15\} \quad B = \{7, 9\} \quad C = \{7, 9, 15, 20\}$$

Diga se as sentenças são verdadeiras ou falsas (e justifique)

$$\begin{array}{ll} B \subseteq C & 15 \in C \\ B \subseteq A & \{7, 9\} \subseteq B \\ B \subset A & \{7\} \subset A \\ A \not\subset C & \emptyset \subseteq C \end{array}$$

Subconjuntos

Exemplo 1

$$A = \{1, 7, 9, 15\} \quad B = \{7, 9\} \quad C = \{7, 9, 15, 20\}$$

Diga se as sentenças são verdadeiras ou falsas (e justifique)

$$\begin{array}{ll} B \subseteq C & 15 \in C \\ B \subseteq A & \{7, 9\} \subseteq B \\ B \subset A & \{7\} \subset A \\ A \not\subset C & \emptyset \subseteq C \end{array}$$

Todas são verdadeiras!!!

Subconjuntos

Exemplo 1

$$A = \{1, 7, 9, 15\} \quad B = \{7, 9\} \quad C = \{7, 9, 15, 20\}$$

Diga se as sentenças são verdadeiras ou falsas (e justifique)

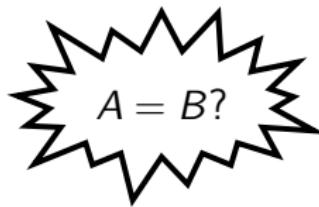
$$\begin{array}{ll} B \subseteq C & 15 \in C \\ B \subseteq A & \{7, 9\} \subseteq B \\ B \subset A & \{7\} \subset A \\ A \not\subset C & \emptyset \subseteq C \end{array}$$

O conjunto Vazio é um subconjunto de todo conjunto

Subconjuntos

CONJUNTOS PODEM TER OUTROS CONJUNTOS

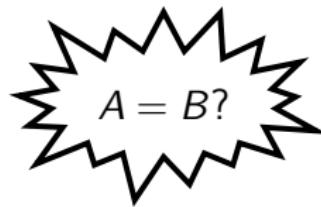
$$A = \{\emptyset, \{a\}, \{b\}, \{a, b\}\} \quad B = \{x \mid x \text{ é um subconjunto de } \{a, b\}\}$$



Subconjuntos

CONJUNTOS PODEM TER OUTROS CONJUNTOS

$$A = \{\emptyset, \{a\}, \{b\}, \{a, b\}\} \quad B = \{x \mid x \text{ é um subconjunto de } \{a, b\}\}$$

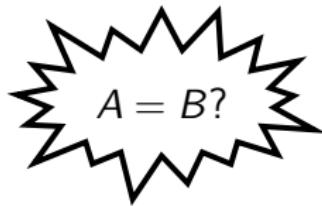


Quais são os subconjuntos de $\{a, b\}$?

Subconjuntos

CONJUNTOS PODEM TER OUTROS CONJUNTOS

$$A = \{\emptyset, \{a\}, \{b\}, \{a, b\}\} \quad B = \{x \mid x \text{ é um subconjunto de } \{a, b\}\}$$



Quais são os subconjuntos de $\{a, b\}$?

$$\emptyset \quad \{a\} \quad \{b\} \quad \{a, b\}$$

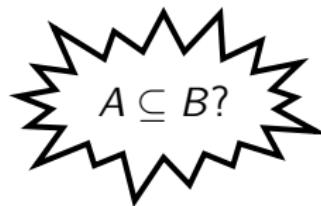
Logo, $A=B$ pois todo elemento de A está em B e vice-versa!!!

Subconjuntos

CONJUNTOS PODEM TER OUTROS CONJUNTOS

A é um conjunto

$$B = \{A, \{A\}\}$$



Subconjuntos

CONJUNTOS PODEM TER OUTROS CONJUNTOS

A é um conjunto

$$B = \{A, \{A\}\}$$



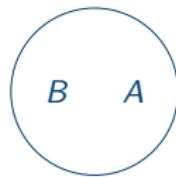
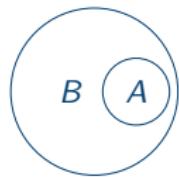
$$\begin{array}{ll} A \in B & \text{e} \\ \{A\} \subseteq B & \text{e} \end{array} \quad \begin{array}{l} \{A\} \in B \\ \{\{A\}\} \subseteq B \end{array}$$

Como A é um elemento de B , portanto A não é um subconjunto de B . Então **não é verdade que $A \subseteq B$!!!**

Diagramas de Venn

Se os conjuntos A e B forem representados por regiões no plano, relações entre A e B podem ser representadas por desenhos chamados de **Diagramas de Venn**.

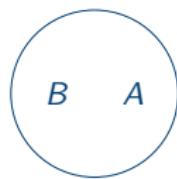
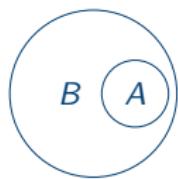
$$A \subseteq B$$



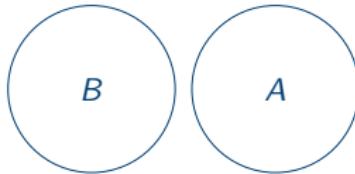
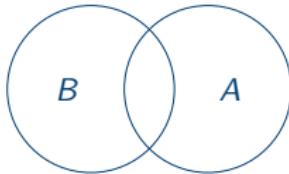
Diagramas de Venn

Se os conjuntos A e B forem representados por regiões no plano, relações entre A e B podem ser representadas por desenhos chamados de **Diagramas de Venn**.

$$A \subseteq B$$



$$A \not\subseteq B$$



Conjunto potência

Dado um conjunto A , o **conjunto potência** de A , denotado por $\mathcal{P}(A)$, é o conjunto formado por todos os subconjuntos de A .

Exemplo 2

Seja $A = \{1, 2, 3\}$

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

Conjunto potência

Dado um conjunto A , o **conjunto potência** de A , denotado por $\mathcal{P}(A)$, é o conjunto formado por todos os subconjuntos de A .

Exemplo 2

Seja $A = \{1, 2, 3\}$

$$\mathcal{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

Exemplo 3

Seja A o conjunto vazio \emptyset

$$\mathcal{P}(A) = \{\emptyset\}$$

Como os conjuntos **não são ordenados**, uma estrutura diferente é necessária para representar coleções ordenadas.

SEQUÊNCIA

- ▶ Uma **sequência** é uma lista de objetos em ordem.
 - ▶ um “primeiro elemento”, um “segundo elemento”,...
 - ▶ a lista pode ser finita ou não

Sequências

Como os conjuntos **não são ordenados**, uma estrutura diferente é necessária para representar coleções ordenadas.

SEQUÊNCIA

- Uma **sequência** é uma lista de objetos em ordem.
 - um “primeiro elemento”, um “segundo elemento”,...
 - a lista pode ser finita ou não

EXEMPLOS

- 1,0,0,1,0,1,0,0,1,1,1
- 1,4,9,16,25,... (“quadrados dos números positivos”) é infinita
- A sequência finita 1,2,4,...,256 pode ser denotada por $(2^n)_{0 \leq n \leq 8}$

Cardinalidade de conjuntos

Um conjunto é dito **contável** se for o conjunto correspondente a alguma sequência.

- ▶ Os elementos do conjunto podem ser arranjados em uma lista ordenada, a qual pode, portanto, ser contada.
- ▶ Todos os conjuntos finitos são contáveis.
- ▶ Um conjunto que não é contável é dito **incontável**.

Cardinalidade de conjuntos

Um conjunto é dito **contável** se for o conjunto correspondente a alguma sequência.

- ▶ Os elementos do conjunto podem ser arranjados em uma lista ordenada, a qual pode, portanto, ser contada.
- ▶ Todos os conjuntos finitos são contáveis.
- ▶ Um conjunto que não é contável é dito **incontável**.

Cardinalidade de X *número de elementos em um conjunto X . A cardinalidade de X é denotado por $|X|$*

$$|\{2, 5, 7\}| = 3$$

Emparelhamento

Para que dois conjuntos X e Y possuam a mesma cardinalidade :

- emparelhamento de cada x em X com apenas um y em Y
- cada elemento de Y seja usado apenas uma vez neste emparelhamento

Emparelhamento

Para que dois conjuntos X e Y possuam a mesma cardinalidade :

- emparelhamento de cada x em X com apenas um y em Y
- cada elemento de Y seja usado apenas uma vez neste emparelhamento

Exemplo 4

$$X = \{2, 5, 7\} \quad \text{e} \quad Y = \{?, !, \#\}$$

Emparelhamento

Para que dois conjuntos X e Y possuam a mesma cardinalidade :

- emparelhamento de cada x em X com apenas um y em Y
- cada elemento de Y seja usado apenas uma vez neste emparelhamento

Exemplo 4

$$X = \{2, 5, 7\} \quad \text{e} \quad Y = \{?, !, \#\}$$

$$2 \leftrightarrow ?, \quad 5 \leftrightarrow \#, \quad 7 \leftrightarrow !$$

Emparelhamento

Para que dois conjuntos X e Y possuam a mesma cardinalidade :

- emparelhamento de cada x em X com apenas um y em Y
- cada elemento de Y seja usado apenas uma vez neste emparelhamento

Exemplo 4

$$X = \{2, 5, 7\} \quad \text{e} \quad Y = \{?, !, \#\}$$

$$2 \leftrightarrow ?, \quad 5 \leftrightarrow \#, \quad 7 \leftrightarrow !$$

O emparelhamento mostra que ambos os conjuntos possuem a mesma cardinalidade .

Produto cartesiano

O **produto cartesiano** de dois conjuntos A e B , é o conjunto de todos os pares ordenados (a, b) , onde $a \in A$ e $b \in B$, ou seja:

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$$

Produto cartesiano

O **produto cartesiano** de dois conjuntos A e B , é o conjunto de todos os pares ordenados (a, b) , onde $a \in A$ e $b \in B$, ou seja:

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$$

Exemplo 5

Qual é o produto de $A = \{1, 2\}$ e $B = \{a, b, c\}$?

Produto cartesiano

O **produto cartesiano** de dois conjuntos A e B , é o conjunto de todos os pares ordenados (a, b) , onde $a \in A$ e $b \in B$, ou seja:

$$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$$

Exemplo 5

Qual é o produto de $A = \{1, 2\}$ e $B = \{a, b, c\}$?

$$A \times B = \{(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)\}$$

Conjunto universo

Conjunto que contêm “todas as coisas”, denominado U , que contêm todos os objetos para os quais a discussão faz sentido

Conjunto universal

Operações sobre conjuntos

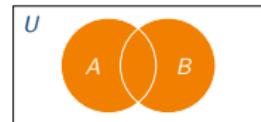
União

$$A \cup B = \{x \in U \mid x \in A \vee x \in B\}$$

$$x \in A \cup B \Leftrightarrow x \in A \vee x \in B$$

Notação

$$\cup_{i=1}^n A_i = A_1 \cup A_2 \cup \dots \cup A_n$$



Operações sobre conjuntos

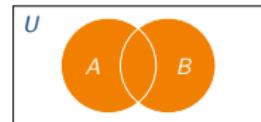
União

$$A \cup B = \{x \in U \mid x \in A \vee x \in B\}$$

Notação

$$x \in A \cup B \Leftrightarrow x \in A \vee x \in B$$

$$\cup_{i=1}^n A_i = A_1 \cup A_2 \cup \dots \cup A_n$$



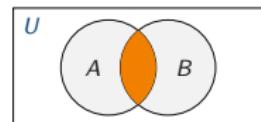
Interseção

$$A \cap B = \{x \in U \mid x \in A \wedge x \in B\}$$

$$x \in A \cap B \Leftrightarrow x \in A \wedge x \in B$$

Notação

$$\cap_{i=1}^n A_i = A_1 \cap A_2 \cap \dots \cap A_n$$



Operações sobre conjuntos

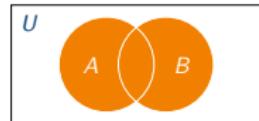
União

$$A \cup B = \{x \in U \mid x \in A \vee x \in B\}$$

Notação

$$x \in A \cup B \Leftrightarrow x \in A \vee x \in B$$

$$\cup_{i=1}^n A_i = A_1 \cup A_2 \cup \dots \cup A_n$$



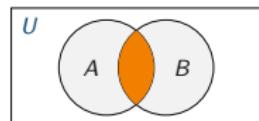
Interseção

$$A \cap B = \{x \in U \mid x \in A \wedge x \in B\}$$

$$x \in A \cap B \Leftrightarrow x \in A \wedge x \in B$$

Notação

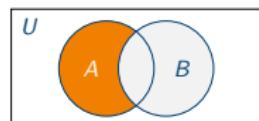
$$\cap_{i=1}^n A_i = A_1 \cap A_2 \cap \dots \cap A_n$$



Diferença

$$A - B = \{x \in U \mid x \in A \vee x \notin B\}$$

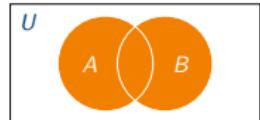
$$x \in A - B \Leftrightarrow x \in A \vee x \in B$$



Operações sobre conjuntos

União

$$A \cup B = \{x \in U \mid x \in A \vee x \in B\}$$

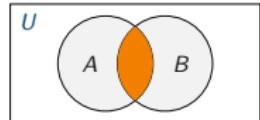


Notação

$$\cup_{i=1}^n A_i = A_1 \cup A_2 \cup \dots \cup A_n$$

Interseção

$$A \cap B = \{x \in U \mid x \in A \wedge x \in B\}$$



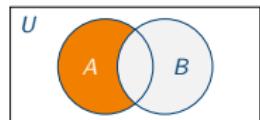
Notação

$$\cap_{i=1}^n A_i = A_1 \cap A_2 \cap \dots \cap A_n$$

Diferença

$$A - B = \{x \in U \mid x \in A \vee x \notin B\}$$

$$x \in A - B \Leftrightarrow x \in A \vee x \notin B$$



Complemento

$$\bar{A} = \{x \in U \mid x \notin A\}$$

$$x \in \bar{A} \Leftrightarrow x \notin A$$



Identidades de conjuntos

Comutatividade	$A \cap B = B \cap A$	$A \cup B = B \cup A$
Associatividade	$(A \cap B) \cap C = A \cap (B \cap C)$	$(A \cup B) \cup C = A \cup (B \cup C)$
Distributividade	$(A \cup B) \cap C = (A \cup B) \cap (B \cup C)$	$(A \cap B) \cup C = (A \cap B) \cup (B \cap C)$
União e intersecção com U	$A \cap U = A$	$A \cup U = U$
Complemento duplo		$\overline{\overline{A}} = A$
Idempotência	$A \cap A = A$	$A \cup A = A$
De Morgan	$\overline{(A \cap B)} = \overline{A} \cup \overline{B}$	$\overline{(A \cup B)} = \overline{A} \cap \overline{B}$
Absorção	$A \cap (A \cup B) = A$	$A \cup (A \cap B) = A$
Diferença de conjuntos		$A - B = A \cap \overline{B}$
União e intersecção com \emptyset	$A \cap \emptyset = \emptyset$	$A \cup \emptyset = A$
União e intersecção com o complemento	$A \cap \overline{A} = \emptyset$	$A \cup \overline{A} = U$
Complementos de U e \emptyset	$\overline{U} = \emptyset$	$U = \overline{\emptyset}$

Conjuntos disjuntos

Dois conjuntos são chamados **disjuntos** se e só se eles não têm nenhum elemento em comum

A e B são disjuntos



$$A \cap B = \emptyset$$

Conjuntos disjuntos

Dois conjuntos são chamados **disjuntos** se e somente se eles não têm nenhum elemento em comum

A e B são disjuntos



$$A \cap B = \emptyset$$

Os conjuntos A_1, A_2, \dots, A_n são chamados **mutuamente disjuntos** (ou **disjuntos par-a-par**) se e somente se $A_i \cap A_j = \emptyset$ para todos $i, j = 1, 2, \dots, n$ e $i \neq j$.

Conjuntos disjuntos

Dois conjuntos são chamados **disjuntos** sse eles não têm nenhum elemento em comum

A e B são disjuntos

\leftrightarrow

$$A \cap B = \emptyset$$

Os conjuntos A_1, A_2, \dots, A_n são chamados **mutuamente disjuntos** (ou **disjuntos par-a-par**) se e somente se $A_i \cap A_j = \emptyset$ para todos $i, j = 1, 2, \dots, n$ e $i \neq j$.

PARTIÇÃO

Seja uma coleção de conjuntos não vazios $\{A_1, A_2, \dots, A_n\}$ é uma **partição** sse

- $A = A_1 \cup A_2 \cup \dots \cup A_n$
- A_1, A_2, \dots, A_n são mutuamente disjuntos

Questions?

Teoria de Conjuntos e Funções
– Conjuntos –

Teoria dos Grafos e Computabilidade

— Funções —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Funções: uma introdução

Funções são associações de cada elemento de um conjunto a um elemento particular de outro conjunto.

$$f(a) = b$$

Sequências são listas ordenadas de elementos.

$$0, 1, 2, 3, 4, \dots$$

Somatórios e produtórios são a soma e o produto, respectivamente, dos termos de uma sequência numérica.

$$\sum_{i=m}^n a_i = a_m + a_{m+1} + \dots + a_n$$

Funções: uma introdução

Frequentemente temos que **associar** cada elemento de um conjunto a um elemento particular de outro conjunto.

Por exemplo, podemos:

1. Associar cada aluno de Grafos a um conceito A, B, C, D, E ou F.
2. Associar cada inteiro ao seu quadrado.
3. Associar cada professor a uma disciplina.

O conceito de função formaliza este tipo de associação.

Funções: uma introdução

Frequentemente temos que **associar** cada elemento de um conjunto a um elemento particular de outro conjunto.

Por exemplo, podemos:

1. Associar cada aluno de Grafos a um conceito A, B, C, D, E ou F.
2. Associar cada inteiro ao seu quadrado.
3. Associar cada professor a uma disciplina.

O conceito de função formaliza este tipo de associação.

Sejam A e B conjuntos não-vazios. Uma função f de A para B é uma associação de exatamente um elemento de B a cada elemento de A .

$$f(a) = b$$

se b for o único elemento de B associado por meio de f ao elemento a de A .

Funções: uma introdução

Se f é uma função de A para B , para denotar o tipo da função, a função pode ser escrita como

$$f : A \mapsto B$$

domínio o conjunto A é chamado de domínio de f .

contra-domínio o conjunto B é chamado de co-domínio ou contra-domínio de f .

imagem a imagem de f é o conjunto de valores que f pode assumir

$$\text{imagem de } f = \{b \in B \mid b = f(a) \text{ para algum } a \in A\}$$

imagem inversa a imagem inversa (ou inversa) de um elemento $b \in B$ é o conjunto de valores de $a \in A$ que são mapeados a b via f

$$\text{inversa de } b = \{a \in A \mid f(a) = b\}$$

Exemplos de funções

Exemplo 6

Sejam os conjuntos $A = \{x, y, z\}$ e $B = \{1, 2, 3, 4\}$

Seja a função $f : A \mapsto B$ definida como:

$$f(x) = 2 \quad f(y) = 4 \quad f(z) = 2$$

domínio $\{x, y, z\}$

inversa de 1 e 3 é \emptyset

contra-domínio $\{1, 2, 3, 4\}$

inversa de 2 é $\{x, z\}$

imagem $\{2, 4\}$

inversa de 4 é $\{y\}$

A função f pode ser representada como o conjunto de pares ordenados :

$$f = \{(x, 2), (y, 4), (z, 2)\}$$

Exemplos de funções

Exemplo 7

Alguns exemplos

- Função quadrado $f: \mathbb{R} \rightarrow \mathbb{R}$

$$f(x) = x^2 \text{ ou } f : x \mapsto x^2$$

- Função sucessor $f: \mathbb{Z} \rightarrow \mathbb{Z}$

$$f(x) = x + 1 \text{ ou } f : x \mapsto x + 1$$

- Função constante $f: \mathbb{N} \rightarrow \mathbb{Z}$

$$f(r) = 2 \text{ ou } f : r \mapsto 2$$

- Função pai $f: P \rightarrow P$, em que P é o conjunto de todas as pessoas

$$f : p \mapsto p', \text{ em que } p' \text{ é o pai de } p$$

Igualdade de funções

Duas funções f e g funções são iguais sse elas:

- ▶ têm o mesmo domínio
- ▶ têm o mesmo contra-domínio
- ▶ mapeiam cada elemento do domínio ao mesmo elemento do contra-domínio.

Formalmente, para duas funções f e g definidas em $A \rightarrow B$:

$$f = g \text{ sse } \forall a \in A : f(a) = g(a)$$

Igualdade de funções

Duas funções f e g funções são iguais sse elas:

- ▶ têm o mesmo domínio
- ▶ têm o mesmo contra-domínio
- ▶ mapeiam cada elemento do domínio ao mesmo elemento do contra-domínio.

Formalmente, para duas funções f e g definidas em $A \rightarrow B$:

$$f = g \text{ sse } \forall a \in A : f(a) = g(a)$$

Exemplo 8

Sejam $f(x) = |x|$ e $g(x) = \sqrt{x^2}$. Então, $f = g$ uma vez que $\forall x \in \mathbb{R}$

$$|x| = \sqrt{x^2}$$

Função injetora

Uma função $f : A \rightarrow B$ é uma função injetora (ou injetiva ou um-para-um) se e somente se para todos $a_1, a_2 \in A$

$$a_1 \neq a_2 \rightarrow f(a_1) \neq f(a_2)$$

Uma função é dita injetora se cada elemento do domínio é mapeado para uma elemento diferente do contra-domínio.

Função injetora

Uma função $f : A \rightarrow B$ é uma função injetora (ou injetiva ou um-para-um) se e somente se para todos $a_1, a_2 \in A$

$$a_1 \neq a_2 \rightarrow f(a_1) \neq f(a_2)$$

Uma função é dita injetora se **cada elemento** do domínio é mapeado para uma **elemento diferente** do contra-domínio.

Exemplo 9

Seja $f : \mathbb{R} \rightarrow \mathbb{R}$. Então,

$$f(x) = x + 1 \quad (\text{injetora})$$

$$f(x) = \frac{x}{10} \quad (\text{injetora})$$

$$f(x) = x^2 \quad (\text{não é injetora})$$

$$f(x) = (x - 1)(x - 2) \quad (\text{não é injetora})$$

Função sobrejetora

Uma função $f : A \rightarrow B$ é uma função sobrejetora (ou sobrejetiva) se e somente se para todo $b \in B$ há $a \in A$ de forma que $f(a) = b$

Uma função é dita sobrejetora se cada elemento do contra-domínio é imagem de pelo menos um elemento do domínio.

Função sobrejetora

Uma função $f : A \rightarrow B$ é uma função sobrejetora (ou sobrejetiva) se e somente se para todo $b \in B$ há $a \in A$ de forma que $f(a) = b$

Uma função é dita sobrejetora se **cada elemento** do contra-domínio é imagem de pelo menos um elemento do domínio.

Exemplo 10

Seja $f : \mathbb{R} \rightarrow \mathbb{R}$. Então,

$$f(x) = x + 1 \quad (\text{sobrejetora})$$

$$f(x) = \frac{x}{10} \quad (\text{sobrejetora})$$

$$f(x) = x^2 \quad (\text{não é sobrejetora})$$

$$f(x) = 2^x \quad (\text{não é sobrejetora})$$

Função bijetora

Uma função $f : A \rightarrow B$ é uma função bijetora se e somente se ele é sobrejetora e injetora.

Função bijetora

Uma função $f : A \rightarrow B$ é uma função bijetora se e somente se ele é sobrejetora e injetora.

Exemplo 11

Seja $f : \mathbb{R} \rightarrow \mathbb{R}$. Então,

$$f(x) = x + 1 \quad (\text{bijetora})$$

$$f(x) = \frac{x}{10} \quad (\text{bijetora})$$

$$f(x) = 2^x \quad (\text{não é bijetora})$$

$$f(x) = (x - 1)(x - 2) \quad (\text{não é bijetora})$$

Função inversa

Uma função $f : A \rightarrow B$ é uma função bijetora. A função inversa de f , denotada por $f^{-1} : B \rightarrow A$ tal que

$$f^{-1}(y) = x \text{ sse } y = f(x)$$

Função inversa

Uma função $f : A \rightarrow B$ é uma função bijetora. A função inversa de f , denotada por $f^{-1} : B \rightarrow A$ tal que

$$f^{-1}(y) = x \text{ sse } y = f(x)$$

Exemplo 12

A função $f : \mathbb{N} \rightarrow \mathbb{N}$ definida como $f(x) = x + 1$ é inversível porque ela é bijetora. Sua inversa

$$f^{-1}(x) = x - 1$$

Função inversa

Uma função $f : A \rightarrow B$ é uma função bijetora. A **função inversa** de f , denotada por $f^{-1} : B \rightarrow A$ tal que

$$f^{-1}(y) = x \text{ sse } y = f(x)$$

Exemplo 12

A função $f : \mathbb{N} \rightarrow \mathbb{N}$ definida como $f(x) = x + 1$ é inversível porque ela é bijetora. Sua inversa

$$f^{-1}(x) = x - 1$$

Exemplo 13

A função $f : \mathbb{R} \rightarrow \mathbb{R}$ definida como $f(x) = x^2$ não é inversível porque ela não é bijetora: $f(1) = f(-1) = 1$, logo f^{-1} não é definido.

Composição de funções

Sejam $g : A \rightarrow B'$ e $f : B \rightarrow C$ funções tais que a imagem de g é um subconjunto do domínio de f ($B' \subseteq B$).

A **função composta** de f com g , denotada por $f \circ g : A \rightarrow C$, é definida para todo $a \in A$ da seguinte forma:

$$(f \circ g)(a) = f(g(a))$$

A função $f \circ g$ é chamada de **composição de f e g** .

Composição de funções

Exemplo 14

Sejam $f : \mathbb{Z} \rightarrow \mathbb{Z}$ e $g : \mathbb{Z} \rightarrow \mathbb{Z}$ tais que $f(n) = n+1$ e $g(n) = n^2$

$$f \circ g = g \circ f?$$

Composição de funções

Exemplo 14

Sejam $f : \mathbb{Z} \rightarrow \mathbb{Z}$ e $g : \mathbb{Z} \rightarrow \mathbb{Z}$ tais que $f(n) = n+1$ e $g(n) = n^2$

$$f \circ g = g \circ f?$$

$\forall n \in \mathbb{Z}$, temos que

$$(f \circ g)(n) = f(g(n)) = f(n^2) = n^2 + 1$$

no entanto,

$$(g \circ f)(n) = g(f(n)) = g(n+1) = (n+1)^2$$

Portanto,

$$f \circ g \neq g \circ f$$

Composição de funções

Exemplo 14

Sejam $f : \mathbb{Z} \rightarrow \mathbb{Z}$ e $g : \mathbb{Z} \rightarrow \mathbb{Z}$ tais que $f(n) = n+1$ e $g(n) = n^2$

$$f \circ g = g \circ f?$$

$\forall n \in \mathbb{Z}$, temos que

$$(f \circ g)(n) = f(g(n)) = f(n^2) = n^2 + 1$$

no entanto,

$$(g \circ f)(n) = g(f(n)) = g(n+1) = (n+1)^2$$

Portanto,

$$f \circ g \neq g \circ f$$

A composição não é comutativa

Composição de funções

Dado um domínio A, a função identidade $I_A : A \rightarrow A$ é definida por

$$I_A(a) = a, \quad \forall a \in A$$

Composição de funções

Dado um domínio A, a função identidade $I_A : A \rightarrow A$ é definida por

$$I_A(a) = a, \quad \forall a \in A$$

Teorema Se f é uma função de X para Y . Sejam I_X uma função identidade de X e I_Y uma função identidade de Y .

$$f \circ I_X = f$$

$$I_Y \circ f = f$$

Composição de funções

Dado um domínio A, a função identidade $I_A : A \rightarrow A$ é definida por

$$I_A(a) = a, \quad \forall a \in A$$

Teorema Se f é uma função de X para Y . Sejam I_X uma função identidade de X e I_Y uma função identidade de Y .

$$\begin{aligned} f \circ I_X &= f \\ I_Y \circ f &= f \end{aligned}$$

Teorema Se $f : X \rightarrow Y$ é uma função bijetora com função inversa definida por $f^{-1} : Y \rightarrow X$, então

$$\begin{aligned} f^{-1} \circ f &= I_X \\ f \circ f^{-1} &= I_Y \end{aligned}$$

Teorema Se $f : X \rightarrow Y$ e $g : Y \rightarrow Z$ são funções injetoras, então

$g \circ f$ é injetora

Teorema Se $f : X \rightarrow Y$ e $g : Y \rightarrow Z$ são funções injetoras, então

$g \circ f$ é injetora

Teorema Se $f : X \rightarrow Y$ e $g : Y \rightarrow Z$ são funções sobrejetoras, então

$g \circ f$ é sobrejetora

Funções piso e teto

A função **piso** (em inglês, *floor*) associa um número real x ao maior número inteiro menor ou igual a ele.

O valor da função piso é denotado por $\lfloor x \rfloor$

A função **teto** (em inglês, *ceiling*) associa um número real x ao menor número inteiro maior ou igual a ele.

O valor da função teto é denotado por $\lceil x \rceil$

Ambas as funções (piso e teto) $f : \mathbb{R} \rightarrow \mathbb{Z}$.

Funções piso e teto

A função **piso** (em inglês, *floor*) associa um número real x ao maior número inteiro menor ou igual a ele.

O valor da função piso é denotado por $\lfloor x \rfloor$

A função **teto** (em inglês, *ceiling*) associa um número real x ao menor número inteiro maior ou igual a ele.

O valor da função teto é denotado por $\lceil x \rceil$

Ambas as funções (piso e teto) $f : \mathbb{R} \rightarrow \mathbb{Z}$.

Exemplo 15

- ▶ $\lfloor \pi \rfloor = 3$ e $\lceil \pi \rceil = 4$
- ▶ $\lfloor 3 \rfloor = 3$ e $\lceil 3 \rceil = 3$

Algumas propriedades

1. $\lfloor -x \rfloor = -\lceil x \rceil$
2. $\lceil -x \rceil = -\lfloor x \rfloor$
3. $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$

Algumas propriedades

1. $\lfloor -x \rfloor = -\lceil x \rceil$
2. $\lceil -x \rceil = -\lfloor x \rfloor$
3. $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$

Algumas observações

1. $x = \lfloor x \rfloor + \epsilon$, para algum $0 \leq \epsilon < 1$
2. $x = \lceil x \rceil - \epsilon$, para algum $0 \leq \epsilon < 1$

Funções piso e teto

Exemplo 16

Demonstre a propriedade $\lfloor -x \rfloor = -\lceil x \rceil$

Funções piso e teto

Exemplo 16

Demonstre a propriedade $\lfloor -x \rfloor = -\lceil x \rceil$

Solução Seja o número x representado por $n + \epsilon$, em que $n \in \mathbb{Z}$ e $0 \leq \epsilon < 1$. Há dois casos a se considerar ($\epsilon = 0$ ou não)

Caso 1 $\epsilon = 0$, assim $x = n$ e $\lfloor -x \rfloor = -\lceil x \rceil = -n$

Caso 2 $0 < \epsilon < 1$. Assim

$$\begin{aligned}\lfloor -x \rfloor &= \lfloor -(n + \epsilon) \rfloor \\ &= \lfloor -n - \epsilon \rfloor \\ &= -(n + 1)\end{aligned}$$

e,

$$\begin{aligned}-\lceil x \rceil &= -\lceil n + \epsilon \rceil \\ &= -(n + 1)\end{aligned}$$

Portanto, $\lfloor -x \rfloor = -\lceil x \rceil$

Funções parciais

Uma função parcial f de um conjunto A para um conjunto B é uma associação a cada elemento a em um subconjunto de A, chamado de domínio de definição de f , a um único elemento b de B.

Os conjuntos A e B são chamados de domínio e contra-domínio de f , respectivamente.

Dizemos que f é indefinida para elementos de A que não estão no domínio de definição de f . Quando o domínio de definição de f é o próprio domínio A, dizemos que f é uma função total.

Funções parciais

Uma função parcial f de um conjunto A para um conjunto B é uma associação a cada elemento a em um subconjunto de A , chamado de domínio de definição de f , a um único elemento b de B .

Os conjuntos A e B são chamados de domínio e contra-domínio de f , respectivamente.

Dizemos que f é indefinida para elementos de A que não estão no domínio de definição de f . Quando o domínio de definição de f é o próprio domínio A , dizemos que f é uma função total.

Exemplo 17

A função $f : \mathbb{Z} \rightarrow \mathbb{R}$ em que $f(n) = \sqrt{n}$ é uma função parcial de \mathbb{Z} para \mathbb{R} em que o domínio de definição é o conjunto dos inteiros não-negativos. A função f é indefinida para os inteiros negativos.

Questions?

Teoria de Conjuntos e Funções
– Funções –



Teoria dos Grafos e Computabilidade

— Sequências —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Sequências: uma introdução

Sequências são listas ordenadas de elementos. Ainda, são estruturas discretas que aparecem com frequência em computação

1. progressão aritmétrica
2. progressão geométrica
3. strings

Sequências: uma introdução

Sequências são listas ordenadas de elementos. Ainda, são estruturas discretas que aparecem com frequência em computação

1. progressão aritmétrica
2. progressão geométrica
3. strings

Formalmente, uma sequência é uma função definida do conjunto dos naturais (a partir de um valor específico, normalmente 0 ou 1) para um conjunto arbitrário S .

Usamos a_n , chamado de termo da sequência, para denotar a imagem do inteiro n , e a sequência inteira é frequentemente denotada como $\{a_n\}$.

Sequências: uma introdução

Sequências são listas ordenadas de elementos. Ainda, são estruturas discretas que aparecem com frequência em computação

1. progressão aritmétrica
2. progressão geométrica
3. strings

Formalmente, uma sequência é uma função definida do conjunto dos naturais (a partir de um valor específico, normalmente 0 ou 1) para um conjunto arbitrário S .

Usamos a_n , chamado de termo da sequência, para denotar a imagem do inteiro n , e a sequência inteira é frequentemente denotada como $\{a_n\}$.

Exemplo 18

1. Sequência dos n primeiros naturais: $0, 1, 2, 3, 4, 5, 6, \dots$
2. Sequência dos n números primos: $2, 3, 5, 7, 11, \dots$

Sequências

Sequências importantes são as progressões aritméticas

progressão aritmética sequência da forma

$$a, a + d, a + 2d, a + 3d, \dots, a + nd$$

em que o termo inicial a e a diferença comum d são números reais.

Sequências

Sequências importantes são as progressões aritméticas e geométricas.

progressão aritmética sequência da forma

$$a, a + d, a + 2d, a + 3d, \dots, a + nd$$

em que o termo inicial a e a diferença comum d são números reais.

progressão geométrica sequência da forma

$$a, ar, ar^2; ar^3; \dots, ar^n$$

em que o termo inicial a e a razão r são números reais.

Fórmulas explícitas para sequências

Uma **fórmula explícita** define como obter o k -ésimo termo de uma sequência diretamente em função de k .

Fórmulas explícitas para sequências

Uma **fórmula explícita** define como obter o k -ésimo termo de uma sequência diretamente em função de k .

Exemplo 19

A sequência

$$1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \dots, \frac{(-1)^n}{n+1}, \dots$$

pode ser definida como uma função dos naturais para os reais

$$0 \mapsto 1 \quad 1 \mapsto -\frac{1}{2} \quad 2 \mapsto \frac{1}{3} \quad 3 \mapsto -\frac{1}{4} \quad \dots \quad n \mapsto \frac{(-1)^n}{n+1}$$

Formalmente, a sequência pode ser descrita por $f : \mathbb{N} \rightarrow \mathbb{R}$, em que

$$f(n) = \frac{(-1)^n}{n+1}, \text{ para } n = 1, 2, 3, \dots$$

Fórmulas explícitas para sequências

Uma **fórmula explícita** define como obter o k -ésimo termo de uma sequência diretamente em função de k .

Exemplo 19

A mesma sequência pode ser definida como uma função dos inteiros positivos para os reais:

$$0 \mapsto 1 \quad 1 \mapsto -\frac{1}{2} \quad 2 \mapsto \frac{1}{3} \quad 3 \mapsto -\frac{1}{4} \quad \dots \quad n \mapsto \frac{(-1)^{n+1}}{n}$$

Formalmente, a sequência pode ser descrita por $g : \mathbb{Z}^+ \rightarrow \mathbb{R}$, em que

$$g(n) = \frac{(-1)^{n+1}}{n}, \text{ para } n = 1, 2, 3, \dots$$

Igualdade de sequências

Duas sequências $\{a_n\}$ e $\{b_n\}$ são iguais se para todo n

$$a_n = b_n$$

Igualdade de sequências

Duas sequências $\{a_n\}$ e $\{b_n\}$ são iguais se para todo n

$$a_n = b_n$$

Exemplo 20

Sejam as sequências (a_1, a_2, a_3, \dots) e (b_2, b_3, b_4, \dots) definidas pelas fórmulas explícitas

$$\begin{aligned} a_i &= \frac{i}{i+1} && \text{para inteiros } i \geq 1 \\ b_j &= \frac{j-1}{j} && \text{para inteiros } j \geq 2 \end{aligned}$$

As sequências são idênticas. Note

$$\begin{array}{ll} a_1 = \frac{1}{1+1} = \frac{1}{2} & b_2 = \frac{2-1}{2} = \frac{1}{2} \\ a_2 = \frac{2}{2+1} = \frac{2}{3} & b_3 = \frac{3-1}{3} = \frac{2}{3} \\ a_3 = \frac{3}{3+1} = \frac{3}{4} & b_4 = \frac{4-1}{4} = \frac{3}{4} \end{array}$$

Definindo uma sequência

Um problema comum é, dados alguns termos iniciais de uma sequência, determinar uma regra para gerar a sequência como um todo.

Maneiras típicas para definir uma sequência

1. Fórmula explícita para cada termo da sequência.
2. Algoritmo que gere a sequência.
3. Fórmula recursiva para cada termo da sequência.

Definindo uma sequência

Um problema comum é, dados alguns termos iniciais de uma sequência, determinar uma regra para gerar a sequência como um todo.

Maneiras típicas para definir uma sequência

1. Fórmula explícita para cada termo da sequência.
2. Algoritmo que gere a sequência.
3. Fórmula recursiva para cada termo da sequência.

OBSERVAÇÃO

Dado um número limitado de termos a_1, a_2, \dots, a_i de uma sequência, é possível achar uma regra mas é garantida apenas para os i termos apresentados.

Nada garante a regra valha para a_{i+1}

Definindo uma sequência

Exemplo 21

Seja a sequência cujos 5 primeiros termos são 1, 2, 3, 4, 5. A fórmula

$$a_n = n, \text{ para } n \geq 1$$

gera estes 5 termos corretamente, e prevê que $a_6 = 6$

O algoritmo

Gere todos os naturais cujo resto da divisão por 10 está entre 1 e 5

também gera estes mesmos cinco termos, e prevê que $a_6 = 11$

As duas descrições concordam para todos os termos dados, mas geram sequências diferentes. Apenas com as informações dadas não há como dizer qual sequência é mais apropriada.

Provendo uma fórmula explícita

Exemplo 22

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 7, 25, 79, 241, 727, 2185, 6559, 19681, 59047$$

Provendo uma fórmula explícita

Exemplo 22

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 7, 25, 79, 241, 727, 2185, 6559, 19681, 59047$$

Solução

Fórmula explícita para gerar os 10 primeiros termos

$$a_n = 3^n - 1, \text{ para inteiros } n \geq 1$$

Provendo uma fórmula explícita

Exemplo 22

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 7, 25, 79, 241, 727, 2185, 6559, 19681, 59047$$

Solução

Fórmula explícita para gerar os 10 primeiros termos

$$a_n = 3^n - 1, \text{ para inteiros } n \geq 1$$

Exemplo 23

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, -2, 3, -4, 5, -6, 7, -8, 9, -10$$

Provendo uma fórmula explícita

Exemplo 22

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 7, 25, 79, 241, 727, 2185, 6559, 19681, 59047$$

Solução

Fórmula explícita para gerar os 10 primeiros termos

$$a_n = 3^n - 1, \text{ para inteiros } n \geq 1$$

Exemplo 23

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, -2, 3, -4, 5, -6, 7, -8, 9, -10$$

Solução

Fórmula explícita para gerar os 10 primeiros termos

$$a_n = (-1)^{n+1} n, \text{ para inteiros } n \geq 1$$

Provendo um algoritmo para a sequência

Exemplo 24

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 2, 2, 3, 3, 3, 4, 4, 4, 4$$

Provendo um algoritmo para a sequência

Exemplo 24

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 2, 2, 3, 3, 3, 4, 4, 4, 4$$

Solução Começando de 1, em ordem crescente, cada natural n é repetido n vezes

Provendo um algoritmo para a sequência

Exemplo 24

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 2, 2, 3, 3, 3, 4, 4, 4, 4$$

Solução Começando de 1, em ordem crescente, cada natural n é repetido n vezes

Exemplo 25

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 16 primeiros termos:

$$0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0$$

Provendo um algoritmo para a sequência

Exemplo 24

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 10 primeiros termos:

$$1, 2, 2, 3, 3, 3, 4, 4, 4, 4$$

Solução Começando de 1, em ordem crescente, cada natural n é repetido n vezes

Exemplo 25

Encontre uma fórmula explícita para a sequência $\{a_n\}$ tendo os 16 primeiros termos:

$$0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0$$

Solução Para cada natural $n \geq 1$, em ordem crescente, adicione à sequência n termos 0, seguidos de n termos 1

Provendo uma fórmula recursiva

Uma **fórmula recursiva** para uma sequência define cada termo em função de termos anteriores.

Definições recursivas são baseadas em relações de recorrência.

Provendo uma fórmula recursiva

Uma **fórmula recursiva** para uma sequência define cada termo em função de termos anteriores.

Definições recursivas são baseadas em relações de recorrência.

Uma relação de recorrência para uma sequência $\{a_n\}$ é uma equação que expressa $\{a_n\}$ em termos de um ou mais termos prévios na sequência para cada $n \geq n_0$, em que n_0 é inteiro não-negativo.

Provendo uma fórmula recursiva

Uma **fórmula recursiva** para uma sequência define cada termo em função de termos anteriores.

Definições recursivas são baseadas em relações de recorrência.

Uma relação de recorrência para uma sequência $\{a_n\}$ é uma equação que expressa $\{a_n\}$ em termos de um ou mais termos prévios na sequência para cada $n \geq n_0$, em que n_0 é inteiro não-negativo.

Exemplo 26

A sequência

$$1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880$$

pode ser definida por fórmula explícita $a_n = n!$, $n \geq 0$; ou pela fórmula recursiva

$$a_n = \begin{cases} a_0 = 1 & \text{se } n = 0 \\ a_n = n \times a_{n-1} & \text{se } n \geq 1 \end{cases}$$

Questions?

Teoria de Conjuntos e Funções
– Sequências –



Teoria dos Grafos e Computabilidade

— Somatórios e produtórios —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Somatórios

Seja uma sequência $\{a_n\}$. O somatório dos termos

$$a_m, a_{m+1}, a_{m+2}, \dots, a_n$$

de $\{a_k\}$ é a soma

$$a_m + a_{m+1} + a_{m+2} + \dots + a_n$$

O somatório é representado por \sum

$$\sum_{i=m}^n a_i = a_m + a_{m+1} + a_{m+2} + \dots + a_n$$

Somatórios

Exemplo 27

Seja uma sequência a_k em $a_k = k^2$

$$\sum_{k=3}^{6} a_k$$

Somatórios

Exemplo 27

Seja uma sequência a_k em $a_k = k^2$

$$\sum_{k=3}^{6} a_k = 3^2 + 4^2 + 5^2 + 6^2$$

Exemplo 27

Seja uma sequência a_k em $a_k = k^2$

$$\sum_{k=3}^{6} a_k = 3^2 + 4^2 + 5^2 + 6^2 = 9 + 16 + 25 + 36 = 86$$

Exemplo 27

Seja uma sequência a_k em $a_k = k^2$

$$\sum_{k=3}^{6} a_k = 3^2 + 4^2 + 5^2 + 6^2 = 9 + 16 + 25 + 36 = 86$$

Uma notação alternativa para somatórios é

$$\sum_{s \in S} f(s)$$

em que S é um conjunto de domínio e f é uma função com domínio S .

Somatórios

Exemplo 27

Seja uma sequência a_k em $a_k = k^2$

$$\sum_{k=3}^{6} a_k = 3^2 + 4^2 + 5^2 + 6^2 = 9 + 16 + 25 + 36 = 86$$

Uma **notação alternativa** para somatórios é

$$\sum_{s \in S} f(s)$$

em que S é um conjunto de domínio e f é uma função com domínio S .

Exemplo 28

$$\sum_{s \in \{0, 3, 7\}} s^2 = 0^2 + 3^2 + 7^2 = 0 + 9 + 49 = 58$$

Variáveis ligadas e livres

A **variável ligada** de um somatório é a variável sob a qual os termos do somatório são definidos. As demais variáveis são chamadas de **variáveis livres**.

Variáveis ligadas e livres

A **variável ligada** de um somatório é a variável sob a qual os termos do somatório são definidos. As demais variáveis são chamadas de **variáveis livres**.

Exemplo 29

$$\sum_{i=1}^n (i - 1)$$

i é a variável ligada e n é a variável livre.

Variáveis ligadas e livres

A **variável ligada** de um somatório é a variável sob a qual os termos do somatório são definidos. As demais variáveis são chamadas de **variáveis livres**.

Exemplo 29

$$\sum_{i=1}^n (i - 1)$$

i é a variável ligada e n é a variável livre.

Exemplo 30

$$\sum_{i=m}^n (i - 1)$$

i é a variável ligada e m e n são variáveis livres.

Mudança de variável

Trocar a variável ligada não altera o valor do somatório.

Trocar a variável ligada não altera o valor do somatório.

Exemplo 31

$$\sum_{i=1}^n \frac{i+1}{i} = \sum_{j=1}^n \frac{j+1}{j} = \sum_{k=1}^n \frac{k+1}{k}$$

Mudança de variável

Trocar a variável ligada não altera o valor do somatório.

Exemplo 31

$$\sum_{i=1}^n \frac{i+1}{i} = \sum_{j=1}^n \frac{j+1}{j} = \sum_{k=1}^n \frac{k+1}{k}$$

Trocar a variável livre altera o valor do somatório.

Mudança de variável

Trocar a variável ligada não altera o valor do somatório.

Exemplo 31

$$\sum_{i=1}^n \frac{i+1}{i} = \sum_{j=1}^n \frac{j+1}{j} = \sum_{k=1}^n \frac{k+1}{k}$$

Trocar a variável livre altera o valor do somatório.

Exemplo 32

Os somatórios são distintos pois $m \neq n$

$$\sum_{i=1}^n \frac{i+1}{i} \neq \sum_{i=1}^m \frac{i+1}{i}$$

Mudança de variável

Dois somatórios são idênticos se e só se eles possuirem termos idênticos

Exemplo 33

Os somatórios são distintos pois $m \neq n$

$$\sum_{j=2}^4 (j-1)^2 = \sum_{i=1}^3 i^2$$

pois

$$\begin{aligned}\sum_{j=2}^4 (j-1)^2 &= (2-1)^2 + (3-1)^2 + (4-1)^2 \\ &= (1)^2 + (2)^2 + (3)^2\end{aligned}$$

e

$$\sum_{i=1}^3 (i)^2 = (1)^2 + (2)^2 + (3)^2$$

Exemplo 34

Substitua $k + 1$ por j

$$\sum_{k=0}^{6} \frac{1}{k+1}$$

Exemplo 34

Substitua $k + 1$ por j

$$\sum_{k=0}^6 \frac{1}{k+1}$$

1. Calcular novos limites

$$k = 0 \Rightarrow j = 0 + 1 = 1$$
$$k = 6 \Rightarrow j = 6 + 1 = 7$$

Mudança de variável

Exemplo 34

Substitua $k + 1$ por j

$$\sum_{k=0}^6 \frac{1}{k+1}$$

1. Calcular novos limites

$$k = 0 \Rightarrow j = 0 + 1 = 1$$
$$k = 6 \Rightarrow j = 6 + 1 = 7$$

2. Calcular o termo geral – como $j = k + 1$ então $k = j - 1$

$$\frac{1}{k+1} = \frac{1}{j-1+1} = \frac{1}{j}$$

Assim,

$$\sum_{k=0}^6 \frac{1}{k+1} = \sum_{j=1}^7 \frac{1}{j}$$

Produtório

Seja uma sequência $\{a_n\}$. O produtório dos termos

$$a_m, a_{m+1}, a_{m+2}, \dots, a_n$$

de $\{a_k\}$ é o produto

$$a_m \times a_{m+1} \times a_{m+2} \times \dots \times a_n$$

O produtório é representado por \prod

$$\prod_{i=m}^n a_i = a_m \times a_{m+1} \times a_{m+2} \times \dots \times a_n$$

Somatórios

Exemplo 35

Seja uma sequência a_k em $a_k = k^2$

$$\prod_{k=3}^6 a_k$$

Somatórios

Exemplo 35

Seja uma sequência a_k em $a_k = k^2$

$$\prod_{k=3}^{6} a_k = 3^2 \times 4^2 \times 5^2 \times 6^2$$

Exemplo 35

Seja uma sequência a_k em $a_k = k^2$

$$\prod_{k=3}^{6} a_k = 3^2 \times 4^2 \times 5^2 \times 6^2 = 9 \times 16 \times 25 \times 36$$

Exemplo 35

Seja uma sequência a_k em $a_k = k^2$

$$\prod_{k=3}^{6} a_k = 3^2 \times 4^2 \times 5^2 \times 6^2 = 9 \times 16 \times 25 \times 36$$

Uma **notação alternativa** para produtório é

$$\prod_{s \in S} f(s)$$

em que S é um conjunto de domínio e f é uma função com domínio S .

Somatórios

Exemplo 35

Seja uma sequência a_k em $a_k = k^2$

$$\prod_{k=3}^{6} a_k = 3^2 \times 4^2 \times 5^2 \times 6^2 = 9 \times 16 \times 25 \times 36$$

Uma notação alternativa para produtório é

$$\prod_{s \in S} f(s)$$

em que S é um conjunto de domínio e f é uma função com domínio S .

Exemplo 36

$$\prod_{s \in \{1,3,7\}} s^2 = 1^2 \times 3^2 \times 7^2 = 1 \times 9 \times 49$$

Algums propriedades

Seja c um número real e sejam $\{a_n\}$ e $\{b_n\}$ duas sequências

$$\begin{aligned} &a_m, a_{m+1}, a_{m+2}, \dots, a_n \\ &b_m, b_{m+1}, b_{m+2}, \dots, b_n \end{aligned}$$

As seguintes propriedades são válidas

1.

$$\sum_{i=n}^m a_i + \sum_{i=n}^m b_i = \sum_{i=n}^m (a_i + b_i)$$

2.

$$c \times \sum_{i=n}^m a_i = \sum_{i=n}^m (c \times a_i)$$

3.

$$\prod_{i=n}^m a_i \times \prod_{i=n}^m b_i = \prod_{i=n}^m (a_i \times b_i)$$

Questions?

Teoria de Conjuntos e Funções
– Somatórios e produtórios –



Teoria dos Grafos e Computabilidade

— Data structures for graphs —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Incidence matrix —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Matriz de incidência nó-arco

Seja um grafo $G = (V, A)$ em que $|V| = n$ e $|A| = m$. Uma matriz de incidência $A_{n \times m}$ nó-arco é representada por:

- ▶ Uma linha para cada nó
- ▶ Uma coluna para cada aresta

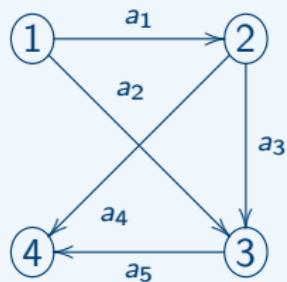
$$a = (i, j) \in A \Rightarrow [\begin{array}{ccccc} 0 & +1 & 0 & -1 & 0 \end{array}]^T$$

Matriz de incidência nó-arco

Seja um grafo $G = (V, A)$ em que $|V| = n$ e $|A| = m$. Uma matriz de incidência $A_{n \times m}$ nó-arco é representada por:

- Uma linha para cada nó
- Uma coluna para cada aresta

$$a = (i, j) \in A \Rightarrow [0 \quad +1 \quad 0 \quad -1 \quad 0]^T$$



$$A_{n \times m} = \begin{bmatrix} +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & +1 & +1 & 0 \\ 0 & -1 & -1 & 0 & +1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

Questions?

Data structures for graphs
– Incidence matrix –



Teoria dos Grafos e Computabilidade

— Adjacency matrix —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Matriz de adjacência

Seja um grafo $G = (V, A)$ em que $|V| = n$ e $|A| = m$. Uma matriz de adjacência $A_{n \times n}$ é representada por:

- ▶ Uma linha para cada nó
- ▶ Uma coluna para cada nó

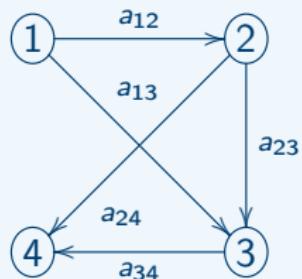
$$a_{ij} = \begin{cases} 1, & (i,j) \in A \\ 0, & (i,j) \notin A \end{cases}$$

Matriz de adjacência

Seja um grafo $G = (V, A)$ em que $|V| = n$ e $|A| = m$. Uma matriz de adjacência $A_{n \times n}$ é representada por:

- Uma linha para cada nó
- Uma coluna para cada nó

$$a_{ij} = \begin{cases} 1, & (i, j) \in A \\ 0, & (i, j) \notin A \end{cases}$$



$$A_{n \times n} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Questions?

Data structures for graphs
– Adjacency matrix –

Teoria dos Grafos e Computabilidade

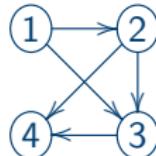
— Adjacency list —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Lista de adjacência

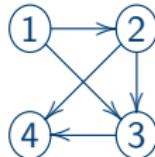
Seja um grafo $G = (V, A)$ em que $|V| = n$ e $|A| = m$. Uma lista de adjacência $A_{n \times n}$ é representada por uma lista de nós (ou vértices) em que cada nó aponta para a lista de seus sucessores (ou nós adjacentes).



$$\Rightarrow n \text{ vértices} + m \text{ arestas} = \\ m + n \text{ elementos na lista}$$

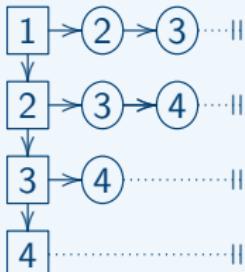
Lista de adjacência

Seja um grafo $G = (V, A)$ em que $|V| = n$ e $|A| = m$. Uma lista de adjacência $A_{n \times n}$ é representada por uma lista de nós (ou vértices) em que cada nó aponta para a lista de seus sucessores (ou nós adjacentes).

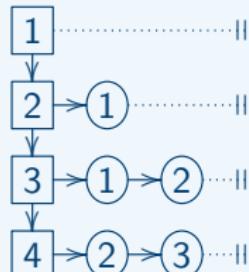


$\Rightarrow n$ vértices + m arestas =
 $m + n$ elementos na lista

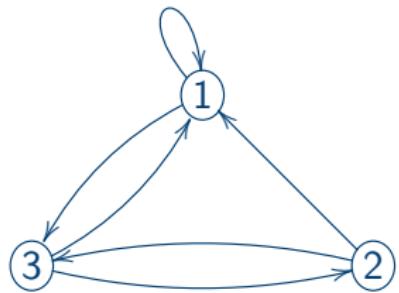
SUCESSORES



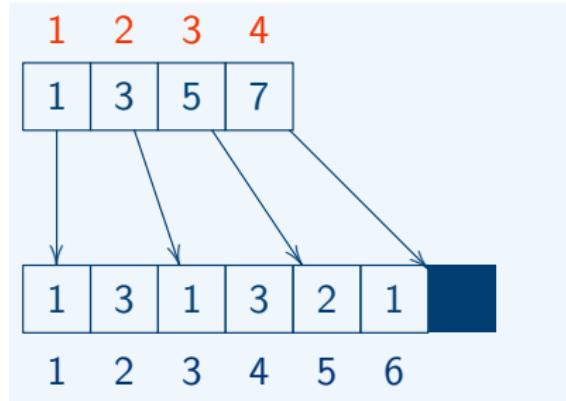
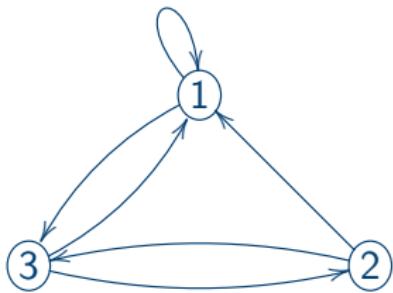
PREDECESSORES



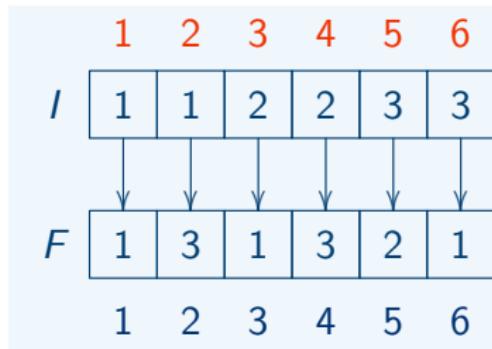
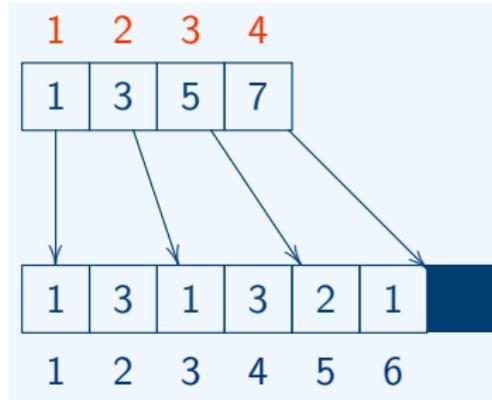
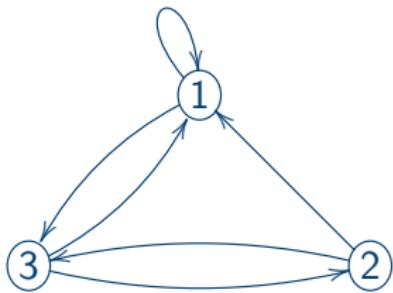
Lista de adjacência



Lista de adjacência



Lista de adjacência



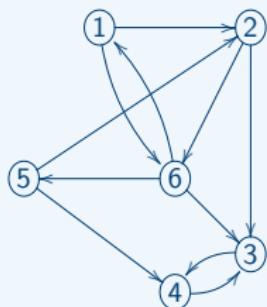
Monte o grafo a partir da representação

Exemplo 1

$$A_{n \times n} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

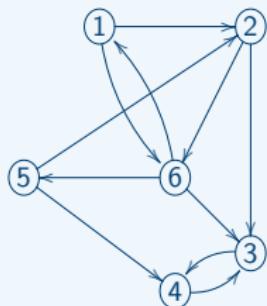
Exemplo 1

$$A_{n \times n} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$



Exemplo 1

$$A_{n \times n} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$



MATRIZ DE INCIDÊNCIA

$$\begin{bmatrix} +1 & +1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & +1 & 0 & +1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & +1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & +1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & +1 & +1 & 0 & 0 \\ 0 & -1 & +1 & +1 & -1 & +1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Questions?

Data structures for graphs
– Adjacency list –



Teoria dos Grafos e Computabilidade

— Isomorphism and some concepts —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas



Teoria dos Grafos e Computabilidade

— Isomorphism —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

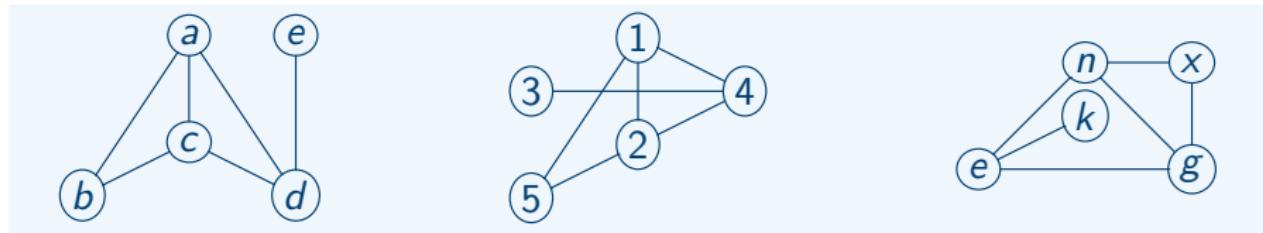
Pontifical Catholic University of Minas Gerais – PUC Minas

Isomorfismo

Dois grafos G e H são ditos **isomorfos** se existir uma correspondência um-para-um entre seus vértices e entre suas arestas, de maneira que as relações de incidência são preservadas

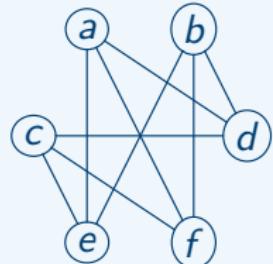
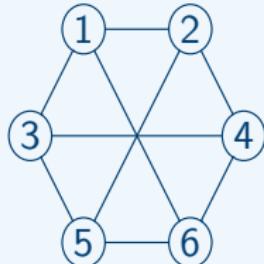
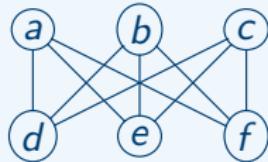
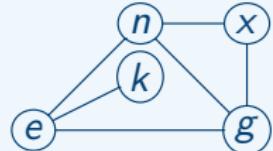
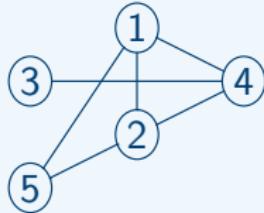
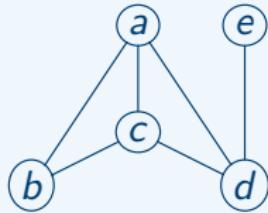
Isomorfismo

Dois grafos G e H são ditos **isomorfos** se existir uma correspondência um-para-um entre seus vértices e entre suas arestas, de maneira que as relações de incidência são preservadas



Isomorfismo

Dois grafos G e H são ditos **isomorfos** se existir uma correspondência um-para-um entre seus vértices e entre suas arestas, de maneira que as relações de incidência são preservadas



Isomorfismo

Condições necessárias mas não suficientes para que G e H sejam isomorfos:

- ▶ mesmo número de vértices
- ▶ mesmo número de arestas
- ▶ mesmo número de componentes
- ▶ mesmo número de vértices com o mesmo grau

Isomorfismo

Condições necessárias mas não suficientes para que G e H sejam isomorfos:

- ▶ mesmo número de **vértices**
- ▶ mesmo número de **arestas**
- ▶ mesmo número de **componentes**
- ▶ mesmo número de **vértices com o mesmo grau**



Isomorfismo

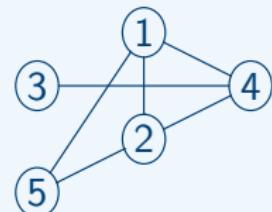
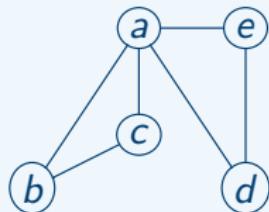
Condições necessárias mas não suficientes para que G e H sejam isomorfos:

- mesmo número de vértices
- mesmo número de arestas
- mesmo número de componentes
- mesmo número de vértices com o mesmo grau



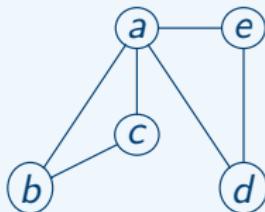
Não existe um algoritmo eficiente para determinar se dois grafos são isomorfos

Some examples

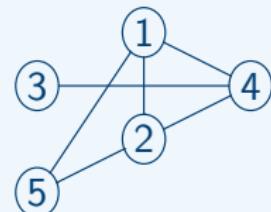


Are these two graphs Isomorphic?

Some examples

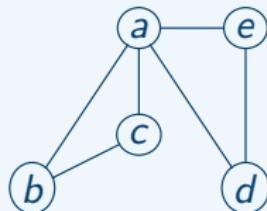


► vertices \Rightarrow 5

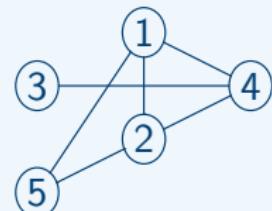


► vertices \Rightarrow 5

Some examples

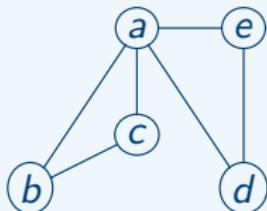


- ▶ vertices \Rightarrow 5
- ▶ edges \Rightarrow 6

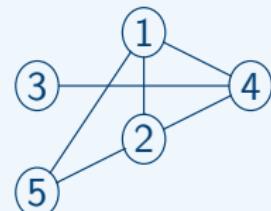


- ▶ vertices \Rightarrow 5
- ▶ edges \Rightarrow 6

Some examples

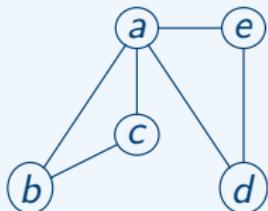


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$

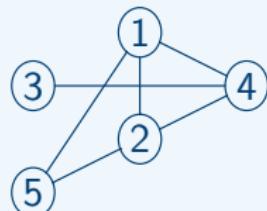


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$

Some examples

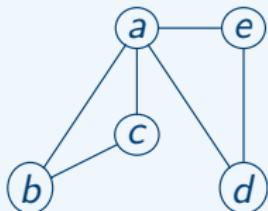


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 2 \ 2 \ 2 \ 3 \ 4$

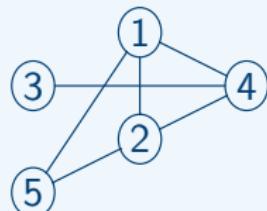


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1 \ 2 \ 3 \ 3 \ 3$

Some examples



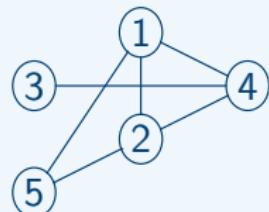
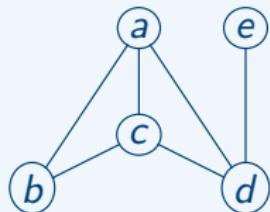
- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 2\ 2\ 2\ 3\ 4$



- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 2\ 3\ 3\ 3$

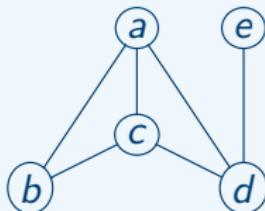
These two graphs are NOT Isomorphic

Some examples

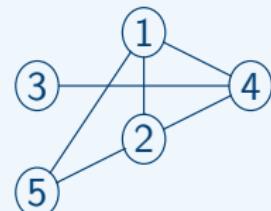


Are these two graphs Isomorphic?

Some examples

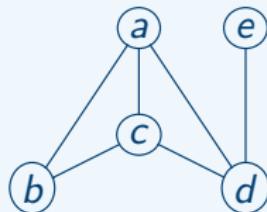


► vertices \implies 5

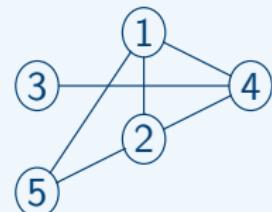


► vertices \implies 5

Some examples

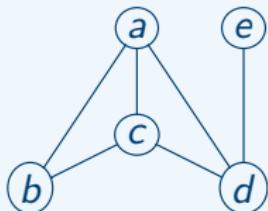


- ▶ vertices \Rightarrow 5
- ▶ edges \Rightarrow 6

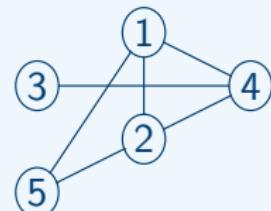


- ▶ vertices \Rightarrow 5
- ▶ edges \Rightarrow 6

Some examples

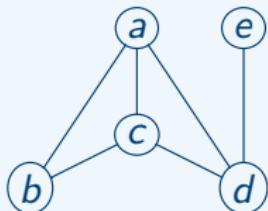


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$

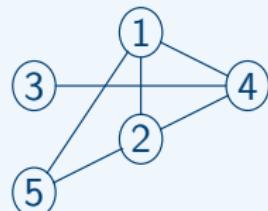


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$

Some examples

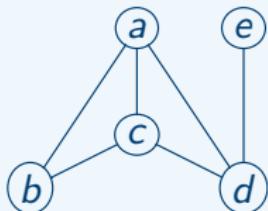


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 2\ 3\ 3\ 3$

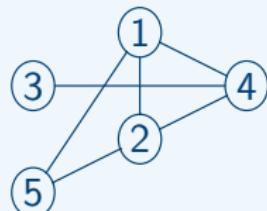


- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 2\ 3\ 3\ 3$

Some examples



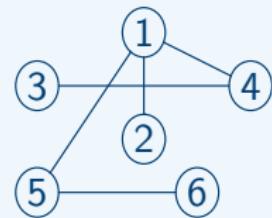
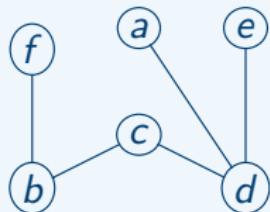
- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 2\ 3\ 3\ 3$



- ▶ vertices $\Rightarrow 5$
- ▶ edges $\Rightarrow 6$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 2\ 3\ 3\ 3$

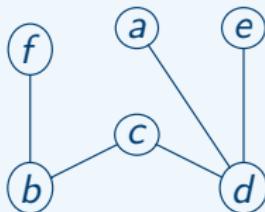
These two graphs are Isomorphic

Some examples

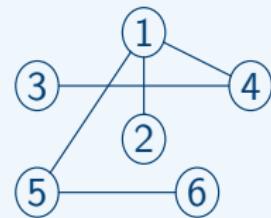


Are these two graphs Isomorphic?

Some examples

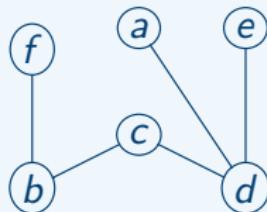


► vertices \Rightarrow 6

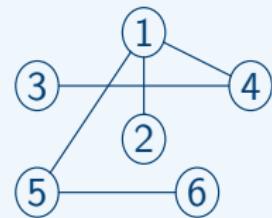


► vertices \Rightarrow 6

Some examples

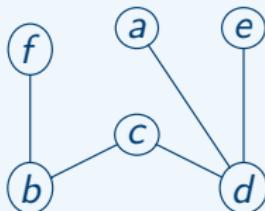


- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$

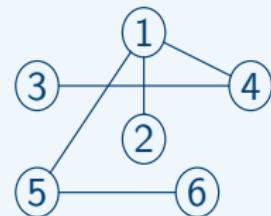


- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$

Some examples

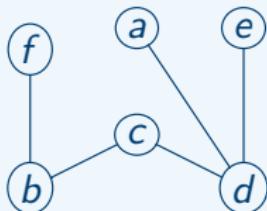


- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$

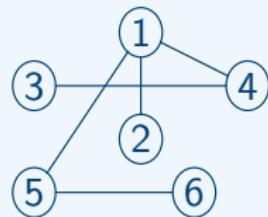


- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$

Some examples

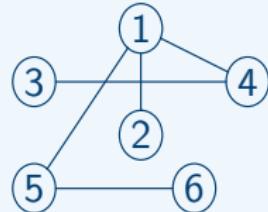
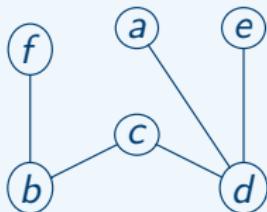


- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 1\ 1\ 2\ 3$



- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 1\ 1\ 2\ 3$

Some examples

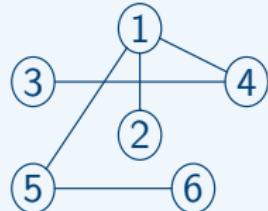
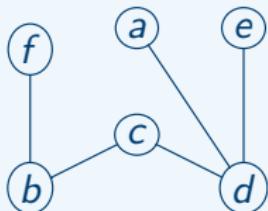


- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 1\ 1\ 2\ 3$

- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 1\ 1\ 2\ 3$

THESE TWO GRAPHS ARE NOT ISOMORPHIC.

Some examples



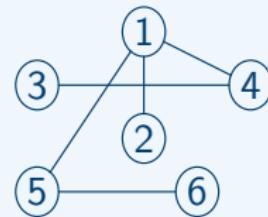
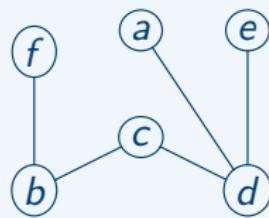
- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 1\ 1\ 2\ 3$

- ▶ vertices $\Rightarrow 6$
- ▶ edges $\Rightarrow 5$
- ▶ components $\Rightarrow 1$
- ▶ degrees $\Rightarrow 1\ 1\ 1\ 2\ 3$

THESE TWO GRAPHS ARE NOT ISOMORPHIC. WHY?

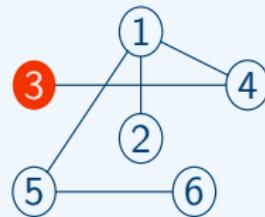
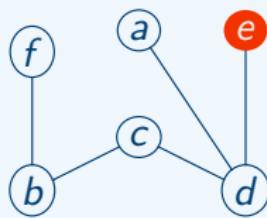
Some examples

THE PROBLEM IS RELATED TO THE RELATIONSHIP
BETWEEN THE VERTICES!!!



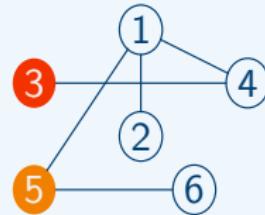
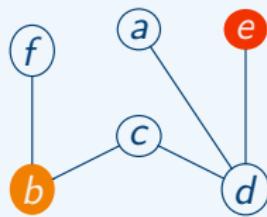
Some examples

THE PROBLEM IS RELATED TO THE RELATIONSHIP
BETWEEN THE VERTICES!!!



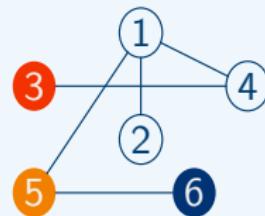
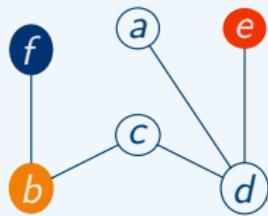
Some examples

THE PROBLEM IS RELATED TO THE RELATIONSHIP
BETWEEN THE VERTICES!!!



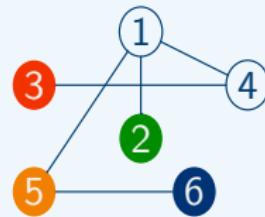
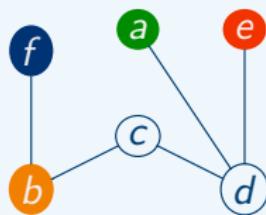
Some examples

THE PROBLEM IS RELATED TO THE RELATIONSHIP
BETWEEN THE VERTICES!!!



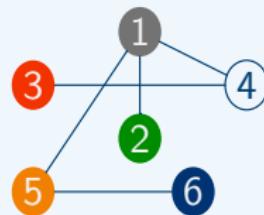
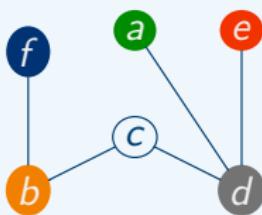
Some examples

THE PROBLEM IS RELATED TO THE RELATIONSHIP
BETWEEN THE VERTICES!!!



Some examples

THE PROBLEM IS RELATED TO THE RELATIONSHIP
BETWEEN THE VERTICES!!!



The gray vertices (1 and d) are adjacent to vertices with different colors

Questions?

Isomorphism and some concepts
– Isomorphism –



Teoria dos Grafos e Computabilidade

— Important concepts —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Grafo complementar

Seja $G = (V, E)$ um grafo simples dirigido ou não-dirigido. O **grafo complementar** de G , denotado por $C(G)$ ou \overline{G} , é um grafo formado da seguinte maneira:

- ▶ Os vértices de $C(G)$ são todos os vértices de G
- ▶ As arestas de $C(G)$ são exatamente as arestas que faltam em G para formarmos um grafo completo

Grafo complementar

Seja $G = (V, E)$ um grafo simples dirigido ou não-dirigido. O **grafo complementar** de G , denotado por $C(G)$ ou \overline{G} , é um grafo formado da seguinte maneira:

- ▶ Os vértices de $C(G)$ são todos os vértices de G
- ▶ As arestas de $C(G)$ são exatamente as arestas que faltam em G para formarmos um grafo completo

Exemplo 1

- ▶ Encontre um grafo com 5 vértices que seja isomorfo a seu complemento.

Grafo complementar

Seja $G = (V, E)$ um grafo simples dirigido ou não-dirigido. O **grafo complementar** de G , denotado por $C(G)$ ou \overline{G} , é um grafo formado da seguinte maneira:

- ▶ Os vértices de $C(G)$ são todos os vértices de G
- ▶ As arestas de $C(G)$ são exatamente as arestas que faltam em G para formarmos um grafo completo

Exemplo 1

- ▶ Encontre um grafo com 5 vértices que seja isomorfo a seu complemento.
- ▶ Qual o número de arestas de um grafo que é isomorfo a seu complemento?

Subgrafo

Um grafo $G_1 = (V_1, A_1)$ é dito ser **subgrafo** de um grafo $G = (V, A)$ quando $V_1 \subset V$ e $A_1 \subset A$.

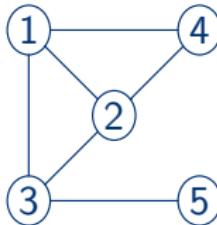
Se $G_2 = (V_2, A_2)$ é um subgrafo de $G_1 = (V_1, A_1)$ e possui toda aresta (v, w) de G_1 tal que ambos, v e w , estejam em V_2 , então G_2 é o **subgrafo induzido** pelo subconjunto de vértices V_2 .

Subgrafo

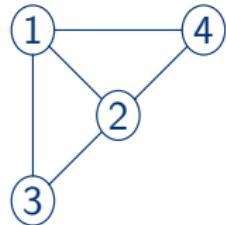
Um grafo $G_1 = (V_1, A_1)$ é dito ser **subgrafo** de um grafo $G = (V, A)$ quando $V_1 \subset V$ e $A_1 \subset A$.

Se $G_2 = (V_2, A_2)$ é um subgrafo de $G_1 = (V_1, A_1)$ e possui toda aresta (v, w) de G_1 tal que ambos, v e w , estejam em V_2 , então G_2 é o **subgrafo induzido** pelo subconjunto de vértices V_2 .

Exemplo 2



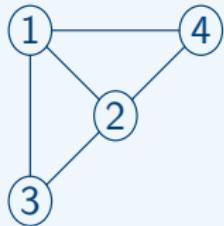
subgrafo
induzido por $\{1, 2, 3, 4\}$



Subgrafo

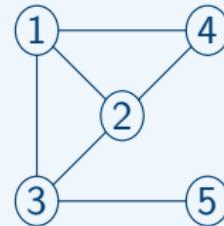
- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de H estão em G

H



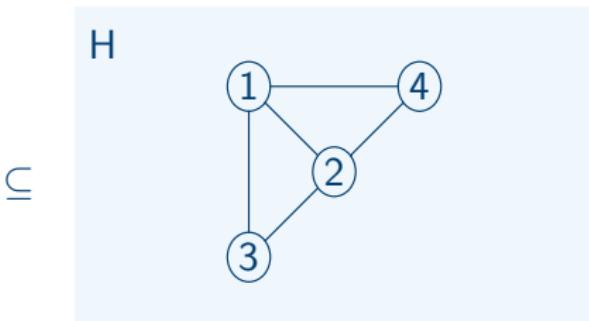
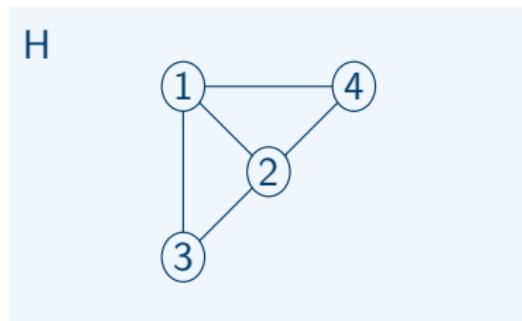
\subseteq

G



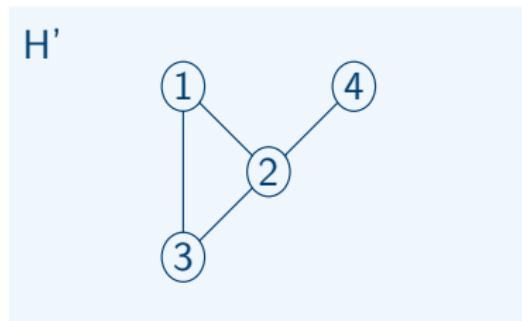
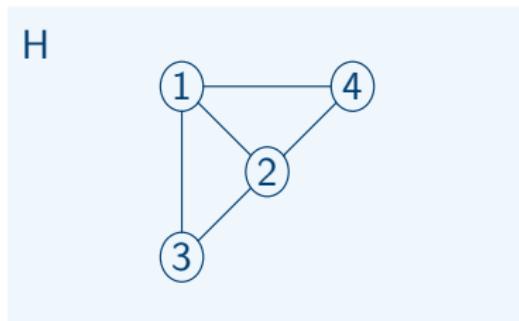
Subgrafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio



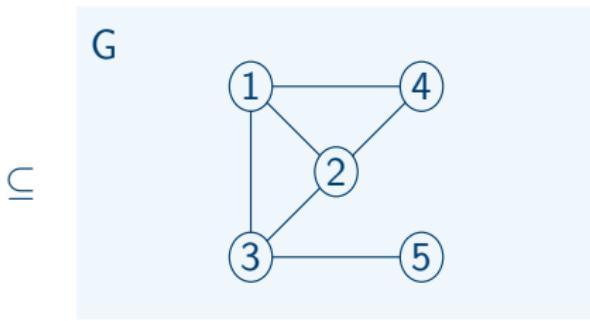
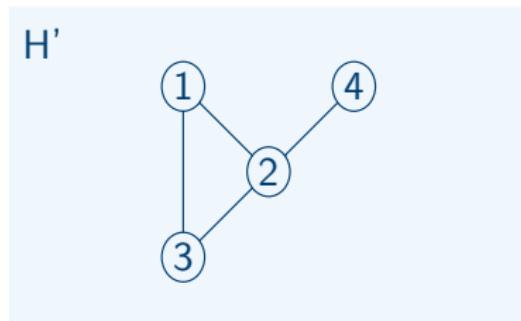
Subgrafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G

 \subseteq 

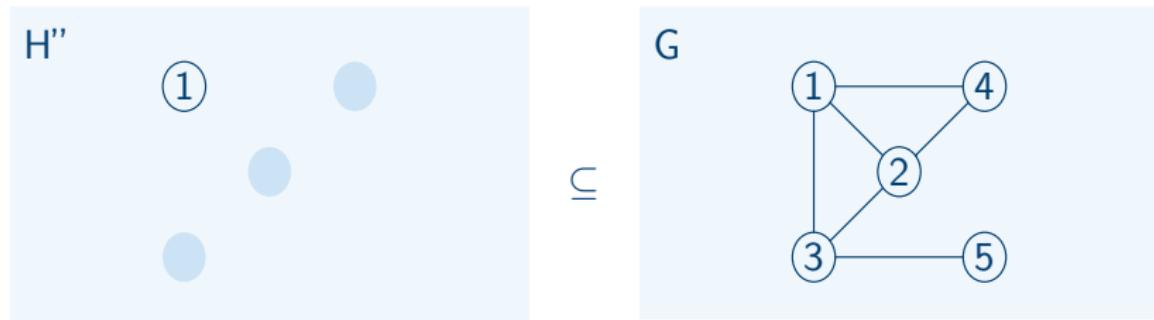
Subgrafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G



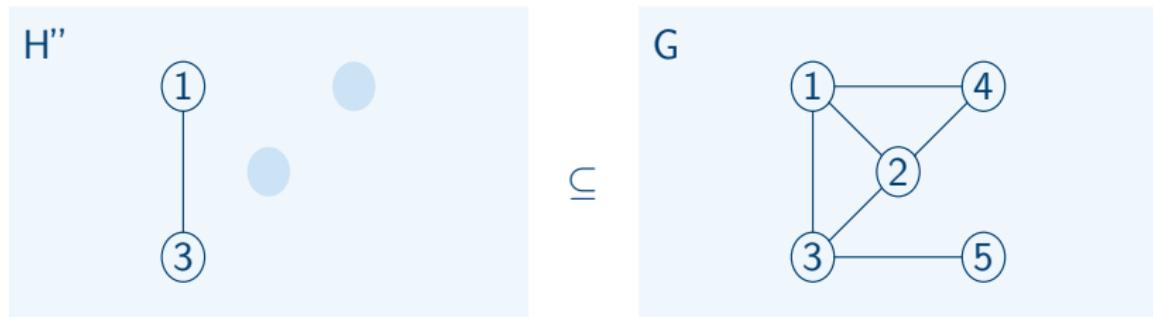
Subgrafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G
 - um vértice simples de G é um subgrafo de G



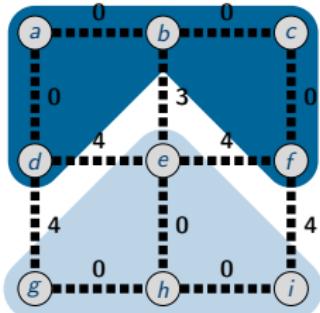
Subgrafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G
 - um vértice simples de G é um subgrafo de G
 - uma aresta simples de G (juntamente com suas extremidades) é subgrafo de G



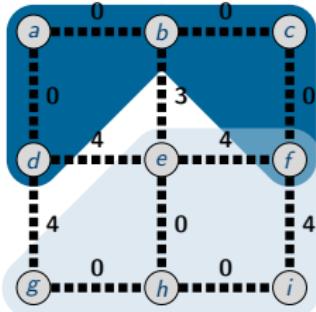
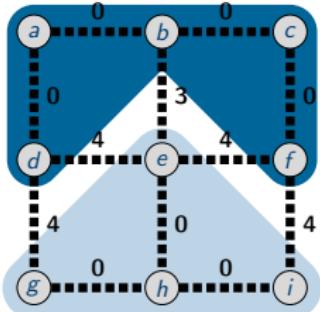
Subgrafo

Subgrafos disjuntos de arestas *dois (ou mais) subgrafos G_1 e G_2 de um grafo G são disjuntos de arestas se G_1 e G_2 não tiverem nenhuma aresta em comum.*



Subgrafo

Subgrafos disjuntos de arestas *dois (ou mais) subgrafos G_1 e G_2 de um grafo G são disjuntos de arestas se G_1 e G_2 não tiverem nenhuma aresta em comum.*
⇒ G_1 e G_2 podem ter vértices em comum?

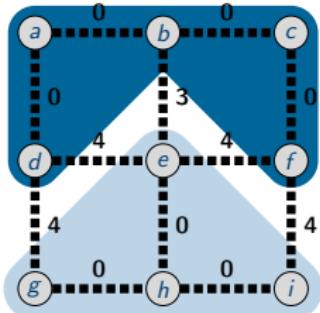


Subgrafo

Subgrafos disjuntos de arestas *dois (ou mais) subgrafos G_1 e G_2 de um grafo G são disjuntos de arestas se G_1 e G_2 não tiverem nenhuma aresta em comum.*

Subgrafos disjuntos de vértices *dois (ou mais) subgrafos G_1 e G_2 de um grafo G são disjuntos de vértices se G_1 e G_2 não tiverem nenhum vértice em comum.*

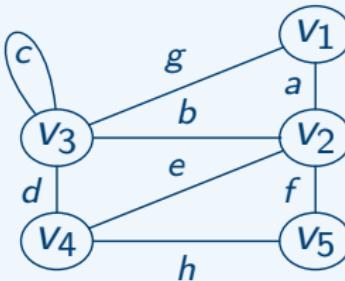
⇒ G_1 e G_2 podem ter arestas em comum?



Caminhos e circuitos

Seqüência de arestas seqüência alternada de vértices e arestas começando e terminando com vértice. Cada aresta é incidente ao vértice que a precede e a antecede

Ex.: v_1 a v_2 a v_1 g v_3



Caminhos e circuitos

Seqüência de arestas seqüência alternada de vértices e arestas começando e terminando com vértice. Cada aresta é incidente ao vértice que a precede e a antecede

Ex.: v_1 a v_2 a v_1 g v_3

Caminho seqüência de arestas no qual nenhuma aresta aparece mais de uma vez

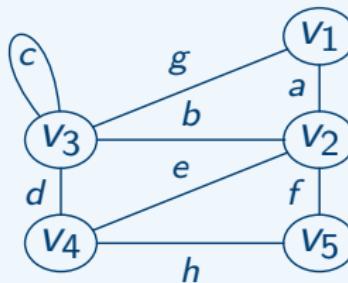
Ex.: v_1 a v_2 b v_3 c v_3 d v_4 e v_2 f v_5

- Caminho aberto: vértice inicial é diferente do vértice final

Ex.: v_1 a v_2 b v_3 c v_3

- Caminho fechado: caminhos que começam e terminam no mesmo vértice

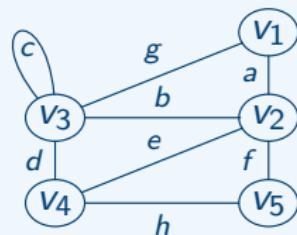
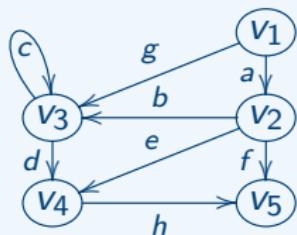
Ex.: v_1 a v_2 b v_3 c v_3 g v_1



Seja G um grafo dirigido e G' o seu grafo não-dirigido associado. Uma cadeia em G é um caminho em G' .

Cadeias

Seja G um grafo dirigido e G' o seu grafo não-dirigido associado. Uma cadeia em G é um caminho em G' .



$g-a-f$ é um caminho de G' e uma cadeia em G

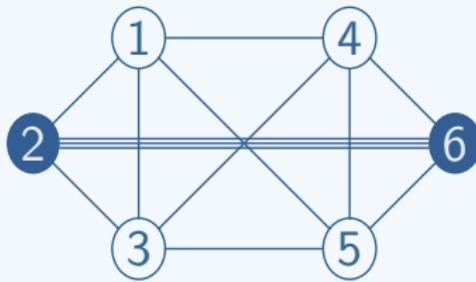
TEOREMA

Se um grafo possui exatamente 2 vértices de grau ímpar, existe uma aresta entre esses dois vértices

Caminhos e circuitos

TEOREMA

Se um grafo possui exatamente 2 vértices de grau ímpar, existe uma aresta entre esses dois vértices

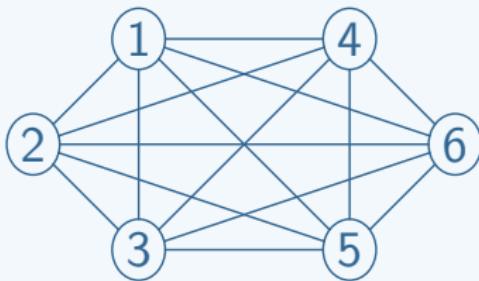


2–6

Teorema Um grafo simples com n vértices e k componentes possui no máximo $(n - k)(n - k + 1)/2$ arestas

Caminhos e circuitos

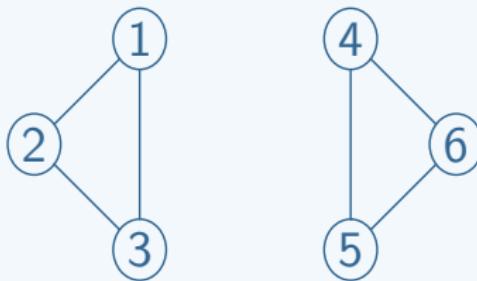
Teorema Um grafo simples com n vértices e k componentes possui no máximo $(n - k)(n - k + 1)/2$ arestas



$$k = 1, n = 6 \implies e = 15$$

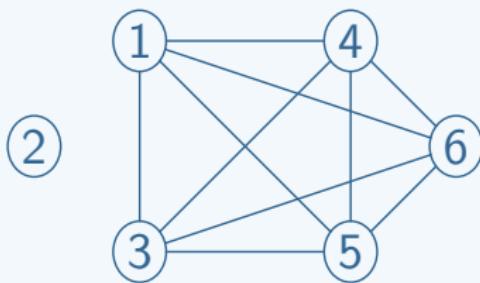
Caminhos e circuitos

Teorema Um grafo simples com n vértices e k componentes possui no máximo $(n - k)(n - k + 1)/2$ arestas



$$k = 2, n = 6 \implies e = 6$$

Teorema Um grafo simples com n vértices e k componentes possui no máximo $(n - k)(n - k + 1)/2$ arestas



$$k = 2, n = 6 \implies e = 10$$

Teorema O número mínimo de arestas de um grafo simples com n vértices e k componentes é $n - k$

Caminhos e circuitos

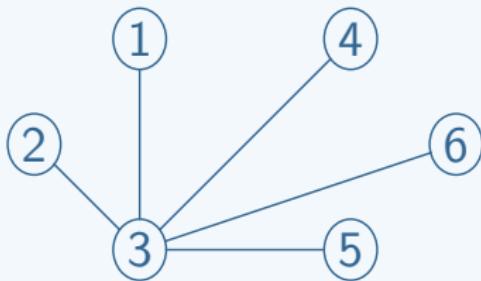
Teorema O número mínimo de arestas de um grafo simples com n vértices e k componentes é $n - k$



$$k = 6, n = 6 \implies e = 0$$

Caminhos e circuitos

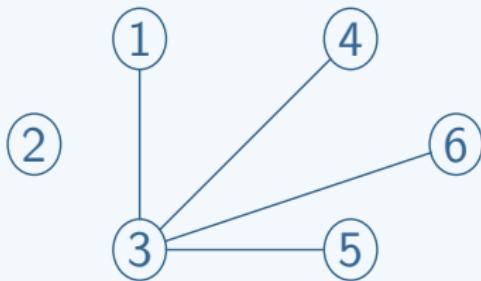
Teorema O número mínimo de arestas de um grafo simples com n vértices e k componentes é $n - k$



$$k = 1, n = 6 \implies e = 5$$

Caminhos e circuitos

Teorema O número mínimo de arestas de um grafo simples com n vértices e k componentes é $n - k$



$$k = 2, n = 6 \implies e = 4$$

Questions?

Isomorphism and some concepts
– Important concepts –



Teoria dos Grafos e Computabilidade

— Lógica Proposicional —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas



Teoria dos Grafos e Computabilidade

— Princípios da Lógica Proposicional —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Princípios da Lógica Proposicional

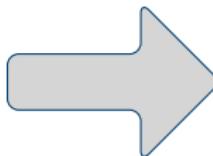
Lógica *Ramo da Filosofia, Matemática e Ciência da Computação que trata das inferências válidas.*

Princípios da Lógica Proposicional

Lógica *Ramo da Filosofia, Matemática e Ciência da Computação que trata das inferências válidas.*

A lógica estuda a preservação da verdade durante uma argumentação .

Hipóteses
verdadeiras



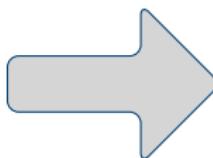
Conclusões
verdadeiras

Princípios da Lógica Proposicional

Lógica *Ramo da Filosofia, Matemática e Ciência da Computação que trata das inferências válidas.*

A lógica estuda a preservação da verdade durante uma argumentação .

Hipóteses
verdadeiras



Conclusões
verdadeiras

As regras da lógicas são essenciais na construção de provas matemáticas, pois dão significados às afirmações matemáticas.

Proposições Lógicas

Asserção uma declaração (afirmação, sentença declarativa).

Proposição uma asserção que é verdadeira (V) ou falsa (F), mas não ambos.

Valor verdade resultado da avaliação de uma proposição (V ou F).

Proposições Lógicas

Asserção uma declaração (afirmação, sentença declarativa).

Proposição uma asserção que é verdadeira (V) ou falsa (F), mas não ambos.

Valor verdade resultado da avaliação de uma proposição (V ou F).

- ▷ $2 + 3 = 5$
- ▷ 3 não é um número ímpar
- ▷ A Terra é arredondada
- ▷ $x > 5$
- ▷ Esta declaração é falsa
- ▷ Você fala francês?
- ▷ Paris é a cidade mais linda?

Proposições Lógicas

Asserção uma declaração (afirmação, sentença declarativa).

- ▷ $2 + 3 = 5$ (asserção)
- ▷ 3 não é um número ímpar (asserção)
- ▷ A Terra é arredondada (asserção)
- ▷ $x > 5$ (asserção)
- ▷ Esta declaração é falsa (asserção)

Proposições Lógicas

Proposição uma asserção que é verdadeira (V) ou falsa (F), mas não ambos.

- ▷ $2 + 3 = 5$ (proposição)
- ▷ 3 não é um número ímpar (proposição)
- ▷ A Terra é arredondada (proposição)

Proposições Lógicas

Valor verdade *resultado da avaliação de uma proposição (V ou F).*

- ▷ $2 + 3 = 5$ (V)
- ▷ 3 não é um número ímpar (F)
- ▷ A Terra é arredondada (V)

Proposições Lógicas

Asserção uma declaração (afirmação, sentença declarativa).

Proposição uma asserção que é verdadeira (V) ou falsa (F), mas não ambos.

Valor verdade resultado da avaliação de uma proposição (V ou F).

- | | |
|--------------------------------|----------------------------------|
| ▷ $2 + 3 = 5$ | (asserção, proposição, V) |
| ▷ 3 não é um número ímpar | (asserção, proposição, F) |
| ▷ A Terra é arredondada | (asserção, proposição, V) |
| ▷ $x > 5$ | (asserção, mas não é proposição) |
| ▷ Esta declaração é falsa | (asserção, mas não é proposição) |
| ▷ Você fala francês? | (nem asserção, nem proposição) |
| ▷ Paris é a cidade mais linda? | (nem asserção, nem proposição) |

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- Belo Horizonte é a capital de Minas Gerais

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- Roma é a capital da França

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- Roma é a capital da França (proposição falsa)

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$ (proposição verdadeira)

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$ (proposição verdadeira)
- ▶ $1 + 1 = 3$

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$ (proposição verdadeira)
- ▶ $1 + 1 = 3$ (proposição falsa)

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$ (proposição verdadeira)
- ▶ $1 + 1 = 3$ (proposição falsa)

SENTENÇAS – NÃO SÃO PROPOSIÇÕES

- ▶ Que horas são?

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$ (proposição verdadeira)
- ▶ $1 + 1 = 3$ (proposição falsa)

SENTENÇAS – NÃO SÃO PROPOSIÇÕES

- ▶ Que horas são? (não é uma sentença declarativa)

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$ (proposição verdadeira)
- ▶ $1 + 1 = 3$ (proposição falsa)

SENTENÇAS – NÃO SÃO PROPOSIÇÕES

- ▶ Que horas são? (não é uma sentença declarativa)
- ▶ $x + 1 = 4$

Proposição

Uma proposição é uma sentença declarativa (uma sentença que estabelece um fato) que pode ser verdadeira ou falsa, mas não ambos.

SENTENÇAS DECLARATIVAS – PROPOSIÇÕES

- ▶ Belo Horizonte é a capital de Minas Gerais (proposição verdadeira)
- ▶ Roma é a capital da França (proposição falsa)
- ▶ $1 + 1 = 2$ (proposição verdadeira)
- ▶ $1 + 1 = 3$ (proposição falsa)

SENTENÇAS – NÃO SÃO PROPOSIÇÕES

- ▶ Que horas são? (não é uma sentença declarativa)
- ▶ $x + 1 = 4$ (não é verdadeiro nem falso)

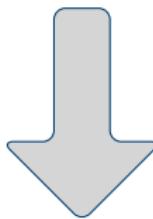
Variáveis proposicionais *Em Lógica, as proposições podem ser denotadas por símbolos, tais como p, q, r, \dots , os quais são chamados de variáveis proposicionais.*

EXEMPLOS

- p : o Sol está brilhando hoje.
- q : $2 + 3 = 5$
- t : Belo Horizonte é a capital de Minas Gerais
- u : São Paulo é a capital do Brasil

Proposições Compostas

Novas proposições podem ser construídas a partir de proposições existentes



Obtenção de
proposições compostas

Proposições Compostas

Novas proposições podem ser construídas a partir de proposições existentes



Obtenção de
proposições compostas

Tabela Verdade

Negação A sentença: “Não é verdade que p ”

- ▶ é uma outra proposição
- ▶ chamada de a negação de p .
- ▶ Notação: $\neg p$, $\sim p$, *not p*

Tabela Verdade

Negação A sentença: “*Não é verdade que p*”

- ▶ é uma outra proposição
- ▶ chamada de a negação de p .
- ▶ Notação: $\neg p$, $\sim p$, *not p*

EXEMPLOS

- ▶ $p : 2 + 3 > 1$
 $\neg p : 2 + 3$ não é maior do que 1, (ou $2 + 3 \leq 1$)
- ▶ $q : \text{“Hoje é quarta-feira”}$
 $\neg q : \text{“Não é verdade que hoje é quarta-feira”, ou}$
 $\neg q : \text{“Hoje não é quarta-feira”}$

Tabela Verdade

Negação A sentença: “Não é verdade que p ”

- ▶ é uma outra proposição
- ▶ chamada de a negação de p .
- ▶ Notação: $\neg p$, $\sim p$, not p

A PARTIR DA DEFINIÇÃO

- ▶ se p é Verdadeiro, então $\neg p$ é Falso
- ▶ se p é Falso, então $\neg p$ é Verdadeiro

Tabela Verdade

Negação A sentença: “Não é verdade que p ”

- é uma outra proposição
- chamada de a negação de p .
- Notação: $\neg p$, $\sim p$, not p

A PARTIR DA DEFINIÇÃO

- se p é Verdadeiro, então $\neg p$ é Falso
- se p é Falso, então $\neg p$ é Verdadeiro

Tabela verdade da negação

p	$\neg p$
V	F
F	V

Tabela Verdade

TABELA VERDADE

Fornece os valores verdade de uma proposição composta em termos dos valores verdade de suas partes componentes.

determinação dos valores verdade de proposições construídas a partir de sentenças mais simples.

Questions?

Lógica Proposicional
– Princípios da Lógica
Proposicional –

Teoria dos Grafos e Computabilidade

— Conectivos Lógicos —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Conektivos Lógicos

Operador negação *constrói uma nova proposição a partir de uma única proposição existente.*

Conektivos *operadores lógicos usados para formar novas proposições a partir de duas ou mais proposições já existentes.*

Conectivos Lógicos

Operador negação constrói uma nova proposição a partir de uma única proposição existente.

Conectivos operadores lógicos usados para formar novas proposições a partir de duas ou mais proposições já existentes.

Conjunção (operação “e”):

- ▶ Notação: $p \wedge q$, p e q ,
 p and q
- ▶ Definição:

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Conectivos Lógicos

Operador negação constrói uma nova proposição a partir de uma única proposição existente.

Conectivos operadores lógicos usados para formar novas proposições a partir de duas ou mais proposições já existentes.

Conjunção (operação “e”):

- ▶ Notação: $p \wedge q$, p e q ,
 p and q
- ▶ Definição:

p	q	$p \wedge q$
V	V	V
V	F	F
F	V	F
F	F	F

Disjunção (operação “ou inclusivo”):

- ▶ Notação: $p \vee q$, p ou q , p or q
- ▶ Definição:

p	q	$p \vee q$
V	V	V
V	F	V
F	V	V
F	F	F

EXEMPLOS DE CONJUNÇÃO ($p \wedge q$)

- p : hoje é terça-feira
 q : está chovendo hoje
 $p \wedge q$: hoje é terça-feira e está chovendo hoje
- p : $2 < 3$
 q : $-5 > -8$
 $p \wedge q$: $2 < 3$ e $-5 > -8$

Principais Conectivos Lógicos

EXEMPLOS DE CONJUNÇÃO ($p \wedge q$)

- p : hoje é terça-feira
 q : está chovendo hoje
 $p \wedge q$: hoje é terça-feira e está chovendo hoje
- p : $2 < 3$
 q : $-5 > -8$
 $p \wedge q$: $2 < 3$ e $-5 > -8$

EXEMPLOS DE DISJUNÇÃO ($p \vee q$)

- p : 2 é um inteiro positivo
 q : $\sqrt{2}$ é um número racional
 $p \vee q$: 2 é um inteiro positivo ou $\sqrt{2}$ é um número racional
- p : $2 + 3 \neq 5$
 q : Belo Horizonte é a capital do Rio de Janeiro
 $p \vee q$: $2 + 3 \neq 5$ ou Belo Horizonte é a capital do Rio de Janeiro

Principais Conectivos Lógicos

DISJUNÇÃO EXCLUSIVA (OPERAÇÃO “XOR”)

- ▶ Notação: $p \oplus q$, p xor q , p ou q (mas não ambos)
- ▶ Definição:

p	q	$p \oplus q$
V	V	F
V	F	V
F	V	V
F	F	F

- ▶ V quando exatamente um dos dois é V

Principais Conectivos Lógicos

CONDICIONAL OU IMPLICAÇÃO (SE p , ENTÃO q)

- Notação: $p \rightarrow q$
- Definição:

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

- V quando:
 - p e q são ambos V
 - p é F (não importando q)

O Condicional

Sejam p e q duas proposições.

A afirmação condicional ou implicação $p \rightarrow q$ e a afirmação
se p , então q

- p é chamada de hipótese, antecedente, ou premissa,
- q é chamada de conclusão ou consequente.

O Condicional

Sejam p e q duas proposições.

A afirmação condicional ou implicação $p \rightarrow q$ e a afirmação
se p , então q

- p é chamada de hipótese, antecedente, ou premissa,
- q é chamada de conclusão ou consequente.

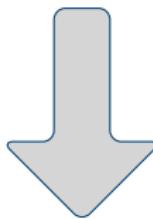
FORMAS DE
EXPRESSAR

- se p , então q
- p é condição suficiente para q
- q é condição necessária para p
- p somente se q
- q é conseqüência lógica de p

O Condicional

EXEMPLO

“Fogo é uma condição necessária para fumaça”



“Se há fumaça, então há fogo”

- ▶ o antecedente (ou hipótese) é: “Há fumaça”
- ▶ o conseqüente (ou conclusão) é: “Há fogo”

INDIQUE O ANTECEDENTE E O CONSEQÜENTE

- ▶ “Se a chuva continuar, o rio vai transbordar”.
- ▶ “Uma condição suficiente para a falha de uma rede é que a chave geral páre de funcionar”.
- ▶ “Os abacates só estão maduros quando estão escuros e macios”.

Proposição condicional

A implicação $p \rightarrow q$ pode ser entendida como uma promessa:

Se você me garantir p , eu te garanto q .

Quebra da promessa *A promessa só é quebrada quando você me garantir p e eu não te garantir q em troca.*

Mantida *A promessa é mantida quando você me garante p e eu te garanto q , ou quando você não me garante p (e neste caso eu sou livre para te garantir q ou não sem quebrar a promessa).*

Proposição condicional

A implicação $p \rightarrow q$ pode ser entendida como uma promessa:

Se você me garantir p , eu te garanto q .

Quebra da promessa *A promessa só é quebrada quando você me garantir p e eu não te garantir q em troca.*

Mantida *A promessa é mantida quando você me garante p e eu te garanto q , ou quando você não me garante p (e neste caso eu sou livre para te garantir q ou não sem quebrar a promessa).*

Se eu for eleito, eu vou abaixar os impostos

Falsa *A proposição é falsa se eu for eleito e não abaixar os impostos.*

Verdadeira *Se eu não for eleito, eu posso abaixar os impostos ou não, sem assim quebrar minha promessa. Logo, se eu não for eleito, a proposição condicional é verdadeira independentemente de se eu abaixar os impostos ou não.*

Observação

Linguagem usual a implicação $p \rightarrow q$ supõe uma relação de causa e efeito entre p e q .

“Se fizer sol amanhã, eu vou à praia”.

Lógica $p \rightarrow q$ diz apenas que não teremos p verdadeiro e q falso ao mesmo tempo.

“Se hoje é domingo, então $2+2=5$ ”.

Observação

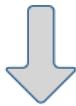
Linguagem usual a implicação $p \rightarrow q$ supõe uma relação de causa e efeito entre p e q .

“Se fizer sol amanhã, eu vou à praia”.

Lógica $p \rightarrow q$ diz apenas que não teremos p verdadeiro e q falso ao mesmo tempo.

“Se hoje é domingo, então $2+2=5$ ”.

Note que se p é F, então $p \rightarrow q$ é V para qualquer q



“Uma falsa hipótese implica em qualquer conclusão”.

O Condicional

Exemplo 1

“Se $2+2=5$, então no Brasil não há corrupção”.

Exemplo 2

Quando é que a implicação “Se hoje é terça-feira, então $2+3=6$ ” é Verdadeira?

O Condicional

- Se $p \rightarrow q$ é uma condicional. então:
 - o **converso** de $p \rightarrow q$ é a implicação $q \rightarrow p$
 - o **inverso** de $p \rightarrow q$ é a implicação $\neg p \rightarrow \neg q$
 - a **contrapositiva** de $p \rightarrow q$ é a implicação $\neg q \rightarrow \neg p$

O Condicional

- Se $p \rightarrow q$ é uma condicional. então:
 - o **converso** de $p \rightarrow q$ é a implicação $q \rightarrow p$
 - o **inverso** de $p \rightarrow q$ é a implicação $\neg p \rightarrow \neg q$
 - a **contrapositiva** de $p \rightarrow q$ é a implicação $\neg q \rightarrow \neg p$

SE MURILO É MINEIRO, ENTÃO MURILO É BRASILEIRO.

- $p \rightarrow q$:
 - p : “Murilo é mineiro”
 - q : “Murilo é brasileiro”
- $q \rightarrow p$: “Se Murilo é brasileiro, então Murilo é mineiro”
- $\neg p \rightarrow \neg q$: “Se Murilo não é mineiro, Murilo não é brasileiro”
- $\neg q \rightarrow \neg p$: “Se Murilo não é brasileiro, Murilo não é mineiro”

Principais Conectivos Lógicos

BICONDICIONAL OU EQUIVALÊNCIA ($p \rightarrow q \wedge q \rightarrow p$)

:

- Notação: $p \leftrightarrow q$
- Definição:

p	q	$p \leftrightarrow q$
V	V	V
V	F	F
F	V	F
F	F	V

- V somente quando:
 - p e q têm o mesmo valor verdade

O Bicondicional

FORMAS DE
EXPRESSAR

$$p \leftrightarrow q$$

- p se, e somente se, q
- p é necessário e suficiente para q
- se p então q , e conversamente

Exemplo 3

a equivalência “ $3 > 2$ se e somente se $0 < 3 - 2$ ” é Verdadeira?

- p : $3 > 2$ (V)
- q : $0 < 3 - 2$ (V)
- logo: $p \leftrightarrow q$ é Verdadeira

Proposições Compostas

DEFINIÇÃO DE PROPOSIÇÕES COMPOSTAS

Podem ter muitas partes componentes , cada parte sendo uma sentença representada por alguma variável proposicional . Estas proposições são construídas com o auxílio dos conectivos lógicos .

Exemplo 4

$$r : p \rightarrow [q \wedge (p \rightarrow q)]$$

$$s : \neg(p \leftrightarrow q) \leftrightarrow [(p \wedge \neg q) \vee (q \wedge \neg p)]$$

$$t : [\neg p \wedge (p \vee q)] \rightarrow q$$

Ordem de precedência

Em uma expressão composta, a ordem de aplicação (precedência) dos operadores é:

1. negação: \neg
2. conjunção: \wedge
3. disjunção: \vee
4. implicação: \rightarrow
5. implicação dupla: \leftrightarrow

Exemplo 5

1. $p \vee \neg q \wedge r$ é equivalente à $p \vee ((\neg q) \wedge r)$
2. $p \rightarrow q \vee r$ é equivalente à $p \rightarrow (q \vee r)$

Tabelas verdade de proposições compostas

A sentença: $s : p \rightarrow [q \wedge (p \rightarrow r)]$

- ▶ envolve 3 proposições independentes
- ▶ logo, há $2^3 = 8$ situações possíveis:

p	q	r	$p \rightarrow [q \wedge (p \rightarrow r)]$
V	V	V	?
V	V	F	?
V	F	V	?
V	F	F	?
F	V	V	?
F	V	F	?
F	F	V	?
F	F	F	?

Questions?

Lógica Proposicional
– Conectivos Lógicos –



Teoria dos Grafos e Computabilidade

— Tabelas verdade e equivalência lógica —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Construindo tabelas verdade

A **tabela verdade** de uma proposição composta de n variáveis proposicionais é obtida por:

1. as primeiras n colunas da tabela devem ser rotuladas com as variáveis proposicionais
 - outras colunas servirão para combinações intermediárias
2. sob cada uma das primeiras colunas, lista-se os 2^n possíveis conjuntos de valores verdade das variáveis proposicionais
3. para cada linha, computa-se os valores verdade restantes

Construindo tabelas verdade

Exemplo 6

Tabela verdade de $(p \vee q) \rightarrow (r \leftrightarrow p)$: (1/3)

p	q	r
V	V	V
V	V	F
V	F	V
V	F	F
F	V	V
F	V	F
F	F	V
F	F	F

Construindo tabelas verdade

Exemplo 6

Tabela verdade de $(p \vee q) \rightarrow (r \leftrightarrow p)$: (2/3)

p	q	r	$p \vee q$	$r \leftrightarrow p$
V	V	V	V	V
V	V	F	V	F
V	F	V	V	V
V	F	F	V	F
F	V	V	V	F
F	V	F	V	V
F	F	V	F	F
F	F	F	F	V

Construindo tabelas verdade

Exemplo 6

Tabela verdade de $(p \vee q) \rightarrow (r \leftrightarrow p)$: (3/3)

p	q	r	$p \vee q$	$r \leftrightarrow p$	$(p \vee q) \rightarrow (r \leftrightarrow p)$
V	V	V	V	V	V
V	V	F	V	F	F
V	F	V	V	V	V
V	F	F	V	F	F
F	V	V	V	F	F
F	V	F	V	V	V
F	F	V	F	F	V
F	F	F	F	V	V

Construindo Tabelas verdade

Exemplo 7

Tabela verdade de $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$: (1/3)

p	q
V	V
V	F
F	V
F	F

Construindo Tabelas verdade

Exemplo 7

Tabela verdade de $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$: (2/3)

p	q	$p \rightarrow q$	$\neg q$	$\neg p$	$\neg q \rightarrow \neg p$
V	V	V	F	F	V
V	F	F	V	F	F
F	V	V	F	V	V
F	F	V	V	V	V

Construindo Tabelas verdade

Exemplo 7

Tabela verdade de $(p \rightarrow q) \leftrightarrow (\neg q \rightarrow \neg p)$: (3/3)

p	q	$p \rightarrow q$	$\neg q$	$\neg p$	$\neg q \rightarrow \neg p$	\leftrightarrow
V	V	V	F	F	V	V
V	F	F	V	F	F	V
F	V	V	F	V	V	V
F	F	V	V	V	V	V



equivalentes

Classificação de Proposições Compostas

Tautologia proposição que é *sempre V* (para todas as possíveis situações).

- Exemplo: $p \vee \neg p$ (verifique!)

Contradição (ou absurdo) : proposição que é *sempre F* (em todas as possíveis situações).

- Exemplo: $p \wedge \neg p$ (verifique!)

Contingência proposição que *pode ser V ou F*, dependendo dos valores verdade de suas variáveis proposicionais.

- Nem tautologia nem contradição.

Equivalência lógica

- ▶ Se $p \leftrightarrow q$ é uma tautologia, as proposições p e q são ditas logicamente equivalentes.
 - ▶ Notação: $p \Leftrightarrow q$
- ▶ Se $p \Leftrightarrow q$, os dois lados são simplesmente diferentes modos de construir a mesma sentença.
- ▶ Um importante recurso usado na argumentação lógica é a substituição de uma proposição por outra que seja equivalente.

Determinação da equivalência por meio de Tabelas Verdade.

Exemplo 8

Mostre que $\neg(p \vee q)$ e $\neg p \wedge \neg q$ são equivalentes. (1/3)

\overline{p}	\overline{q}
p	q
V	V
V	F
F	V
F	F

Equivalência lógica

Determinação da equivalência por meio de Tabelas Verdade.

Exemplo 8

Mostre que $\neg(p \vee q)$ e $\neg p \wedge \neg q$ são equivalentes. (2/3)

p	q	$p \vee q$	$\neg p$	$\neg q$
V	V	V	F	F
V	F	V	F	V
F	V	V	V	F
F	F	F	V	V

Equivalência lógica

Determinação da equivalência por meio de **Tabelas Verdade**.

Exemplo 8

Mostre que $r : \neg(p \vee q)$ e $s : \neg p \wedge \neg q$ são equivalentes. (3/3)

p	q	$p \vee q$	$\neg p$	$\neg q$	$\neg(p \vee q)$	$\neg p \wedge \neg q$	$r \leftrightarrow s$
V	V	V	F	F	F	F	V
V	F	V	F	V	F	F	V
F	V	V	V	F	F	F	V
F	F	F	V	V	V	V	V

Algumas Equivalências importantes

<i>Equivalência</i>	<i>Nome das leis</i>
$p \vee p \Leftrightarrow p$	Idempotência
$p \wedge p \Leftrightarrow p$	
$\neg(\neg p) \Leftrightarrow p$	Dupla negação
$p \vee q \Leftrightarrow q \vee p$	Comutatividade
$p \wedge q \Leftrightarrow q \wedge p$	
$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$	Associatividade
$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	
$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$	Distributividade
$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	
$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$	Leis de De Morgan
$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$	

Uso das equivalências

Exemplo 9

- ▶ $p \vee q$: "O rio é raso ou poluído."
- ▶ $\neg(p \vee q)$: ??
- ▶ pelas leis de De Morgan:
$$\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$$
- ▶ logo:
 $\neg(p \vee q)$: "O rio não é raso E não é poluído."

Note que $\neg(p \vee q)$ não é equivalente a

O rio não é raso OU não é poluído.

Aplicações de lógica proposicional

A lógica tem importantes aplicações na Matemática, Ciência da Computação, e diversas outras disciplinas

- ▶ tradução de sentenças em linguagem natural, frequentemente ambíguas, para uma linguagem precisa,
- ▶ especificação de circuitos lógicos,
- ▶ solução de quebra-cabeças (o que é essencial para inteligência artificial),
- ▶ automatização do processo de construção de provas matemáticas,

Traduzindo Sentenças para Lógica

Exemplo 10

Encontrar a proposição que traduz a seguinte sentença:

Você não pode andar de patins se você tem menos do que 1,20m, a não ser que você tenha mais do que 16 anos'

► Definindo:

q: "você pode andar de patins"

r: "você tem menos do que 1,20m"

s: "você tem mais do que 16 anos"

► a sentença pode ser traduzida por:

$$p : (r \wedge \neg s) \rightarrow \neg q$$

Especificação de sistemas

Traduzir sentenças de linguagem natural para linguagem lógica é parte essencial da especificação de sistemas de hardware e software.

Exemplo 11

Expresse a especificação como uma proposição composta

A resposta automática não pode ser enviada quando o sistema de arquivos está cheio'

- ▶ Definindo:
 - q: "a resposta automática pode ser enviada"
 - r: "o sistema de arquivos está cheio"
- ▶ a especificação pode ser traduzida por:

$$p : r \rightarrow \neg q$$

Questions?

Lógica Proposicional
– Tabelas verdade e equivalência
lógica –



Teoria dos Grafos e Computabilidade

— Lógica de Predicados —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Predicados e Quantificadores —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Predicados e quantificadores

SERVEM PARA DECLARAÇÕES DA FORMA

- ▶ $x > 3$
- ▶ $x = y + 3$
- ▶ $x + y = z$

OBSERVAÇÕES

- ▶ Não são V nem F enquanto os valores das variáveis não forem especificados .
- ▶ Produção de proposições a partir destas declarações .

Predicados

- ▶ A declaração x é maior do que 3 tem duas partes:
 - ▶ a variável x (= “sujeito”)
 - ▶ é maior do que 3 (= “predicado”)
- ▶ O predicado é uma propriedade que o sujeito da declaração pode ter.
- ▶ Podemos denotar x é maior do que 3 por $P(x)$:
 - ▶ P é o predicado
 - ▶ x é a variável

Diz-se também que $P(x)$ é o valor da função proposicional P em x . Uma vez que um valor tenha sido atribuído a x , $P(x)$ se torna uma proposição e tem um valor verdade.

Predicados

Exemplo 1

seja $P(x)$ a declaração $x > 3$. Quais são os valores verdade de $P(4)$ e $P(2)$?

- ▶ $P(4)$, que é “ $4 > 3$ ”, é V
- ▶ $P(2)$, que é “ $2 > 3$ ”, é F

Exemplo 2

$$x = y + 3.$$

- ▶ Pode ser denotado por $Q(x, y)$
- ▶ Quando se atribui valores para x e para y , $Q(x, y)$ passa a ter um valor verdade.

Exemplo 3

Seja $Q(x, y)$ a declaração “ $x = y + 3$ ”. Quais são os valores verdade de $Q(1, 2)$ e $Q(3, 0)$?

Similarmente, $R(x, y, z)$ pode ser “ $x + y = z$ ”.

Exemplo 4

quais os valores verdade de $R(1, 2, 3)$ e $R(0, 0, 1)$?

Exemplo 3

Seja $Q(x, y)$ a declaração “ $x = y + 3$ ”. Quais são os valores verdade de $Q(1, 2)$ e $Q(3, 0)$?

Similarmente, $R(x, y, z)$ pode ser “ $x + y = z$ ”.

Exemplo 4

quais os valores verdade de $R(1, 2, 3)$ e $R(0, 0, 1)$?

Em geral, uma declaração envolvendo as n variáveis x_1, x_2, \dots, x_n pode ser denotada por: $P(x_1, x_2, \dots, x_n)$

- ▶ que é o valor da função P para a tupla: (x_1, x_2, \dots, x_n)
- ▶ P também é chamado de predicado

- ▶ Quando se atribui valores a todas as variáveis em uma função proposicional, a declaração resultante se torna uma proposição com um valor verdade determinado.
- ▶ Outra forma de criar uma proposição a partir de uma função proposicional: a quantificação
- ▶ Dois tipos principais de quantificadores: quantificação universal e quantificação existencial.

Quantificador Universal

- Muitas declarações afirmam que uma propriedade é V ou F para todos os valores de uma variável em um domínio em particular

Quantificador Universal

- Muitas declarações afirmam que uma propriedade é V ou F para todos os valores de uma variável em um domínio em particular
ou seja, em um universo de discurso ou domínio

Quantificador Universal

- Muitas declarações afirmam que uma propriedade é V ou F para todos os valores de uma variável em um domínio em particular
 - ou seja, em um universo de discurso ou domínio
- Tal declaração é expressa com um quantificador universal :
 - estabelece que $P(x)$ é V para todos os valores de x no universo de discurso
 - é o universo de discurso que especifica os possíveis valores da variável x .

Quantificador Universal

A quantificação universal de $P(x)$ é a proposição:

$P(x)$ é V para todos os valores de x no universo de discurso.

Denotada por: $\forall x P(x)$

- \forall é o quantificador universal
- “para todo x , $P(x)$ ”
- “para todos os x , $P(x)$ ”

Quantificador Universal

A quantificação universal de $P(x)$ é a proposição:

$P(x)$ é V para todos os valores de x no universo de discurso.

Denotada por: $\forall x P(x)$

- \forall é o quantificador universal
- “para todo x , $P(x)$ ”
- “para todos os x , $P(x)$ ”

Exemplo 5

Seja $P(x)$ dado por “ $x + 1 > x$ ”.

- Qual o valor verdade da quantificação $\forall x P(x)$, sendo que o universo de discurso consiste de todos os números reais?

como $P(x)$ é V para todos os reais x , a quantificação $\forall x P(x)$
é V

Exemplo 6

Seja $Q(x)$ a declaração “ $x < 2$ ”.

- ▶ Qual o valor verdade da quantificação $\forall x Q(x)$?
- ▶ O universo de discurso consiste de todos os números reais.

Exemplo 6

Seja $Q(x)$ a declaração “ $x < 2$ ”.

- ▶ Qual o valor verdade da quantificação $\forall x Q(x)$?
- ▶ O universo de discurso consiste de todos os números reais.

Solução

- ▶ $Q(x)$ não é verdade para todo número real x
- ▶ $Q(3)$, por exemplo, é F
- ▶ Portanto: $\forall x Q(x)$ é F

Quantificador Universal

- Quando todos os elementos do universo de discurso podem ser listados , como

x_1, x_2, \dots, x_n

- Segue que a quantificação universal é o mesmo que a conjunção :

$$P(x_1) \wedge P(x_2) \wedge \dots \wedge P(x_n)$$

a qual é V sse:

$P(x_1), P(x_2), \dots, P(x_n)$ são todos V

Quantificador Universal

Exemplo 7

qual o valor verdade de $\forall x P(x)$, em que:

- ▶ $P(x)$ é “ $x^2 < 10$ ”
- ▶ o universo de discurso são os inteiros não maiores do que 4?

Quantificador Universal

Exemplo 7

qual o valor verdade de $\forall x P(x)$, em que:

- $P(x)$ é “ $x^2 < 10$ ”
- o universo de discurso são os inteiros não maiores do que 4?

como $P(4)$ é F, segue que $\forall x P(x)$ é F

Quantificador Universal

Exemplo 7

qual o valor verdade de $\forall x P(x)$, em que:

- ▶ $P(x)$ é “ $x^2 < 10$ ”
- ▶ o universo de discurso são os inteiros não maiores do que 4?

como $P(4)$ é F, segue que $\forall x P(x)$ é F

Exemplo 8

o que significa a declaração $\forall x T(x)$, se:

- ▶ $T(x)$ é “ x tem pai e mãe”
- ▶ o universo de discurso consiste de todas as pessoas?

Quantificador Universal

Exemplo 7

qual o valor verdade de $\forall x P(x)$, em que:

- ▶ $P(x)$ é “ $x^2 < 10$ ”
- ▶ o universo de discurso são os inteiros não maiores do que 4?

como $P(4)$ é F, segue que $\forall x P(x)$ é F

Exemplo 8

o que significa a declaração $\forall x T(x)$, se:

- ▶ $T(x)$ é “ x tem pai e mãe”
- ▶ o universo de discurso consiste de todas as pessoas?

a declaração pode ser traduzida para “toda pessoa tem pai e mãe” e por consequência é V

Quantificador Universal

- ▶ Especificar o universo de discurso é importante quando se usa quantificadores.
- ▶ O valor verdade de uma declaração quantificada frequentemente depende de quais elementos estão neste universo de discurso.

Exemplo 9

Quantificador Universal

- ▶ Especificar o universo de discurso é importante quando se usa quantificadores.
- ▶ O valor verdade de uma declaração quantificada frequentemente depende de quais elementos estão neste universo de discurso.

Exemplo 9

Qual é o valor verdade de $\forall x (x^2 \geq x)$ se:

- ▶ o universo de discurso consiste de todos os números reais?
- ▶ o universo de discurso consiste de todos os números inteiros?

Quantificador Universal

- ▶ Especificar o universo de discurso é importante quando se usa quantificadores.
- ▶ O valor verdade de uma declaração quantificada frequentemente depende de quais elementos estão neste universo de discurso.

Exemplo 9

Qual é o valor verdade de $\forall x (x^2 \geq x)$ se:

- ▶ o universo de discurso consiste de todos os números reais?
- ▶ o universo de discurso consiste de todos os números inteiros?

Solução

- ▶ note que $x^2 \geq x$ sse $x(x - 1) \geq 0$ ou seja: sse $x \leq 0$ ou $x \geq 1$
 - ▶ $\forall x (x^2 \geq x)$ é **F** se o universo de discurso consiste dos reais
 - ▶ mas é **V** se o universo de discurso consiste dos inteiros

Quantificador Universal

Note que, para mostrar que uma declaração da forma $\forall x P(x)$ é F:

- só é preciso encontrar **um valor** de x no universo de discurso para o qual $P(x)$ é F
- este valor é chamado de **contra-exemplo** da declaração $\forall x P(x)$

Quantificador Universal

Note que, para mostrar que uma declaração da forma $\forall x P(x)$ é F:

- só é preciso encontrar **um valor** de x no universo de discurso para o qual $P(x)$ é F
- este valor é chamado de **contra-exemplo** da declaração $\forall x P(x)$

Exemplo 10

Seja $P(x)$ dado por $x^2 > 0$.

- Para mostrar que a declaração $\forall x P(x)$ é F, onde o universo de discurso consiste dos inteiros, é só mostrar **um contra-exemplo**.
- Vemos que $x = 0$ é um contra-exemplo, uma vez que $x^2 = 0$ quando $x = 0$.

Quantificador Universal

Note que, para mostrar que uma declaração da forma $\forall x P(x)$ é F:

- só é preciso encontrar **um valor** de x no universo de discurso para o qual $P(x)$ é F
- este valor é chamado de **contra-exemplo** da declaração $\forall x P(x)$

Exemplo 10

Seja $P(x)$ dado por $x^2 > 0$.

- Para mostrar que a declaração $\forall x P(x)$ é F, onde o universo de discurso consiste dos inteiros, é só mostrar **um contra-exemplo**.
- Vemos que $x = 0$ é um contra-exemplo, uma vez que $x^2 = 0$ quando $x = 0$.

Buscar contra-exemplos para declarações quantificadas universalmente é uma atividade importante no estudo da matemática.

Quantificador Existencial

- ▶ Muitas declarações matemáticas estabelecem que existe um elemento com uma certa propriedade.
- ▶ Tais declarações são expressas usando quantificação existencial .

Quantificador Existencial

- ▶ Muitas declarações matemáticas estabelecem que existe um elemento com uma certa propriedade.
- ▶ Tais declarações são expressas usando quantificação existencial .

Forma-se uma proposição que é V se e somente se $P(x)$ é V para pelo menos um valor de x no universo de discurso (ou domínio).

A quantificação existencial de $P(x)$ é a proposição:

- ▶ “existe um elemento x no universo de discurso tal que $P(x)$ é V”
- ▶ usa-se a notação: $\exists x P(x)$

Quantificador Existencial

A quantificação existencial de $P(x)$ é a proposição:

- ▶ existe um elemento x no universo de discurso tal que $P(x)$ é V
- ▶ usa-se a notação: $\exists x P(x)$

SIGNIFICADO

- ▶ existe um x tal que $P(x)$
- ▶ existe pelo menos um x tal que $P(x)$
- ▶ para algum x , $P(x)$

Exemplo 11

Seja $P(x)$ a declaração “ $x > 3$ ”.

- ▶ Qual é o valor verdade da quantificação $\exists x P(x)$?
- ▶ O universo de discurso consiste de todos os números reais.

Quantificador Existencial

Exemplo 11

Seja $P(x)$ a declaração “ $x > 3$ ”.

- ▶ Qual é o valor verdade da quantificação $\exists x P(x)$?
- ▶ O universo de discurso consiste de todos os números reais.

$x > 3$ para, por exemplo, $x = 4$ logo: $\exists x P(x)$ é V

Quantificador Existencial

Exemplo 11

Seja $P(x)$ a declaração “ $x > 3$ ”.

- ▶ Qual é o valor verdade da quantificação $\exists x P(x)$?
- ▶ O universo de discurso consiste de todos os números reais.
 $x > 3$ para, por exemplo, $x = 4$ logo: $\exists x P(x)$ é V

Exemplo 12

Seja $Q(x)$ a declaração “ $x = x + 1$ ”.

- ▶ Qual é o valor verdade da quantificação $\exists x Q(x)$?
- ▶ O universo de discurso consiste de todos os números reais.

Quantificador Existencial

Exemplo 11

Seja $P(x)$ a declaração “ $x > 3$ ”.

- ▶ Qual é o valor verdade da quantificação $\exists x P(x)$?
- ▶ O universo de discurso consiste de todos os números reais.
 $x > 3$ para, por exemplo, $x = 4$ logo: $\exists x P(x)$ é V

Exemplo 12

Seja $Q(x)$ a declaração “ $x = x + 1$ ”.

- ▶ Qual é o valor verdade da quantificação $\exists x Q(x)$?
- ▶ O universo de discurso consiste de todos os números reais.

uma vez que $Q(x)$ é F para todos os nros reais, a quantificação existencial $\exists x Q(x)$ é F

Quantificador Existencial

- Quando todos os elementos do universo de discurso podem ser listados , como

$$x_1, x_2, \dots, x_n$$

- segue que a quantificação existencial é o mesmo que a disjunção :

$$P(x_1) \vee P(x_2) \vee \dots \vee P(x_n)$$

a qual é V sse pelo menos um entre

$$P(x_1), P(x_2), \dots, P(x_n) \text{ for } V$$

Exemplo 13

Qual o valor verdade de $\exists x P(x)$, onde:

- ▶ $P(x)$ é a declaração “ $x^2 > 10$ ”
- ▶ o universo de discurso consiste dos inteiros positivos não maiores do que 4?

Quantificador Existencial

Exemplo 13

Qual o valor verdade de $\exists x P(x)$, onde:

- $P(x)$ é a declaração “ $x^2 > 10$ ”
- o universo de discurso consiste dos inteiros positivos não maiores do que 4?

Como o universo do discurso é $\{1, 2, 3, 4\}$, a proposição $\exists x P(x)$ é o mesmo que a disjunção:

$$P(1) \vee P(2) \vee P(3) \vee P(4)$$

Como $P(4)$ é V, segue que $\exists x P(x)$ é V

Quantificadores - Resumo

Resumo

Declaração	Quando é V?	Quando é F?
$\forall x P(x)$	$P(x)$ é V para todo x	Existe um x para o qual $P(x)$ é F
$\exists x P(x)$	Existe um x para o qual $P(x)$ é V	$P(x)$ é F para todo x

Questions?

Lógica de Predicados
– Predicados e Quantificadores –



Teoria dos Grafos e Computabilidade

— Ligando variáveis —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Ligando variáveis

Quando:

- ▶ um quantificador é usado sobre a variável x
- ▶ ou: quando atribuímos um valor a esta variável

dizemos que esta ocorrência da variável está ligada (ou “amarrada”).

Ligando variáveis

Quando:

- ▶ um quantificador é usado sobre a variável x
- ▶ ou: quando atribuímos um valor a esta variável

dizemos que esta ocorrência da variável está ligada (ou “amarrada”).

Uma ocorrência de variável que não está ligada a um quantificador ou fixa em um valor particular é chamada de variável livre.

Ligando variáveis

Todas as variáveis que ocorrem em uma função proposicional devem estar ligadas, para que ela seja considerada uma proposição. Isto pode ser feito com uma combinação de:

- ▶ quantificadores universais
- ▶ quantificadores existenciais
- ▶ atribuições de valores

Ligando variáveis

Todas as variáveis que ocorrem em uma função proposicional devem estar ligadas, para que ela seja considerada uma proposição. Isto pode ser feito com uma combinação de:

- ▶ quantificadores universais
- ▶ quantificadores existenciais
- ▶ atribuições de valores

ESCOPO

- ▶ A parte de uma expressão lógica à qual um quantificador é aplicado é o seu escopo.
- ▶ Uma variável é livre se estiver fora do escopo de todos os quantificadores na fórmula que a especifica.

Exemplo 14

na declaração $\exists x Q(x, y)$:

- ▶ a variável x está ligada à quantificação $\exists x$
- ▶ mas a variável y está livre :
 - ▶ não está ligada a nenhum quantificador
 - ▶ nenhum valor lhe está sendo atribuído.

Ligando variáveis

Exemplo 14

na declaração $\exists x Q(x, y)$:

- ▶ a variável x está ligada à quantificação $\exists x$
- ▶ mas a variável y está livre:
 - ▶ não está ligada a nenhum quantificador
 - ▶ nenhum valor lhe está sendo atribuído.

Exemplo 15

na declaração $\exists x (P(x) \wedge Q(x)) \vee \forall x R(x)$:

- ▶ Todas as variáveis estão ligadas.
- ▶ O escopo do primeiro quantificador, $\exists x$,

Ligando variáveis

Exemplo 14

na declaração $\exists x Q(x, y)$:

- ▶ a variável x está ligada à quantificação $\exists x$
- ▶ mas a variável y está livre:
 - ▶ não está ligada a nenhum quantificador
 - ▶ nenhum valor lhe está sendo atribuído.

Exemplo 15

na declaração $\exists x (P(x) \wedge Q(x)) \vee \forall x R(x)$:

- ▶ Todas as variáveis estão ligadas.
- ▶ O escopo do primeiro quantificador, $\exists x$, é a expressão $P(x) \wedge Q(x)$

Exemplo 14

na declaração $\exists x Q(x, y)$:

- ▶ a variável x está ligada à quantificação $\exists x$
- ▶ mas a variável y está livre:
 - ▶ não está ligada a nenhum quantificador
 - ▶ nenhum valor lhe está sendo atribuído.

Exemplo 15

na declaração $\exists x (P(x) \wedge Q(x)) \vee \forall x R(x)$:

- ▶ Todas as variáveis estão ligadas.
- ▶ O escopo do primeiro quantificador, $\exists x$, é a expressão $P(x) \wedge Q(x)$
- ▶ O escopo do quantificador “ $\forall x$ ” é

Ligando variáveis

Exemplo 14

na declaração $\exists x Q(x, y)$:

- ▶ a variável x está ligada à quantificação $\exists x$
- ▶ mas a variável y está livre :
 - ▶ não está ligada a nenhum quantificador
 - ▶ nenhum valor lhe está sendo atribuído.

Exemplo 15

na declaração $\exists x (P(x) \wedge Q(x)) \vee \forall x R(x)$:

- ▶ Todas as variáveis estão ligadas.
- ▶ O escopo do primeiro quantificador, $\exists x$, é a expressão $P(x) \wedge Q(x)$
- ▶ O escopo do quantificador “ $\forall x$ ” é $R(x)$

Ligando variáveis

Exemplo 14

na declaração $\exists x Q(x, y)$:

- ▶ a variável x está ligada à quantificação $\exists x$
- ▶ mas a variável y está livre:
 - ▶ não está ligada a nenhum quantificador
 - ▶ nenhum valor lhe está sendo atribuído.

Exemplo 15

na declaração $\exists x (P(x) \wedge Q(x)) \vee \forall x R(x)$:

- ▶ Todas as variáveis estão ligadas.
- ▶ O escopo do primeiro quantificador, $\exists x$, é a expressão $P(x) \wedge Q(x)$
- ▶ O escopo do quantificador “ $\forall x$ ” é $R(x)$

Note que esta expressão pode ser escrita como:

$$\exists x (P(x) \wedge Q(x)) \vee \forall y R(y)$$

Observe que é comum usar a mesma letra para representar variáveis ligadas a diferentes quantificadores ,

Observe que é comum usar a mesma letra para representar variáveis ligadas a diferentes quantificadores, desde que os seus escopos não se sobreponham.

Ligando variáveis

Observe que é comum usar a mesma letra para representar variáveis ligadas a diferentes quantificadores, desde que os seus escopos não se sobreponham.

$$\exists x (P(x) \wedge Q(x)) \vee \forall x R(x)$$

Questions?

Lógica de Predicados
– Ligando variáveis –



Teoria dos Grafos e Computabilidade

— Negações —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Negações

Considere a sentença:

“Todo aluno nesta sala já fez um curso de Cálculo”

Considere a sentença:

“Todo aluno nesta sala já fez um curso de Cálculo”

COMO USAR QUANTIFICADOR?

Considere a sentença:

“Todo aluno nesta sala já fez um curso de Cálculo”

COMO USAR QUANTIFICADOR?

Trata-se de uma quantificação universal: $\forall x P(x)$

em que $P(x)$ é “ x já cursou Cálculo”

Negações

Considere a sentença:

“Todo aluno nesta sala já fez um curso de Cálculo”

COMO USAR QUANTIFICADOR?

Trata-se de uma quantificação universal: $\forall x P(x)$

em que $P(x)$ é “ x já cursou Cálculo”

A negação desta sentença é a declaração:

Não é verdade que todo aluno nesta sala já tenha feito um curso de Cálculo

Existe algum estudante em sala que não cursou Cálculo, ou seja: $\exists x \neg P(x)$

Negações

Considere a sentença:

“Todo aluno nesta sala já fez um curso de Cálculo”

COMO USAR QUANTIFICADOR?

Trata-se de uma quantificação universal: $\forall x P(x)$

em que $P(x)$ é “ x já cursou Cálculo”

A negação desta sentença é a declaração:

Não é verdade que todo aluno nesta sala já tenha feito um curso de Cálculo

Existe algum estudante em sala que não cursou Cálculo, ou seja: $\exists x \neg P(x)$

Equivalência: $\neg \forall x P(x) \equiv \exists x \neg P(x)$

Negações

Agora deseja-se negar:

“Existe um estudante nesta sala que já cursou Cálculo”

Agora deseja-se negar:

“Existe um estudante nesta sala que já cursou Cálculo”

COMO USAR QUANTIFICADOR EXISTENCIAL?

Negações

Agora deseja-se negar:

“Existe um estudante nesta sala que já cursou Cálculo”

COMO USAR QUANTIFICADOR EXISTENCIAL?

Trata-se de uma quantificação existencial: $\exists x Q(x)$, em que $Q(x)$ é “ x já cursou Cálculo”

A negação desta declaração é:

Não é verdade que existe nesta sala um estudante que já tenha cursado Cálculo

Todo estudante desta sala ainda não cursou Cálculo, ou seja: $\forall x \neg Q(x)$

Negações

Agora deseja-se negar:

“Existe um estudante nesta sala que já cursou Cálculo”

COMO USAR QUANTIFICADOR EXISTENCIAL?

Trata-se de uma quantificação existencial: $\exists x Q(x)$, em que $Q(x)$ é “ x já cursou Cálculo”

A negação desta declaração é:

Não é verdade que existe nesta sala um estudante que já tenha cursado Cálculo

Todo estudante desta sala ainda não cursou Cálculo, ou seja: $\forall x \neg Q(x)$

Equivalência: $\neg \exists x Q(x) \equiv \forall x \neg Q(x)$

Negações - Resumo

Resumo

Negação	Declaração Equivalente	Quando é V?	Quando é F?
$\neg \exists x P(x)$	$\forall x \neg P(x)$	Para todo x , $P(x)$ é F	Existe um x para o qual $P(x)$ é V
$\neg \forall x P(x)$	$\exists x \neg P(x)$	Existe um x para o qual $P(x)$ é F	Para todo x , $P(x)$ é V

Exemplo 16

Seja $H(x)$: “ x é honesto”, então esta declaração é:

$$\exists x H(x)$$

em que o universo de discurso consiste de todos os políticos

A negação desta declaração é

$$\neg \exists x H(x) \equiv \forall x \neg H(x)$$

A qual pode ser expressa como:

- ▶ “Todos os políticos não são honestos”, ou
- ▶ “Todos os políticos são desonestos”

Negações

Seja $C(x)$: “ x come hambúrguers”, então esta declaração é

$$\forall x \ C(x)$$

em que o universo de discurso consiste de todos os americanos

A negação desta declaração é :

$$\neg \forall x \ C(x) \equiv \exists x \ \neg C(x)$$

A qual pode ser expressa como:

- ▶ “Alguns americanos não comem hambúrguers”, ou
- ▶ “Existe pelo menos um americano que não come hambúrguers”

Exemplo 17

A negação de “ $\forall x (x^2 > x)$ ”

- ▶ é a declaração: $\neg \forall x (x^2 > x)$
- ▶ que é equivalente a: $\exists x \neg(x^2 > x)$
- ▶ a qual pode ser reescrita como: $\exists x (x^2 \leq x)$

Exemplo 17

A negação de “ $\forall x (x^2 > x)$ ”

- ▶ é a declaração: $\neg \forall x (x^2 > x)$
- ▶ que é equivalente a: $\exists x \neg(x^2 > x)$
- ▶ a qual pode ser reescrita como: $\exists x (x^2 \leq x)$

Note que o valor-verdade desta declaração depende do universo de discurso.

Exemplo 18

A negação de “ $\exists x (x^2 = 2)$ ”

- ▶ é a declaração: $\neg \exists x (x^2 = 2)$
- ▶ que é equivalente a: $\forall x \neg(x^2 = 2)$
- ▶ a qual pode ser reescrita como: $\forall x (x^2 \neq 2)$

Exemplo 18

A negação de “ $\exists x (x^2 = 2)$ ”

- ▶ é a declaração: $\neg \exists x (x^2 = 2)$
- ▶ que é equivalente a: $\forall x \neg(x^2 = 2)$
- ▶ a qual pode ser reescrita como: $\forall x (x^2 \neq 2)$

Note que o valor-verdade desta declaração depende do universo de discurso.

Traduzindo linguagem para lógica

- ▶ Tarefa crucial em matemática, programação em lógica, IA, engenharia de software e outros.
- ▶ Esta tarefa é mais complexa quando envolve predicados e quantificadores, em que pode haver **mais de um modo** de traduzir uma dada sentença.
- ▶ Não há “receita” sendo o objetivo é produzir expressões **simples** e **úteis**.

Exemplo 19

Use lógica de predicados para expressar:

“Todo estudante nesta turma já estudou Cálculo”.

Exemplo 19

Use lógica de predicados para expressar:

“Todo estudante nesta turma já estudou Cálculo”.

COMO TRADUZIR PARA LÓGICA?

Sentenças para lógica (um quantificador)

Exemplo 19

Use lógica de predicados para expressar:

“Todo estudante nesta turma já estudou Cálculo”.

COMO TRADUZIR PARA LÓGICA?

1. reescrever para facilitar identificação dos quantificadores:

“Para cada estudante nesta turma, este estudante já estudou Cálculo”

2. introduzir uma variável x :

“Para cada estudante x nesta turma, x já estudou Cálculo”

3. incluir o predicado $C(x)$: “ x já estudou Cálculo”

4. assim, assumindo que o universo de discurso consiste dos estudantes na turma:

$$\forall x \ C(x)$$

Sentenças para lógica (um quantificador)

EXISTEM OUTRAS ABORDAGENS CORRETAS

- ▶ pode-se usar universos de discurso diferentes e outros predicados
- ▶ a abordagem escolhida vai depender do raciocínio que queremos desenvolver.

Sentenças para lógica (um quantificador)

Exemplo 20

Podemos estar interessados em focar em um grupo de pessoas maior do que a turma. Se o universo de discurso passar a ser “todas as pessoas”, teremos:

Para cada pessoa x , se a pessoa x é um estudante desta turma, então x já estudou Cálculo”

Sentenças para lógica (um quantificador)

Exemplo 20

Podemos estar interessados em focar em um grupo de pessoas maior do que a turma. Se o universo de discurso passar a ser “todas as pessoas”, teremos:

Para cada pessoa x , se a pessoa x é um estudante desta turma, então x já estudou Cálculo”

Então, definindo:

$E(x)$: a pessoa x está nesta turma

Esta sentença fica:

$\forall x (E(x) \rightarrow C(x))$

Neste caso, a sentença não pode ser expressa como:

$\forall x (E(x) \wedge C(x))$

pois isto significaria: “todas as pessoas são estudantes nesta turma e já estudaram Cálculo” (!!)

Sentenças para lógica (um quantificador)

Exemplo 21

Podemos estar interessados na formação da turma em outros assuntos além do Cálculo.

Neste caso, pode ser mais adequado usar o predicado:

$Q(x, y)$: “o estudante x já estudou a matéria y ”

Teríamos que substituir $C(x)$ por $Q(x, \text{calculo})$ nas abordagens anteriores:

$$\forall x Q(x, \text{calculo})$$

$$\forall x (E(x) \rightarrow Q(x, \text{calculo}))$$

Exemplo 22

Use predicados e quantificadores para expressar (1/2)

Algum estudante nesta sala já foi a São Paulo.

- ▶ Esta sentença significa:

Existe um estudante nesta sala com a propriedade de que este estudante já visitou SP.

- ▶ Introduzindo uma variável x :

“Existe um estudante x nesta sala que possui a propriedade ‘ x já visitou SP’.”

Sentenças para lógica (um quantificador)

Exemplo 22

Use predicados e quantificadores para expressar

(2/2)

Algum estudante nesta sala já foi a São Paulo.

Supondo universo de discurso = “estudantes nesta sala”, obtemos:

$$\exists x S(x)$$

Se o universo de discurso passar para “todas as pessoas”, a sentença fica:

Existe uma pessoa x tendo a propriedade de que x é um estudante
nesta sala e x já foi a SP.

Sentenças para lógica (um quantificador)

Exemplo 22

Use predicados e quantificadores para expressar (2/2)

Algum estudante nesta sala já foi a São Paulo.

Supondo universo de discurso = “estudantes nesta sala”, obtemos:

$$\exists x S(x)$$

Se o universo de discurso passar para “todas as pessoas”, a sentença fica:

Existe uma pessoa x tendo a propriedade de que x é um estudante
nesta sala e x já foi a SP.

- ▶ Se $E(x)$ for “ x é um estudante nesta sala”, então $\exists x (E(x) \wedge S(x))$
- ▶ Note que não pode ser:

$$\exists x (E(x) \rightarrow S(x))$$

pois, para isto ser V, bastaria ter alguém fora da sala . (!!)

Questions?

Lógica de Predicados – Negações –



Teoria dos Grafos e Computabilidade

— Graph traversing —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas



Teoria dos Grafos e Computabilidade

— Depth-First search —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

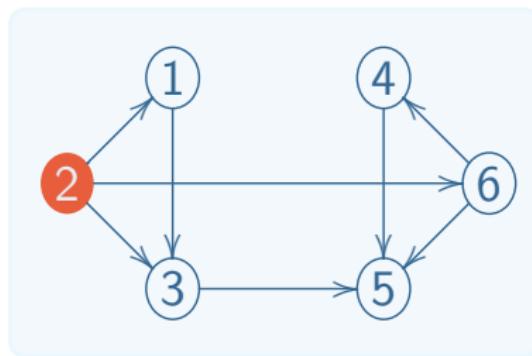
Pontifical Catholic University of Minas Gerais – PUC Minas

Caminhamento em grafos

Na busca em profundidade, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o vértice mais profundo.

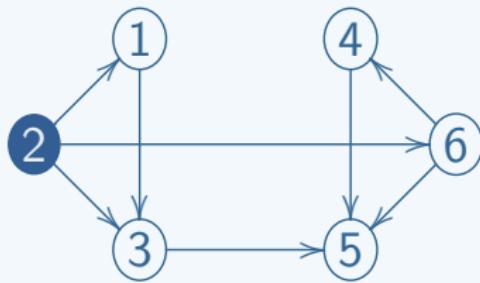
Caminhamento em grafos

Na busca em profundidade, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o vértice mais profundo.



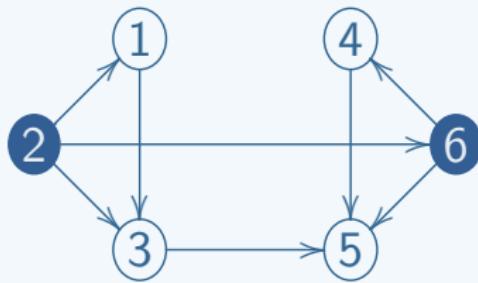
Caminhamento em grafos

Na busca em profundidade, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o vértice mais profundo.



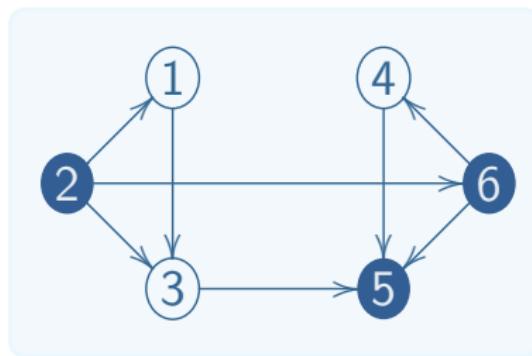
Caminhamento em grafos

Na busca em profundidade, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o vértice mais profundo.



Caminhamento em grafos

Na busca em profundidade, deve-se caminhar no grafo visitando todos os seus vértices sempre procurando o vértice mais profundo.

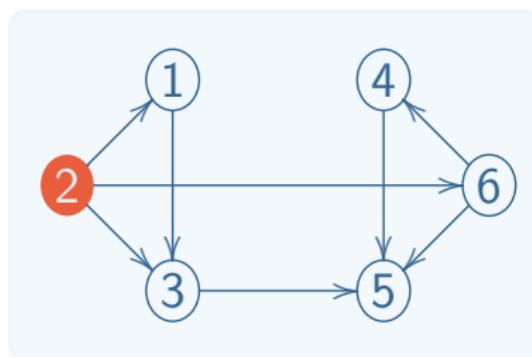


Caminhamento em grafos

Na busca em largura, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.

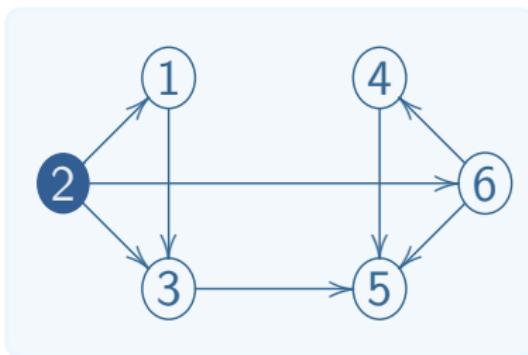
Caminhamento em grafos

Na busca em largura, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



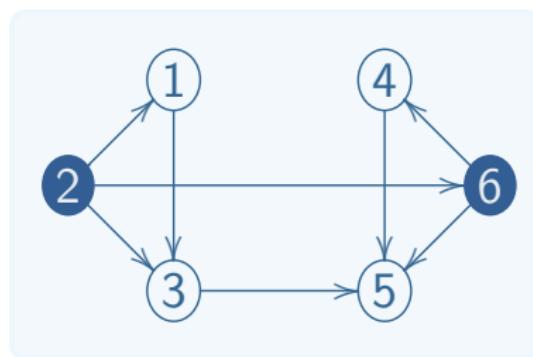
Caminhamento em grafos

Na busca em largura, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



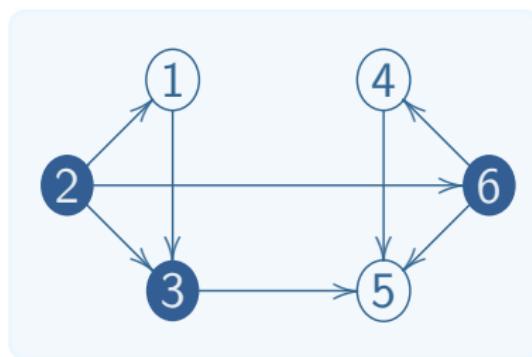
Caminhamento em grafos

Na busca em largura, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.

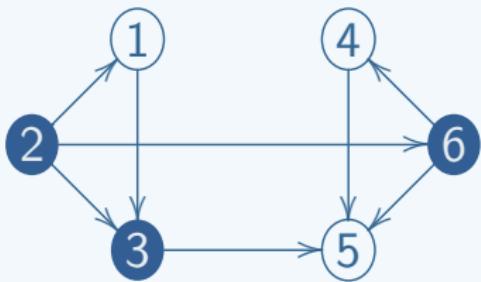
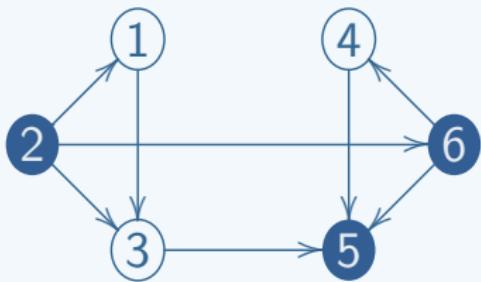


Caminhamento em grafos

Na busca em largura, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



Diferença entre os caminhamentos

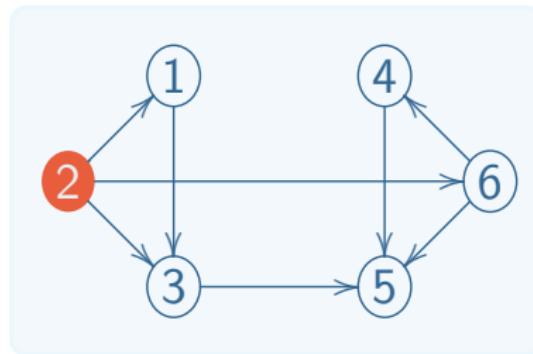


Busca em profundidade

- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;

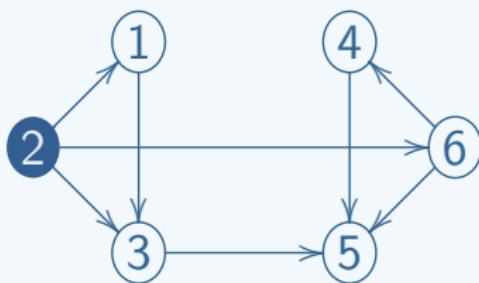
Busca em profundidade

- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;



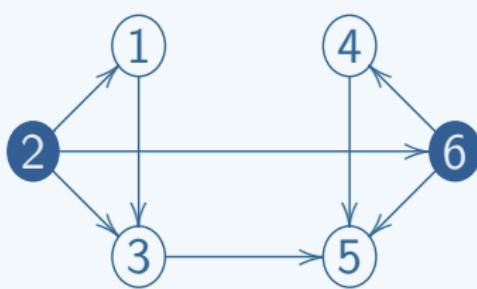
Busca em profundidade

- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;



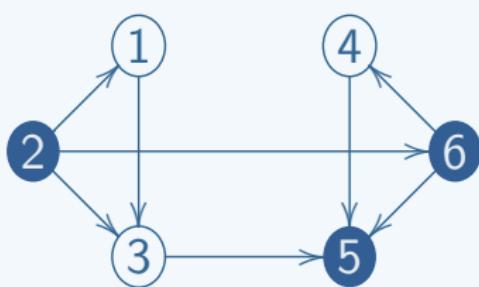
Busca em profundidade

- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;



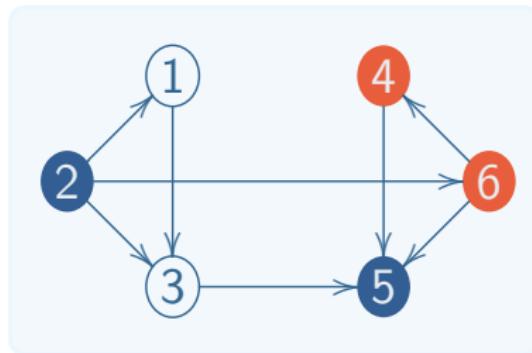
Busca em profundidade

- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;



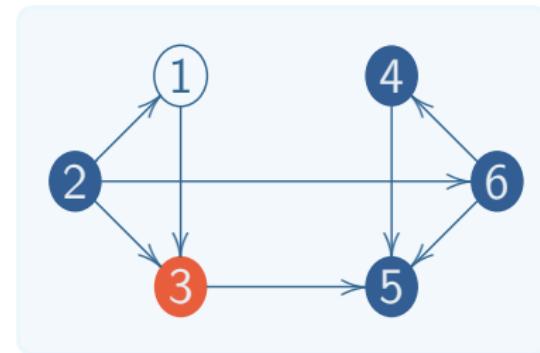
Busca em profundidade

- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;
- Quando todas as arestas de v tiverem sido exploradas volta-se até para explorar arestas que saem do vértice a partir do qual v foi descoberto.



Busca em profundidade

- As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tem arestas não descobertas saindo dele;
- Quando todas as arestas de v tiverem sido exploradas volta-se até para explorar arestas que saem do vértice a partir do qual v foi descoberto.

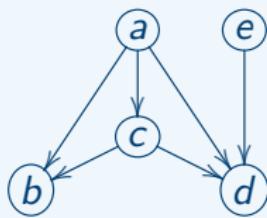


Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**

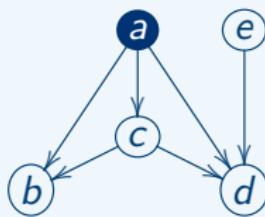
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



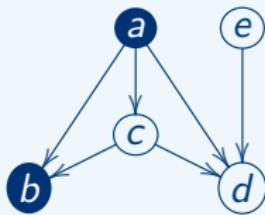
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



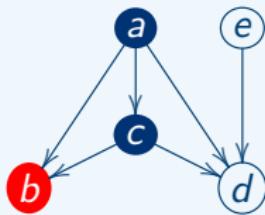
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



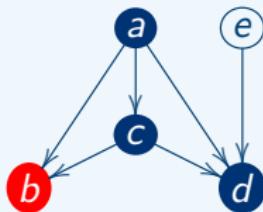
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



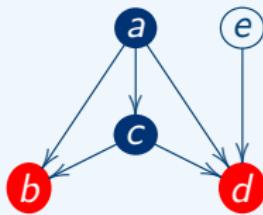
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



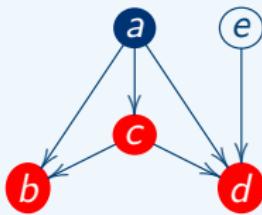
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



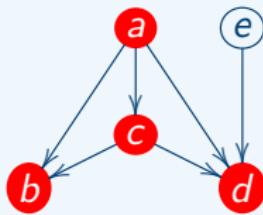
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



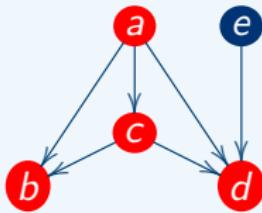
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



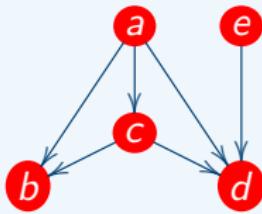
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



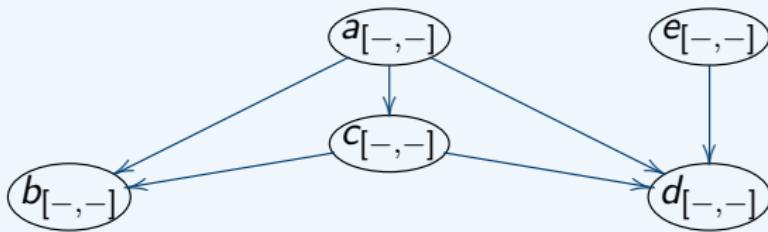
Busca em profundidade

- ▶ Todos os vértice são inicializados com **branco**
- ▶ Quando um vértice é visitado pela primeira vez ele torna-se **azul**
- ▶ Quando sua lista de adjacentes foi totalmente explorada ele torna-se **vermelho**



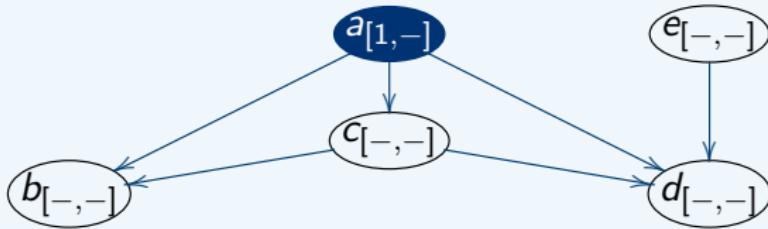
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



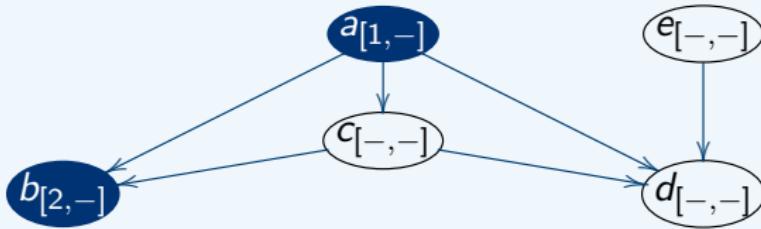
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



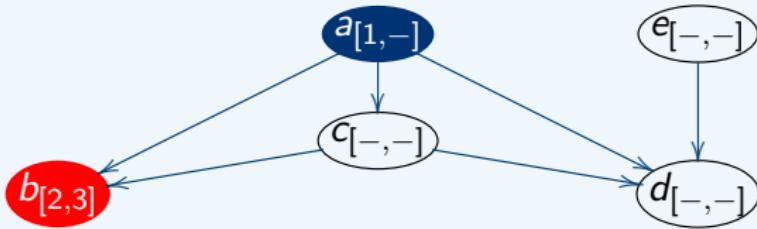
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



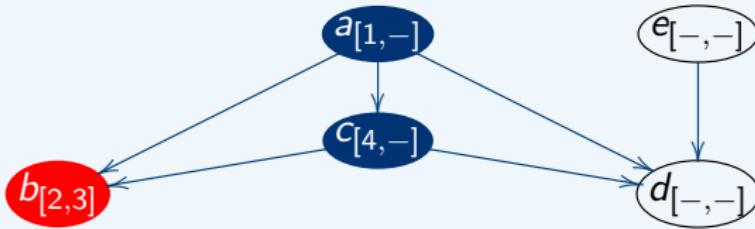
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



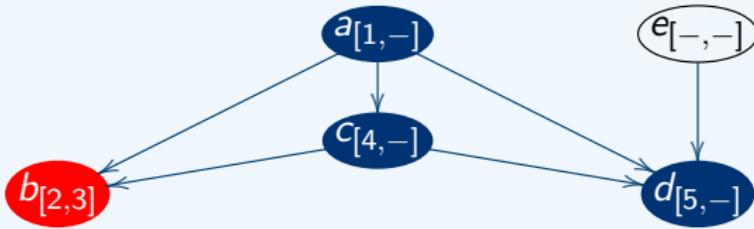
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



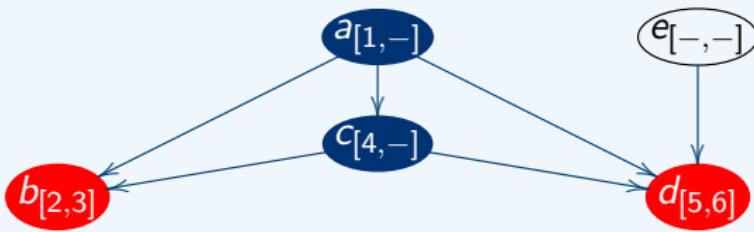
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



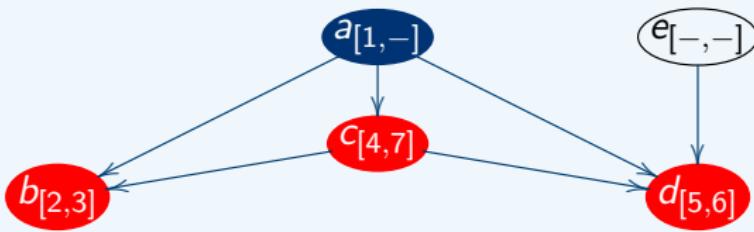
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



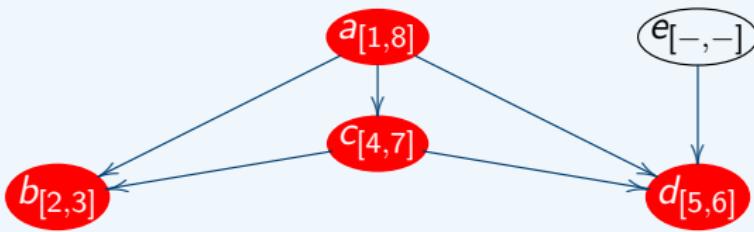
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



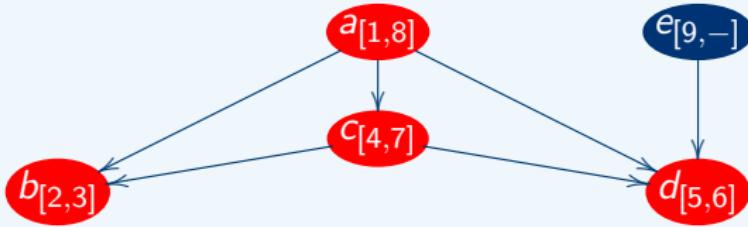
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



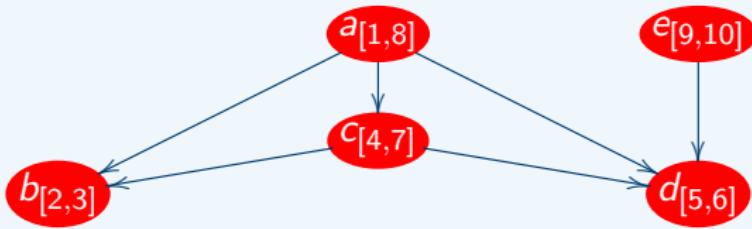
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



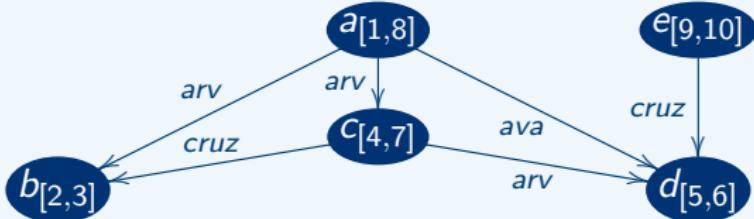
Busca em profundidade

- ▶ O tempo de **descoberta** $d[v]$ é o momento em que o vértice v foi visitado pela **primeira vez**
- ▶ O tempo de **término** do exame da lista de adjacentes $t[v]$ é o momento em que a visita a **toda lista** de vértices adjacentes a v foi concluída.
- ▶ $d[v]$ e $t[v]$ são inteiros entre 1 e $2V$, onde V é o número de vértices do grafo



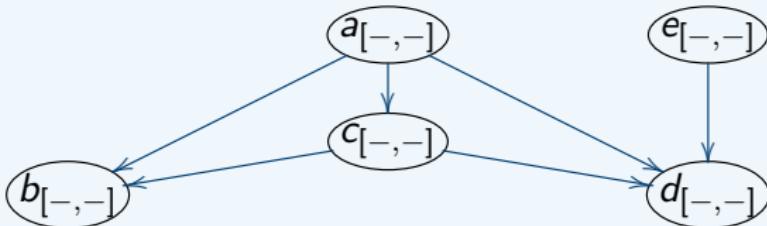
Classificação das arestas

- ▶ De árvore: uma aresta (u,v) é de **árvore** se o vértice v foi visitado a primeira vez passando pela aresta (u,v)
- ▶ De retorno: uma aresta (u,v) é uma aresta de **retorno** se esta conecta um vértice u com um predecessor v já presente em uma árvore de busca
- ▶ De avanço: Não pertencem a árvore de busca em profundidade mas conectam um vértice a um **descendente** que pertence a árvore de busca
- ▶ De cruzamento: conectam vértice de uma **mesma árvore** de busca ou de árvores diferentes



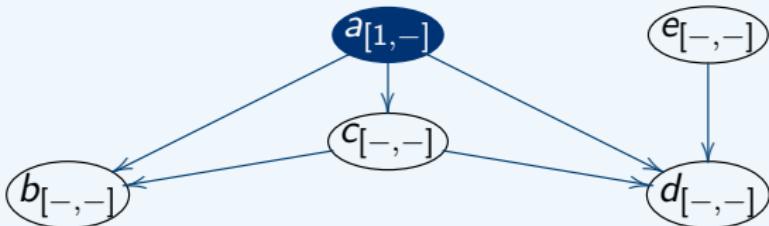
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



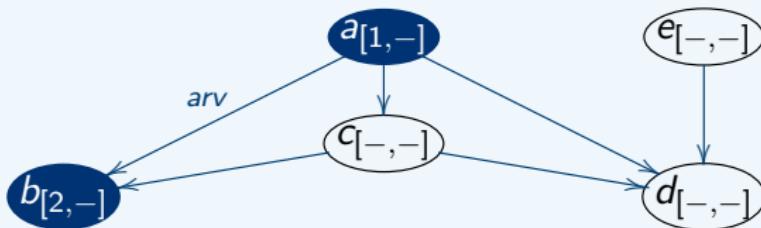
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



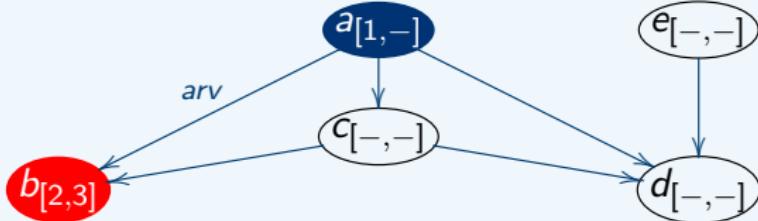
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



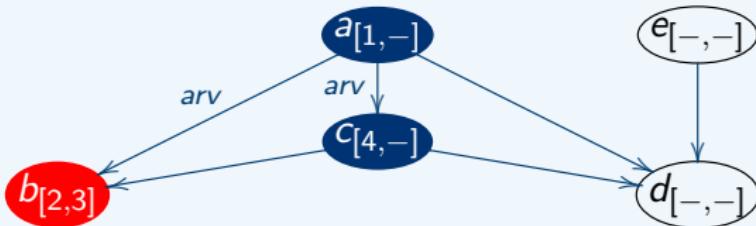
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



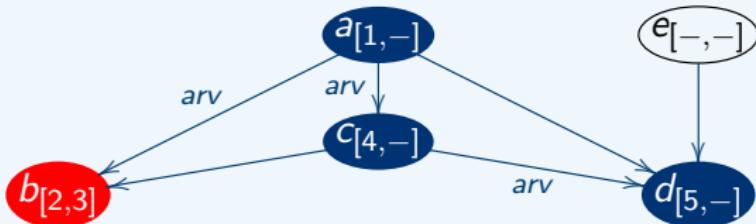
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



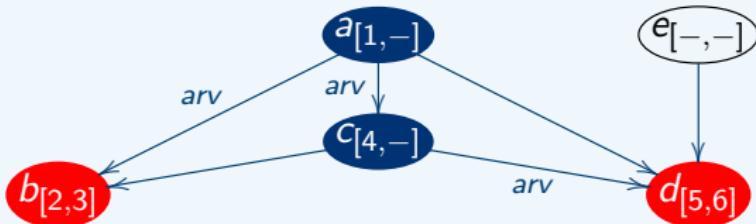
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



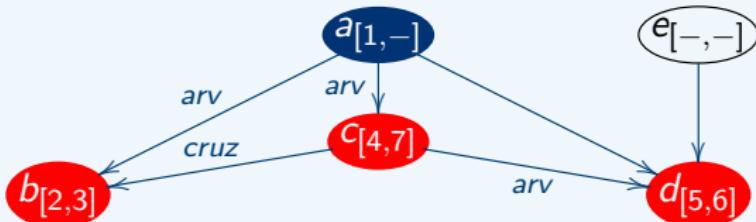
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



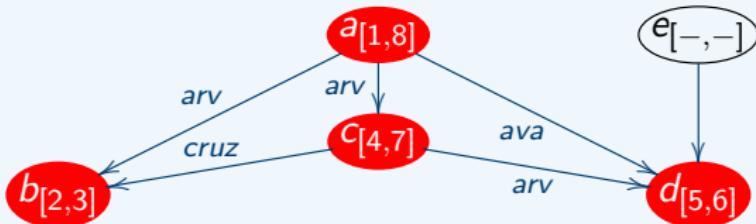
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



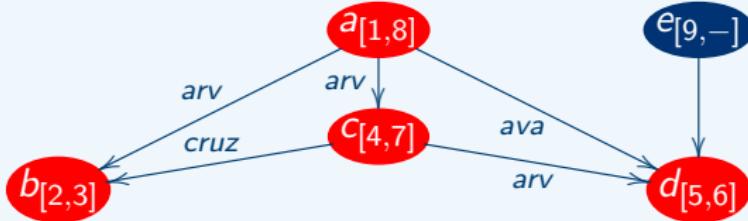
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



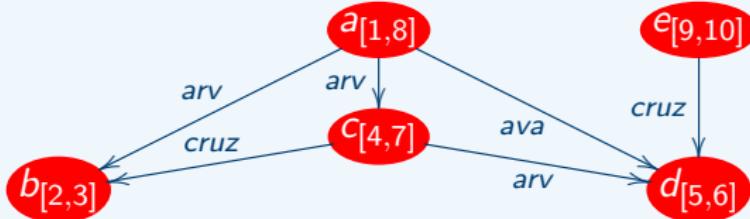
Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



Busca em profundidade

- As arestas $e = (u, v)$ podem ser classificadas pela cor do vértice v que é alcançado quando se passa pela aresta e primeira vez
 - Branco : aresta de árvore
 - Azul : aresta de retorno
 - Vermelho : (i) Se u é visitado antes de v então e é uma aresta de avanço; (ii) Se v é visitado antes de u então e é de cruzamento



Exemplo 1

Como verificar se há um circuito em um grafo direcionado?

Questions?

Graph traversing

– Depth-First search –

Teoria dos Grafos e Computabilidade

— Breadth-First search —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

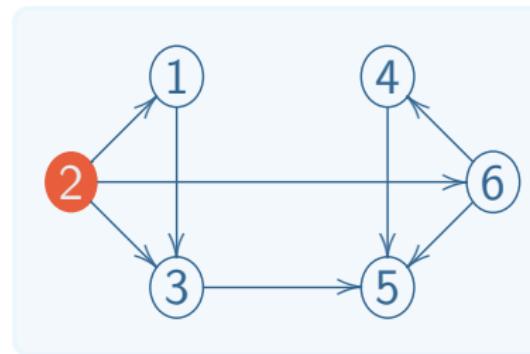
Pontifical Catholic University of Minas Gerais – PUC Minas

Busca em largura

- ▶ Na busca em largura, deve-se expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.

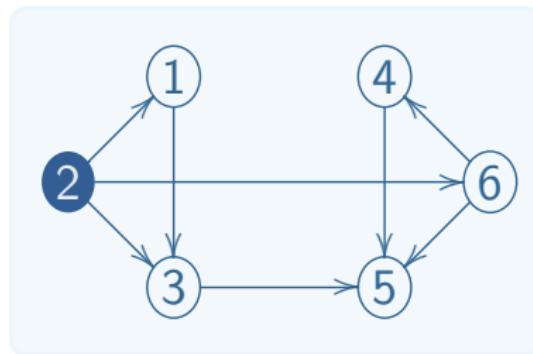
Busca em largura

- Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



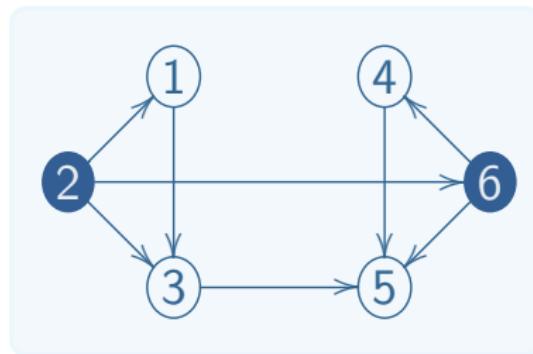
Busca em largura

- Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



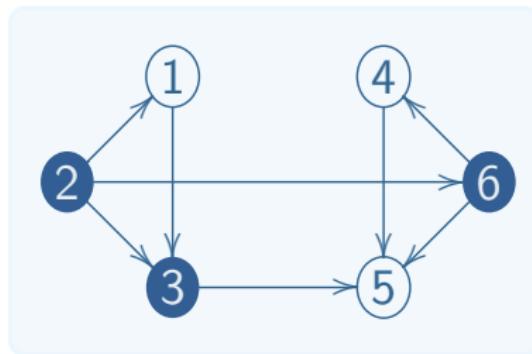
Busca em largura

- ▶ Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



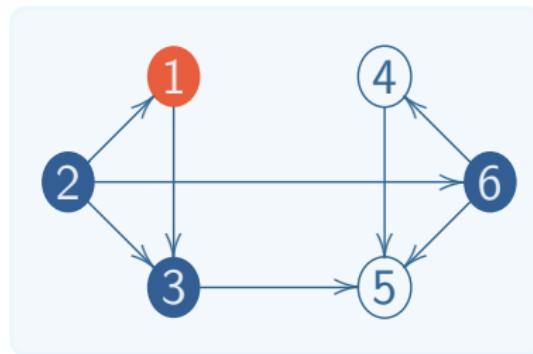
Busca em largura

- ▶ Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



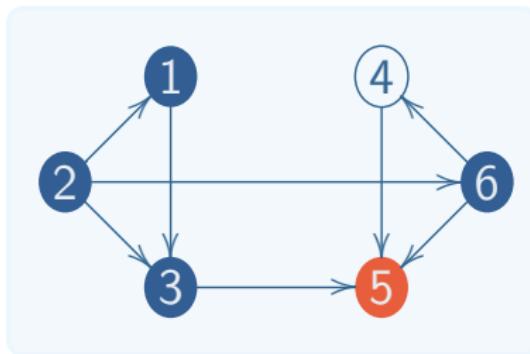
Busca em largura

- Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.



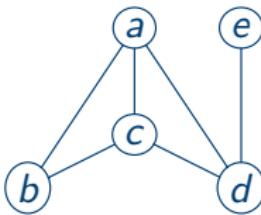
Busca em largura

- ▶ Na busca em largura, deve-se **expandir** o conjunto de vértices de forma uniforme em que são visitados todos os vértices de **mesma distância** ao início antes de visitar outros níveis.
- ▶ Na busca em largura o algoritmo descobre todos os vértices a uma distância k do vértice de origem antes de descobrir os que estão a uma distância $k + 1$



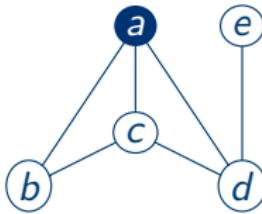
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



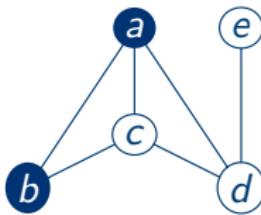
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



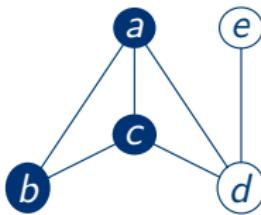
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



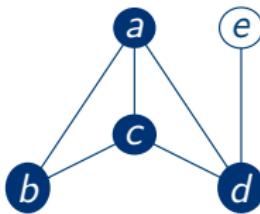
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



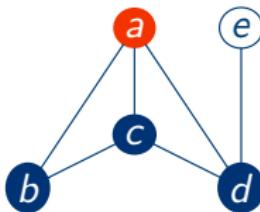
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



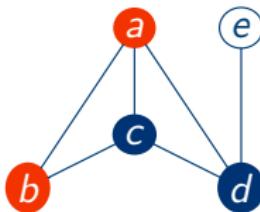
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



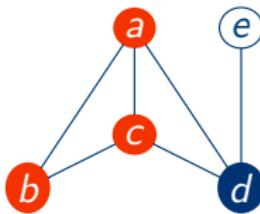
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



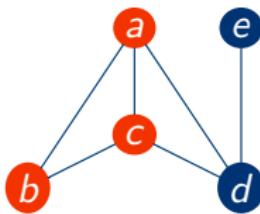
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



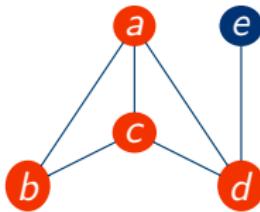
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



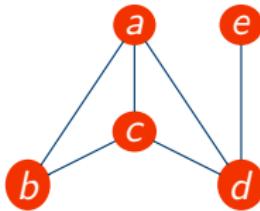
Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



Busca em largura

- ▶ Cada vértice é colorido de branco , azul ou vermelho
- ▶ Todos os vértices são inicializados com branco
- ▶ Quando um vértice é descoberto pela primeira vez ele torna-se azul
- ▶ Vértices cujos adjacentes são todos descobertos tornam-se vermelhos .
Atenção, pois não será necessário todos os descendentes serem descobertos para se tornar vermelho.
- ▶ Se $(u, v) \in A$ e o vértice u é vermelho, entao v tem quer ser azul ou vermelho
- ▶ Vértices azul podem ter adjacentes brancos.



Caminho mais curto

- ▶ A busca em largura encontra o caminho mais curto entre dois vértice u e v .
- ▶ O caminho entre dois vertices quaisquer fica armazenado no vetor antecessor

Alguns algoritmos

Caminho mais curto

- ▶ A busca em largura encontra o caminho mais curto entre dois vértice u e v .
- ▶ O caminho entre dois vertices quaisquer fica armazenado no vetor antecessor

Ordenação topológica

- ▶ Grafos direcionados acíclicos pode ser usados para indicar precedência de eventos
- ▶ Uma aresta direcionada (u, v) indica que a atividade u tem que ocorrer antes da atividade v
- ▶ Os vértices ordenados topologicamente aparecem em ordem inversa aos seus tempos de término na busca em profundidade

Algoritmo menor caminho

Sejam $G = (V, A)$ um grafo direcionado e c_{ij} as distâncias associadas à aresta $(i, j) \in A$. Espera-se encontrar o caminho mais curto entre dois vértices s e t !!!

Algoritmo menor caminho

Sejam $G = (V, A)$ um grafo direcionado e c_{ij} as distâncias associadas à aresta $(i, j) \in A$. Espera-se encontrar o caminho mais curto entre dois vértices s e t !!!

O comprimento de um caminho é igual à soma dos comprimentos (distâncias) das arestas que formam o caminho. A distância ou comprimento de uma aresta pode ter diversas interpretações dependendo da aplicação: custos, distâncias, consumo de combustível, etc.

Algoritmo menor caminho

Sejam $G = (V, A)$ um grafo direcionado e c_{ij} as distâncias associadas à aresta $(i, j) \in A$. Espera-se encontrar o **caminho mais curto** entre dois vértices s e t !!!

O **comprimento de um caminho** é igual à **soma** dos comprimentos (**distâncias**) das arestas que formam o caminho. A **distância** ou **comprimento** de uma aresta pode ter diversas interpretações dependendo da aplicação: custos, distâncias, consumo de combustível, etc.

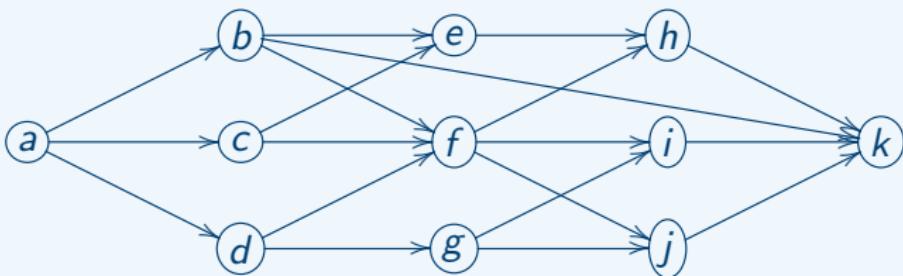
Exemplo 2

Dado um mapa rodoviário, determinar a **rota mais curta** de uma cidade a outra (rota mais rápida, rota com menor consumo de combustível, rota com menor valor de pedágio)

Algoritmo menor caminho

Exemplo 3

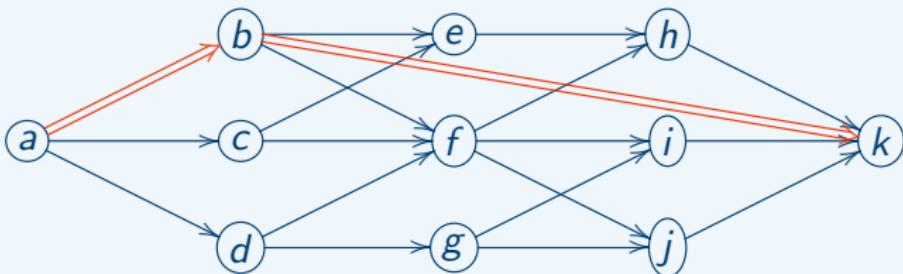
Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis. Determinar o trajeto ótimo (corresponde a achar o **caminho mais curto** de A a K em relação a estes custos).



Algoritmo menor caminho

Exemplo 3

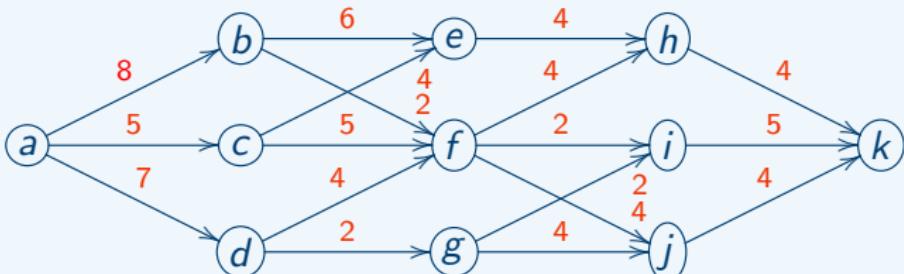
Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis. Determinar o trajeto ótimo (corresponde a achar o **caminho mais curto** de A a K em relação a estes custos).



Algoritmo menor caminho

Exemplo 4

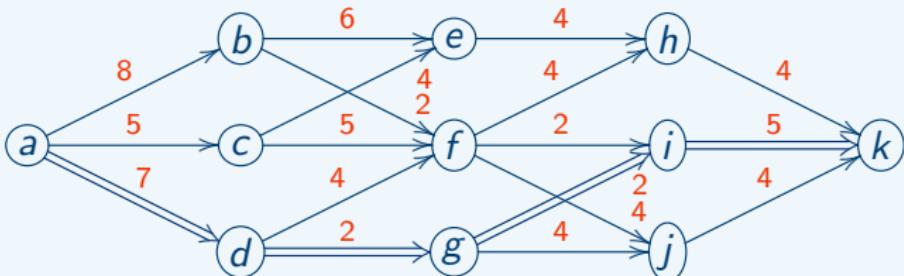
Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis e o custo de construção de cada um. Determinar o trajeto ótimo cujo custo de construção seja mínimo (corresponde a achar o caminho mais curto de A a K em relação a estes custos).



Algoritmo menor caminho

Exemplo 4

Construção de uma estrada entre duas cidades A e K, representado pelo grafo G com diversos trechos possíveis e o custo de construção de cada um. Determinar o trajeto ótimo cujo custo de construção seja mínimo (corresponde a achar o caminho mais curto de A a K em relação a estes custos).



Questions?

Graph traversing
– Breadth-First search –



Teoria dos Grafos e Computabilidade

— Distances over the graph —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

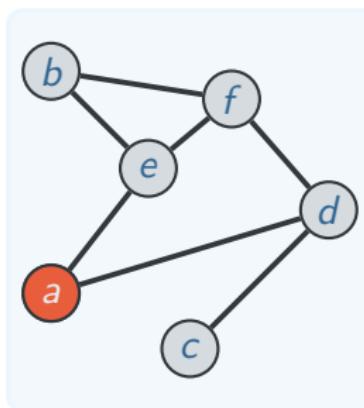
Pontifical Catholic University of Minas Gerais – PUC Minas

Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.

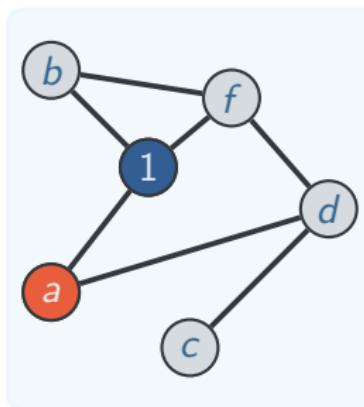
Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



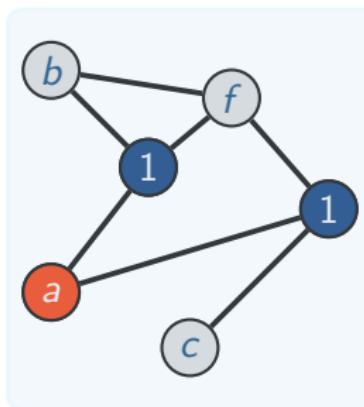
Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



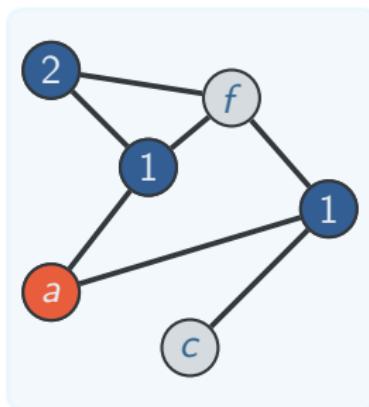
Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



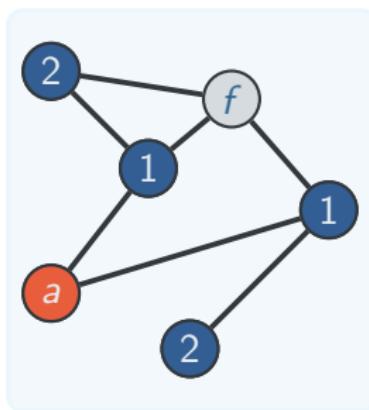
Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



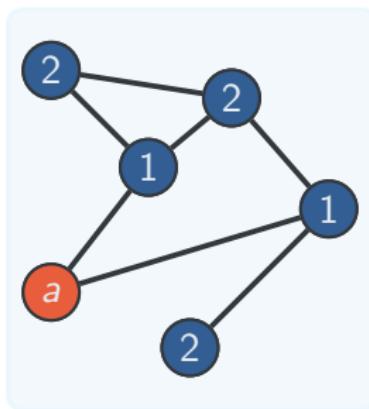
Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



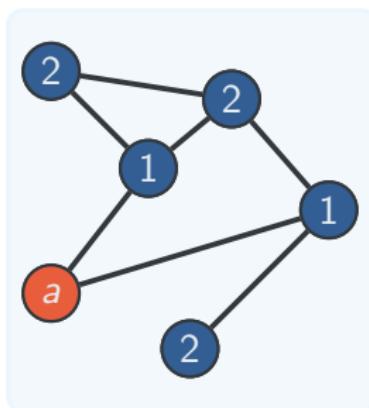
Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



Distances

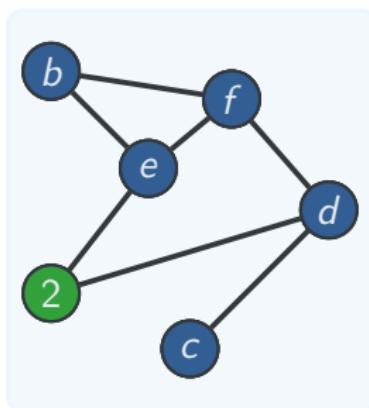
The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



The **eccentricity** of a vertex v is the **greatest distance** between v and any other vertex

Distances

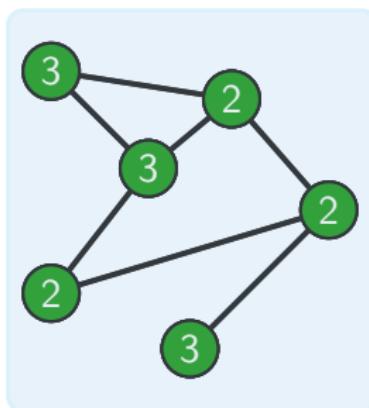
The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



The **eccentricity** of a vertex v is the **greatest distance** between v and any other vertex

Distances

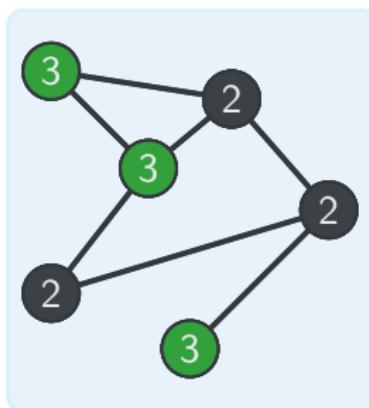
The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



The **eccentricity** of a vertex v is the **greatest distance** between v and any other vertex

Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.

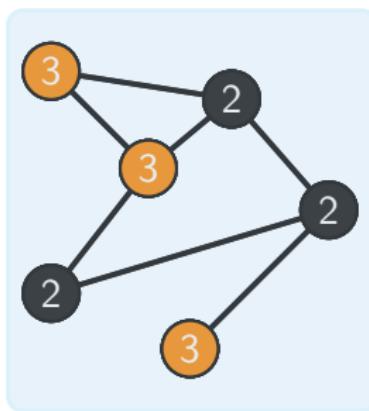


$$r = 2$$

The **radius** r of a graph is the **minimum eccentricity** of any vertex. The central vertex is one whose the eccentricity is r .

Distances

The **distance** between two vertices in a graph is the **number** of edges in a **shortest path** connecting them.



$$d = 3$$

The **diameter** d of a graph is the **maximum eccentricity** of any vertex. The peripheral vertex is one whose the eccentricity is d .

Questions?

Graph traversing

- Distances over the graph –



Teoria dos Grafos e Computabilidade

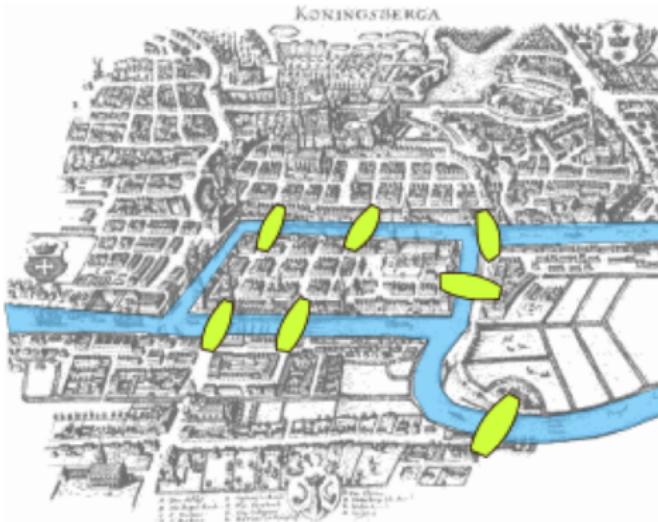
— Special graphs —

Silvio Jamil F. Guimarães

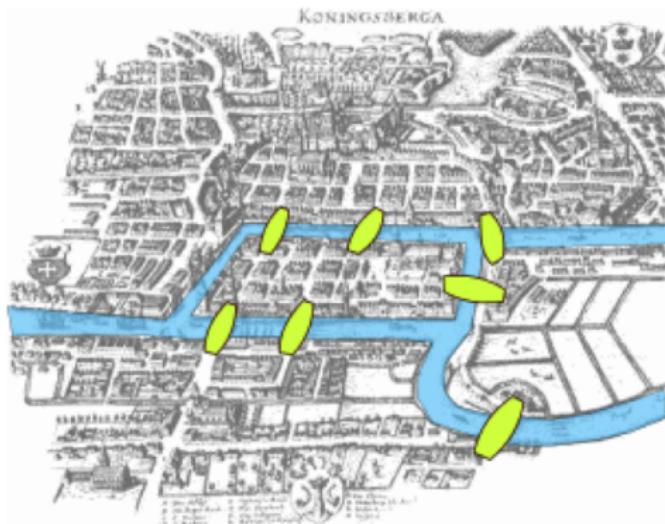
Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Eulerian graph - bridges of Konigsberg

Is there a **path** that crosses each bridge (exactly) once ?

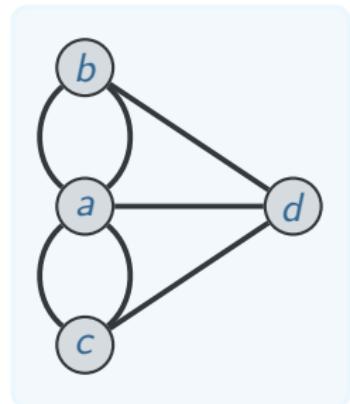
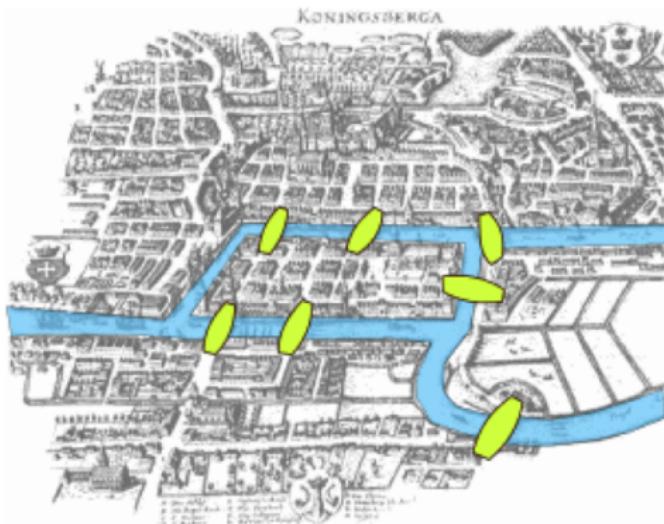


Eulerian graph - bridges of Konigsberg



Is there a **path** that crosses each bridge (exactly) once ?

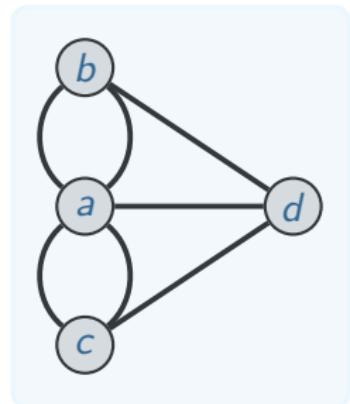
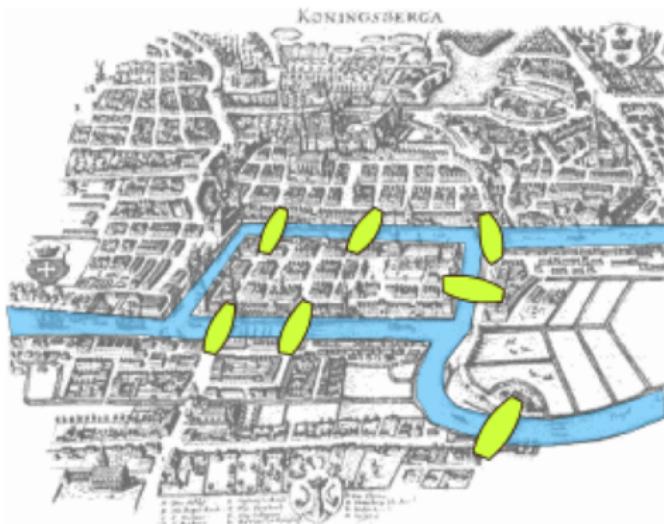
Eulerian graph - bridges of Konigsberg



edges – bridges

Is there a path that crosses each bridge (exactly) once?

Eulerian graph - bridges of Konigsberg



edges – bridges

Is there a path that crosses each bridge (exactly) once?

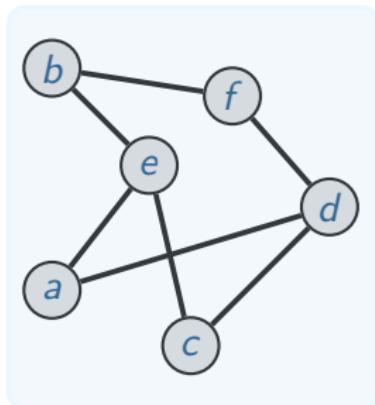
Eulerian tour

Eulerian graph

Eulerian tour (or path): a path in a graph that passes through every edge exactly once.

Eulerian graph

Eulerian tour (or path): a path in a graph that passes through every edge exactly once.

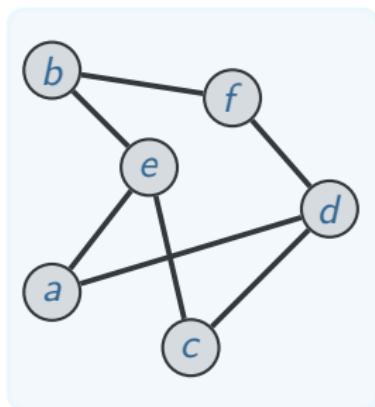


e-a-d-c-e-b-f-d

Eulerian graph

Eulerian tour (or path): a path in a graph that passes through every edge exactly once.

Eulerian cycle (or circuit): a path in a graph that pass through every edge exactly once and starts and ends on the same vertex.

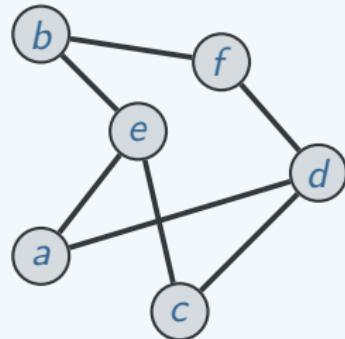


e-a-d-c-e-b-f-d

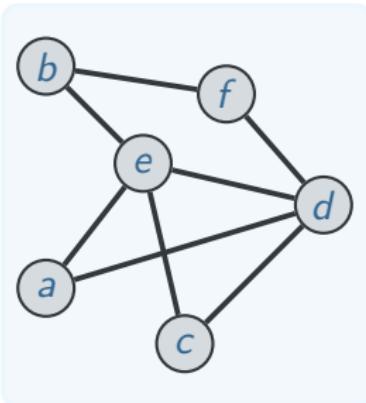
Eulerian graph

Eulerian tour (or path): a path in a graph that passes through every edge exactly once.

Eulerian cycle (or circuit): a path in a graph that pass through every edge exactly once and starts and ends on the same vertex.



e-a-d-c-e-b-f-d



e-a-d-c-e-b-f-d-e

Algorithmic issues - Eulerian tour

Does the given undirected graph $G=(V,E)$ contain an Euler tour?

Algorithmic issues - Eulerian tour

Does the given undirected graph $G=(V,E)$ contain an Euler tour?

Easy to decide

Algorithmic issues - Eulerian tour

Does the given undirected graph $G=(V,E)$ contain an Euler tour?

Easy to decide

Find an Euler tour for the given undirected graph $G=(V,E)$, if possible.

Algorithmic issues - Eulerian tour

Does the given undirected graph $G=(V,E)$ contain an Euler tour?

Easy to decide

Find an Euler tour for the given undirected graph $G=(V,E)$, if possible.

Easy to compute

Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph $G=(V,E)$ **contain** an Hamiltonian tour?

Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph $G=(V,E)$ contain an Hamiltonian tour?

Not easy to decide

Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph $G=(V,E)$ **contain** an Hamiltonian tour?

Not easy to decide

Find an Hamiltonian tour for the given undirected graph $G=(V,E)$, if possible.

Algorithmic issues - Hamiltonian tour

A **Hamiltonian tour** is a path where each **vertex** occurs exactly once

Does the given undirected graph $G=(V,E)$ **contain** an Hamiltonian tour?

Not easy to decide

Find an Hamiltonian tour for the given undirected graph $G=(V,E)$, if possible.

Not easy to compute

Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!



Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!

What's the minimum number of
single flips of a switch to guarantee
that the light bulb turns on?

1. 4
2. 8
3. 16
4. 64



Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!



Light Switches – an exercise

A light bulb is connected to 3 switches in such a way that it lights up only when all the switches are in the proper position. But you don't know what the proper position of each switch is!

DNA Reconstruction – an exercise

Exemplo 5

Given a collection of short DNA strings.

Find longer DNA string that includes them all .

DNA Reconstruction – an exercise

Exemplo 5

Given a collection of short DNA strings.

Find longer DNA string that includes them all .

Possible solutions

- ▶ as a Hamiltonian tour
- ▶ as an Eulerian tour

DNA Reconstruction – an exercise

Exemplo 5

Given a collection of short DNA strings.

Find longer DNA string that includes them all.

Possible solutions

- ▶ as a Hamiltonian tour
- ▶ as an Eulerian tour

How to model over a graph the following instance for the DNA reconstruction problem?

$$S = \{ATG, AGG, TGC, TCC, GTC, GGT, GCA, CAG\}$$

DNA Reconstruction: as a Hamiltonian tour

DNA Reconstruction: as a Hamiltonian tour

DNA Reconstruction: as a Hamiltonian tour

DNA Reconstruction: as an Eulerian tour

DNA Reconstruction: as an Eulerian tour

DNA Reconstruction: as an Eulerian tour

Finding Eulerian tours

Let $G = (V, E)$ be a graph with n vertices being
undirected and connected

Finding Eulerian tours

Let $G = (V, E)$ be a graph with n vertices being
undirected and connected

Determine whether G has an Eulerian tour.

Finding Eulerian tours

Let $G = (V, E)$ be a graph with n vertices being
undirected and connected

Determine whether G has an Eulerian tour.

Theorem If G has an Eulerian tour, G has at most two odd degree vertices.

Finding Eulerian tours

Let $G = (V, E)$ be a graph with n vertices being
undirected and connected

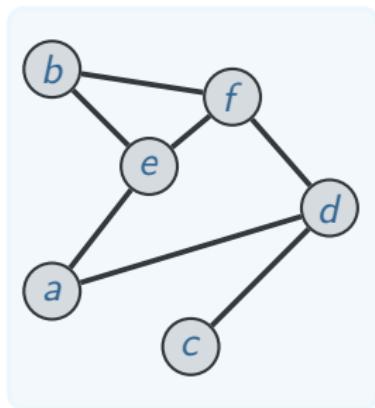
Determine whether G has an Eulerian tour.

Theorem If G has an Eulerian tour, G has at most two odd degree vertices.

If yes, find the tour itself

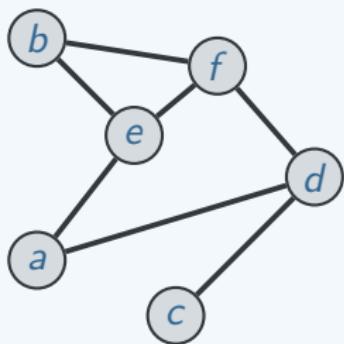
Finding Eulerian tours – Bridges

Let $G=(V,E)$ be an undirected and connected graph. A bridge is an edge, which, if removed, would cause G to be disconnected.



Finding Eulerian tours – Bridges

Let $G=(V,E)$ be an undirected and connected graph. A bridge is an edge, which, if removed, would cause G to be disconnected.



Which of the edges in this graph are bridges?

1. All of them.
2. (d,a)
3. (d,c)
4. (e,a) and (f,g)

Finding Eulerian tours – Bridges

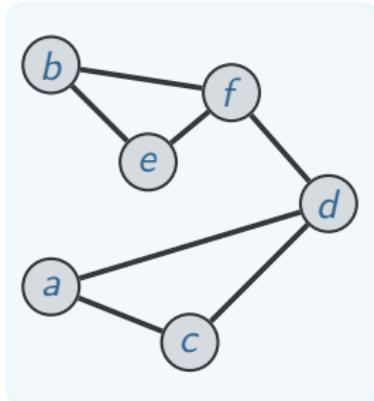
Let $G=(V,E)$ be an undirected and connected graph. A bridge is an edge, which, if removed, would cause G to be disconnected.

In an Eulerian tour, we have to visit every edge on one side of the bridge before we cross it (because there is no coming back).

Finding Eulerian tours – Bridges

Let $G=(V,E)$ be an undirected and connected graph. A **bridge** is an edge, which, if removed, would cause G to be **disconnected**.

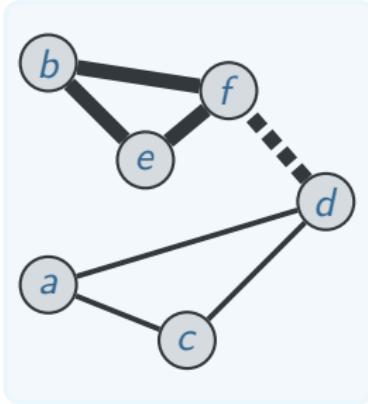
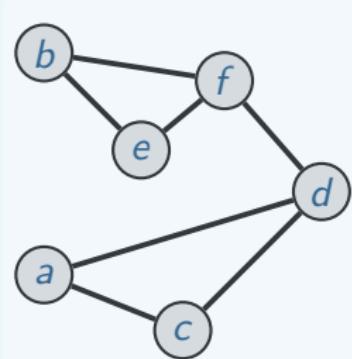
In an Eulerian tour, we have to visit **every edge** on one side of the bridge before we **cross** it (because there is no coming back).



Finding Eulerian tours – Bridges

Let $G=(V,E)$ be an undirected and connected graph. A **bridge** is an edge, which, if removed, would cause G to be **disconnected**.

In an Eulerian tour, we have to visit **every edge** on one side of the bridge before we **cross** it (because there is no coming back).



Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

```
1 Check that G has at most 2 odd degree vertices;  
2 Start at vertex v, an odd degree vertex if possible;  
3 while  $E \neq \emptyset$  do  
4   | if there is more than one edge incident on v then  
5   |   | Cross any edge incident e on v that is not a bridge;  
6   | else  
7   |   | Cross the only edge e available from v;  
8   | end  
9   |  $E = E - \{e\}$ ;  
10 end
```

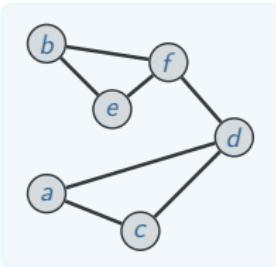
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** *there is more than one edge incident on v* **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



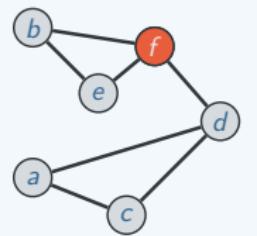
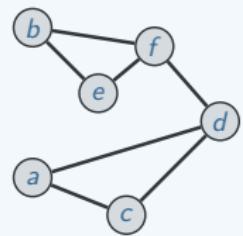
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** *there is more than one edge incident on v* **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

1 Check that G has at most 2 odd degree vertices;

2 Start at vertex v , an odd degree vertex if possible;

3 **while** $E \neq \emptyset$ **do**

4 **if** there is more than one edge incident on v **then**

5 | Cross any edge incident e on v that is not a bridge;

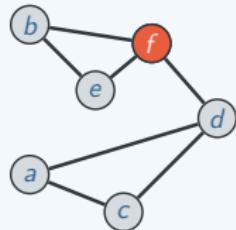
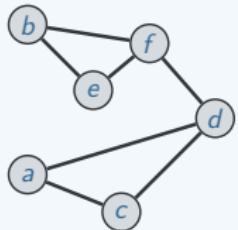
6 **else**

7 | Cross the only edge e available from v ;

8 **end**

9 $E = E - \{e\}$;

10 **end**



Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

1 Check that G has at most 2 odd degree vertices;

2 Start at vertex v , an odd degree vertex if possible;

3 **while** $E \neq \emptyset$ **do**

4 **if** there is more than one edge incident on v **then**

5 | Cross any edge incident e on v that is not a bridge;

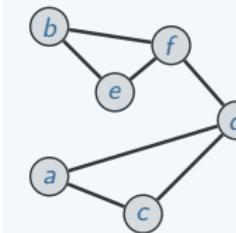
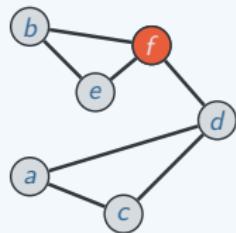
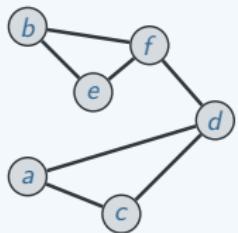
6 **else**

7 | Cross the only edge e available from v ;

8 **end**

9 $E = E - \{e\}$;

10 **end**



Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

1 Check that G has at most 2 odd degree vertices;

2 Start at vertex v , an odd degree vertex if possible;

3 **while** $E \neq \emptyset$ **do**

4 **if** *there is more than one edge incident on v* **then**

5 | Cross any edge incident e on v that is not a bridge;

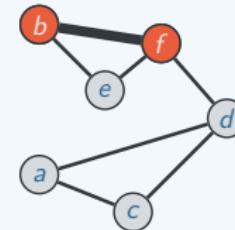
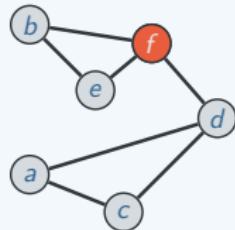
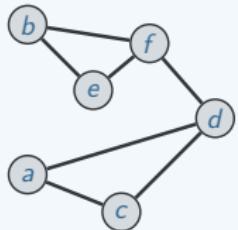
6 **else**

7 | Cross the only edge e available from v ;

8 **end**

9 $E = E - \{e\}$;

10 **end**



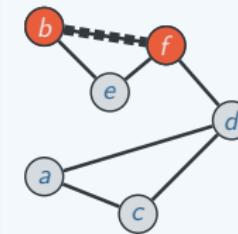
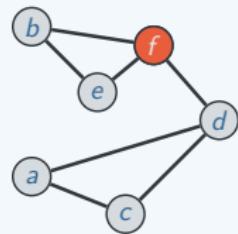
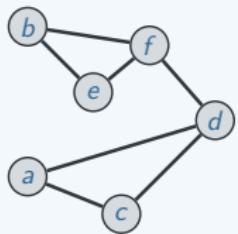
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** *there is more than one edge incident on v* **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



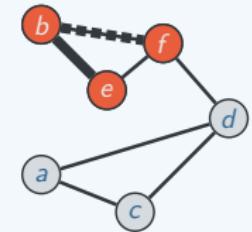
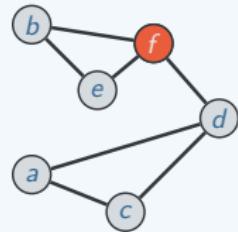
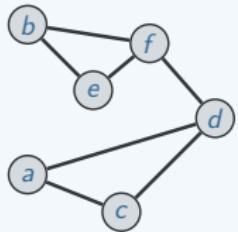
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** *there is more than one edge incident on v* **then**
- 5 | Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 | Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



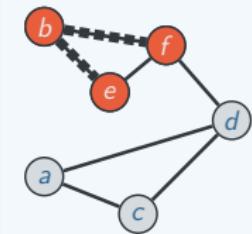
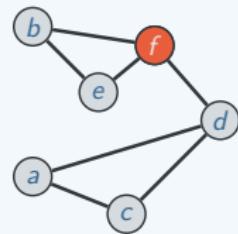
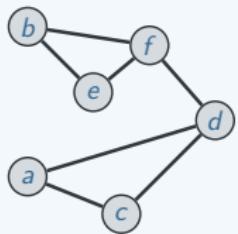
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** *there is more than one edge incident on v* **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



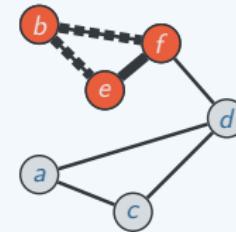
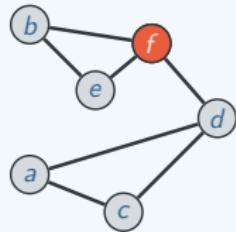
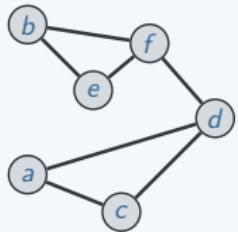
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** there is more than one edge incident on v **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



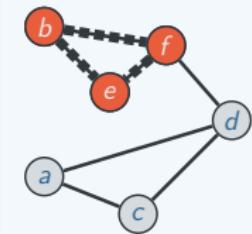
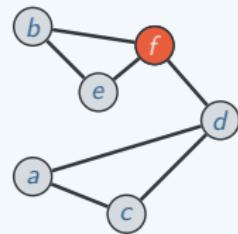
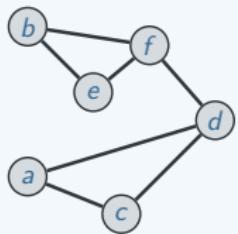
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** there is more than one edge incident on v **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



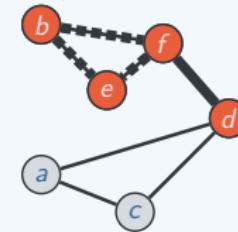
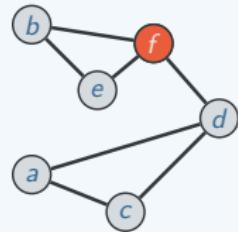
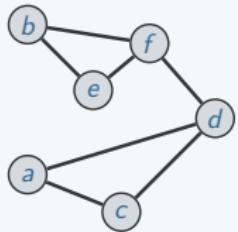
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** *there is more than one edge incident on v* **then**
- 5 | Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 | Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



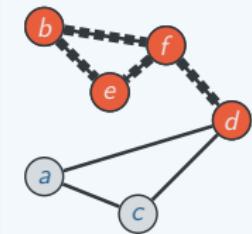
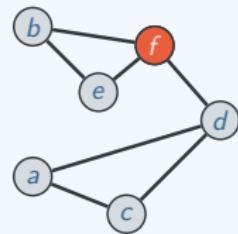
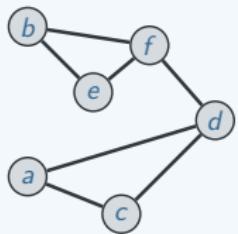
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** there is more than one edge incident on v **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

1 Check that G has at most 2 odd degree vertices;

2 Start at vertex v , an odd degree vertex if possible;

3 **while** $E \neq \emptyset$ **do**

4 **if** *there is more than one edge incident on v* **then**

5 | Cross any edge incident e on v that is not a bridge;

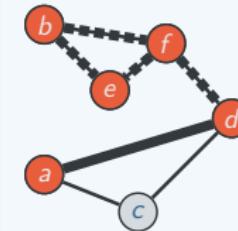
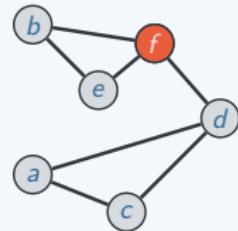
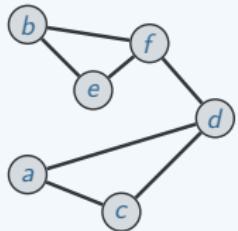
6 **else**

7 | Cross the only edge e available from v ;

8 **end**

9 $E = E - \{e\}$;

10 **end**



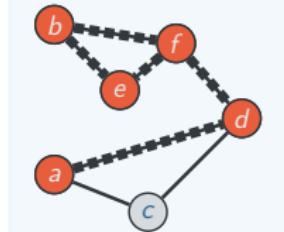
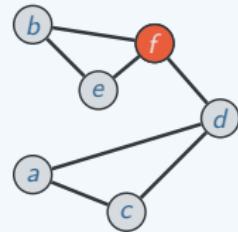
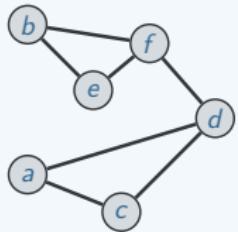
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** there is more than one edge incident on v **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



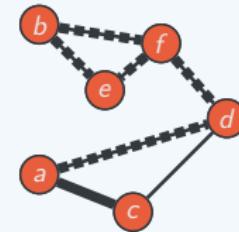
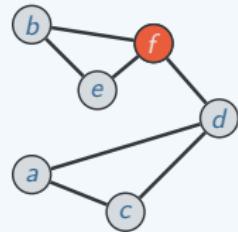
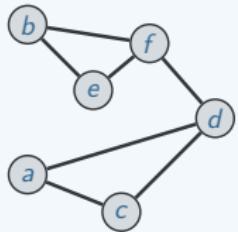
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** there is more than one edge incident on v **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



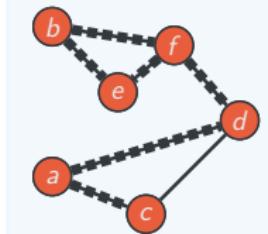
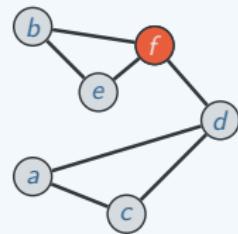
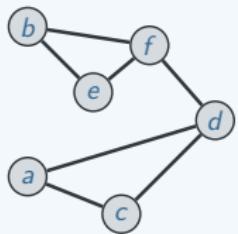
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** there is more than one edge incident on v **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



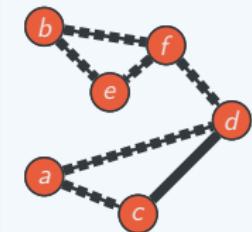
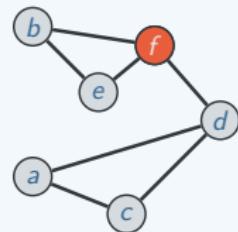
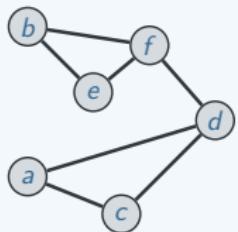
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** there is more than one edge incident on v **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



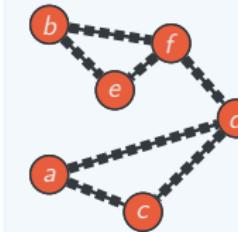
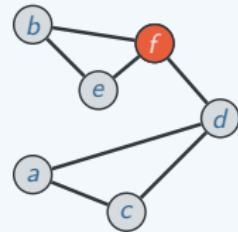
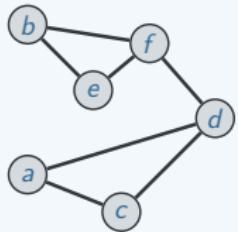
Finding Eulerian tours – Fleury's Algorithm

Algorithm: Eulerian tours – Fleury's Algorithm

input : An undirected and connected graph $G = (V, E)$.

output: The Eulerian tour, if so.

- 1 Check that G has at most 2 odd degree vertices;
- 2 Start at vertex v , an odd degree vertex if possible;
- 3 **while** $E \neq \emptyset$ **do**
- 4 **if** *there is more than one edge incident on v* **then**
- 5 Cross any edge incident e on v that is not a bridge;
- 6 **else**
- 7 Cross the only edge e available from v ;
- 8 **end**
- 9 $E = E - \{e\}$;
- 10 **end**



Questions?

Graph traversing
– Special graphs –



Teoria dos Grafos e Computabilidade

— Connectivity —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas



Teoria dos Grafos e Computabilidade

— Connectivity —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Busca em profundidade *Caminha no grafo*

visitando todos os seus vértices sempre procurando o vértice mais profundo.

Busca em profundidade *Caminha no grafo*

visitando todos os seus vértices sempre procurando o vértice mais profundo.

Busca em largura *Expandir o conjunto de vértices de forma uniforme em que são visitados todos os vértices de mesma distância ao início antes de visitar outros níveis.*

Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $v \in V$

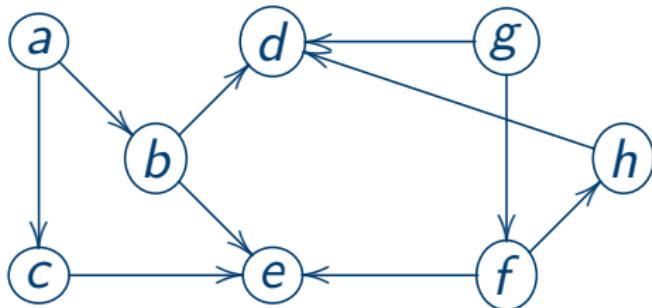
- **direto**: vértices **alcançáveis** de v , com caminho maior ou igual a zero

Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $v \in V$

- **direto**: vértices **alcançáveis** de v , com caminho maior ou igual a zero

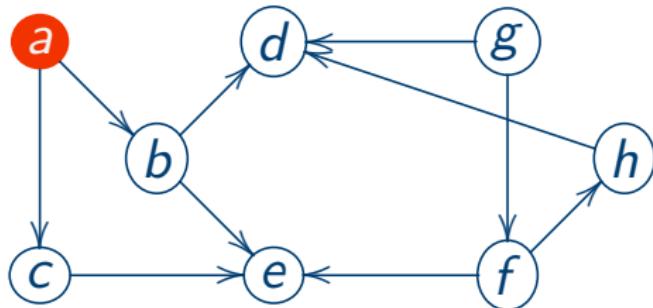


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $v \in V$

- **direto**: vértices **alcançáveis** de v , com caminho maior ou igual a zero

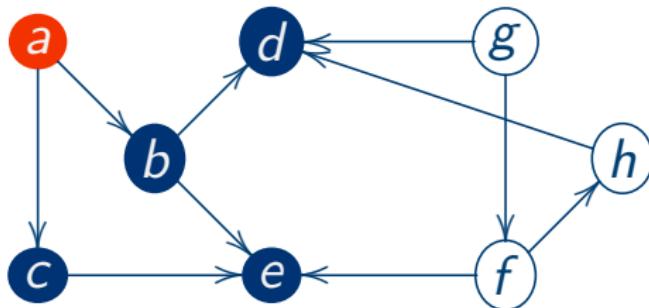


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $v \in V$

- **direto**: vértices **alcançáveis** de v , com caminho maior ou igual a zero

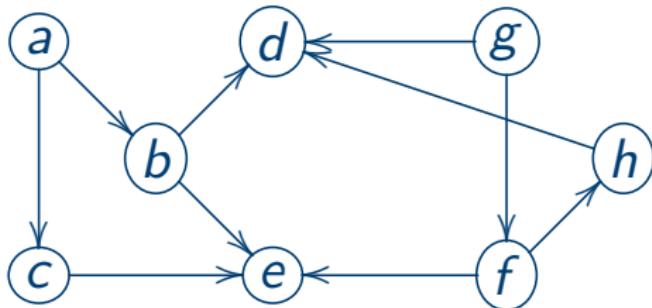


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $v \in V$

- **direto**: vértices **alcançáveis** de v , com caminho maior ou igual a zero
- **inverso**: vértices que **alcançam** v , com caminho maior ou igual a zero

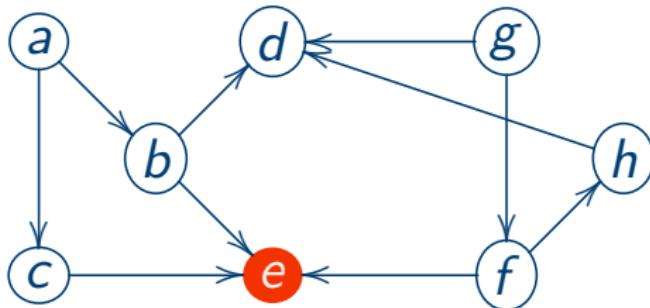


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $v \in V$

- **direto**: vértices **alcançáveis** de v , com caminho maior ou igual a zero
- **inverso**: vértices que **alcançam** v , com caminho maior ou igual a zero

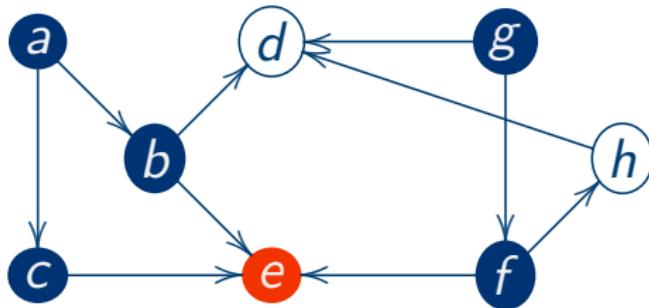


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $v \in V$

- **direto**: vértices **alcançáveis** de v , com caminho maior ou igual a zero
- **inverso**: vértices que **alcançam** v , com caminho maior ou igual a zero



Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $X \subseteq V$

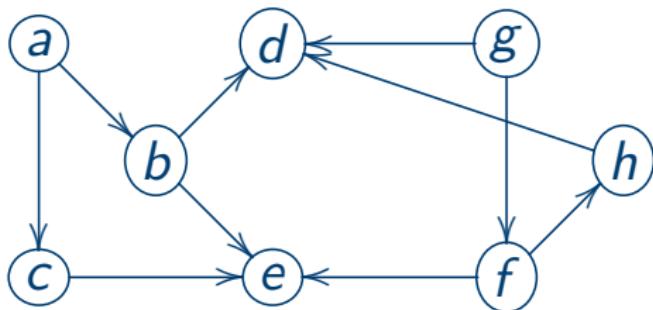
- direto : vértices **alcançáveis** de cada vértice de X , com caminho maior ou igual a zero

Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $X \subseteq V$

- direto : vértices **alcançáveis** de cada vértice de X , com caminho maior ou igual a zero

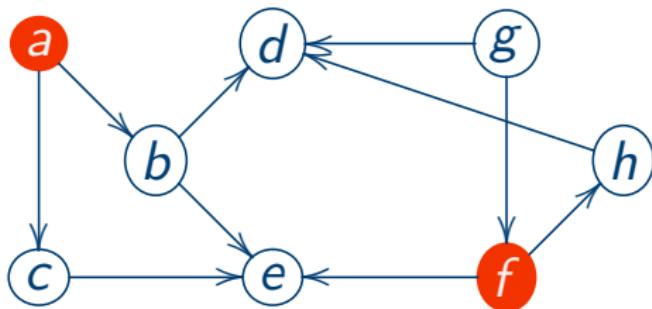


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $X \subseteq V$

- direto : vértices **alcançáveis** de cada vértice de X , com caminho maior ou igual a zero

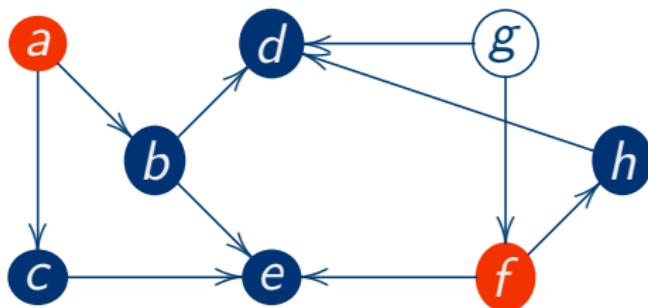


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $X \subseteq V$

- direto : vértices **alcançáveis** de cada vértice de X , com caminho maior ou igual a zero

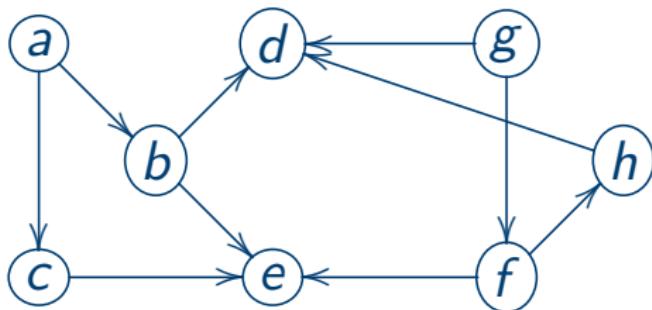


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $X \subseteq V$

- ▶ **direto**: vértices **alcançáveis** de cada vértice de X , com caminho maior ou igual a zero
- ▶ **inverso**: vértices que **alcançam** algum vértice de X , com caminho maior ou igual a zero

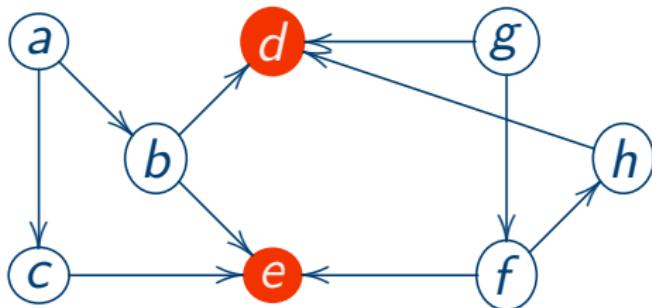


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $X \subseteq V$

- **direto**: vértices **alcançáveis** de cada vértice de X , com caminho maior ou igual a zero
- **inverso**: vértices que **alcançam** algum vértice de X , com caminho maior ou igual a zero

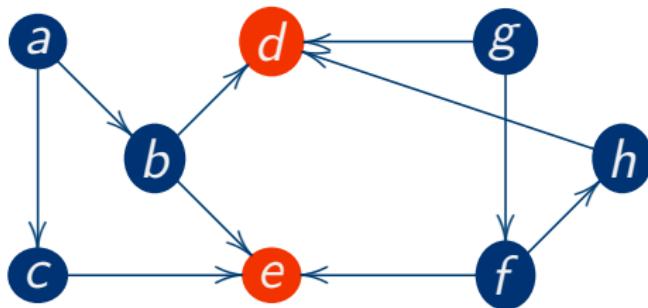


Fecho transitivo em grafos

$G = (V, E)$ é um grafo dirigido

Fecho transitivo de $X \subseteq V$

- **direto**: vértices **alcançáveis** de cada vértice de X , com caminho maior ou igual a zero
- **inverso**: vértices que **alcançam** algum vértice de X , com caminho maior ou igual a zero

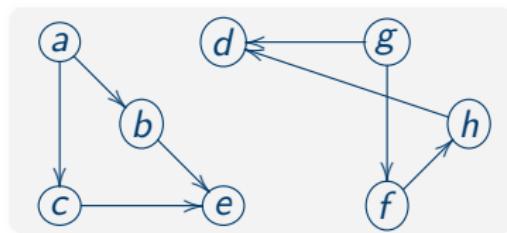


Conectividade em grafos direcionados

Um grafo é **não-conexo** ou desconexo se nem todo par de vértices é unido por uma cadeia

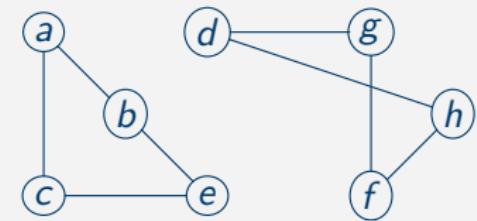
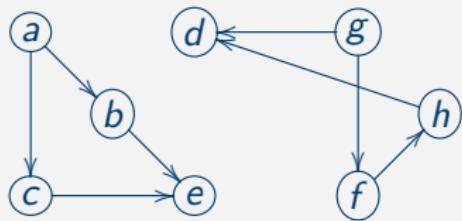
Conectividade em grafos direcionados

Um grafo é **não-conexo** ou desconexo se nem todo par de vértices é unido por uma cadeia



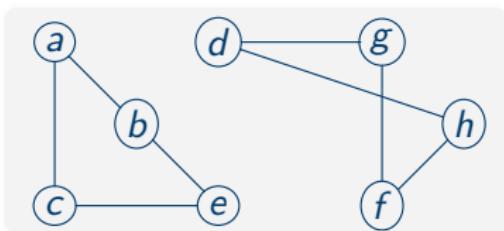
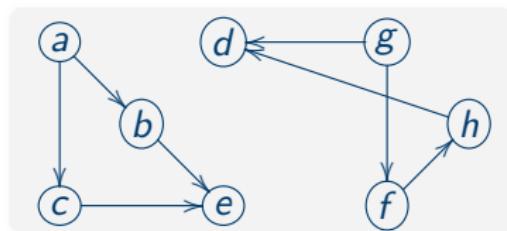
Conectividade em grafos direcionados

Um grafo é **não-conexo** ou desconexo se nem todo par de vértices é unido por uma cadeia



Conectividade em grafos direcionados

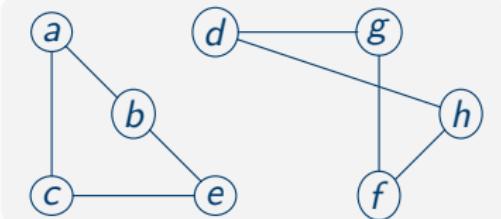
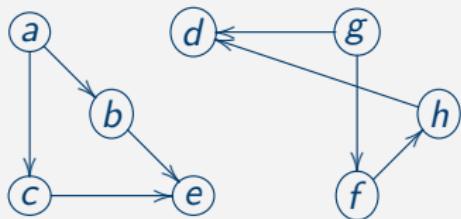
Um grafo é **não-conexo** ou desconexo se nem todo par de vértices é unido por uma cadeia



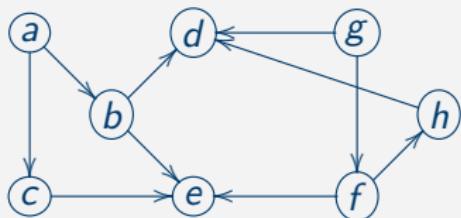
Um grafo é **simplesmente conexo** ou s-conexo se todo par de vértices é unido por pelo menos uma cadeia

Conectividade em grafos direcionados

Um grafo é **não-conexo** ou desconexo se nem todo par de vértices é unido por uma cadeia

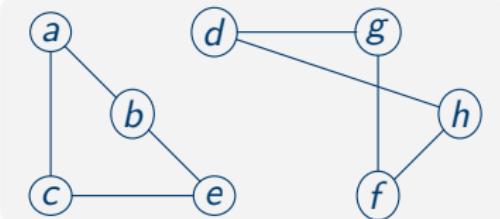
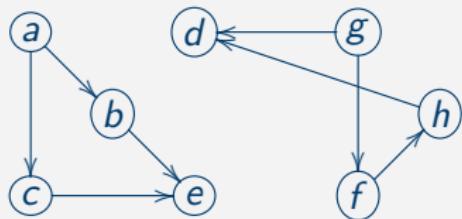


Um grafo é **simplesmente conexo** ou s-conexo se todo par de vértices é unido por pelo menos uma cadeia

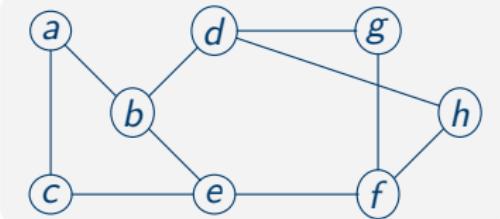
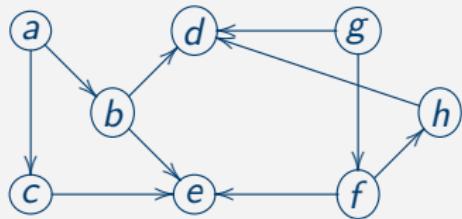


Conectividade em grafos direcionados

Um grafo é **não-conexo** ou desconexo se nem todo par de vértices é unido por uma cadeia



Um grafo é **simplesmente conexo** ou s-conexo se todo par de vértices é unido por pelo menos uma cadeia

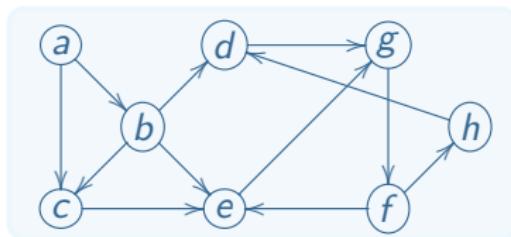


Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro

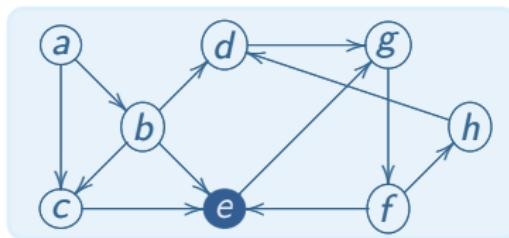
Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro



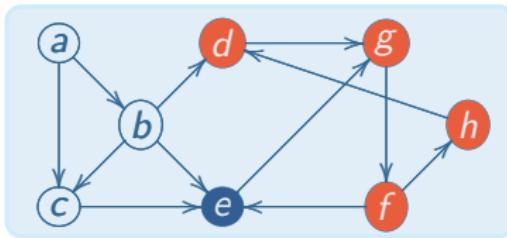
Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro



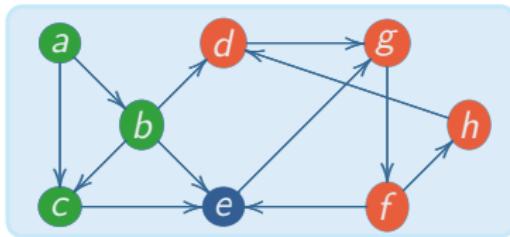
Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro



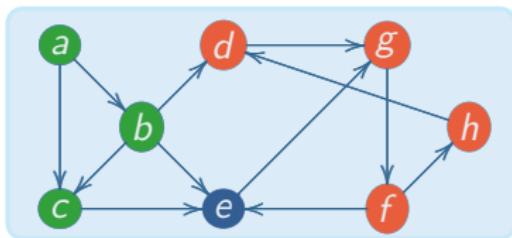
Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro



Conectividade em grafos direcionados

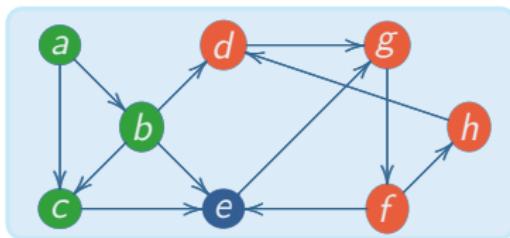
Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro



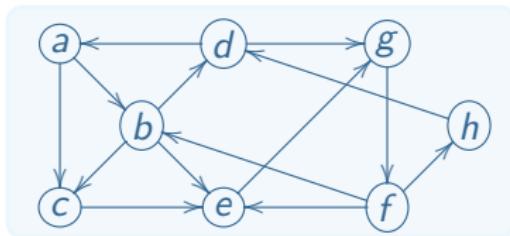
Um grafo é **fortemente conexo** ou f-conexo se todos os vértices são mutuamente alcançáveis

Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro

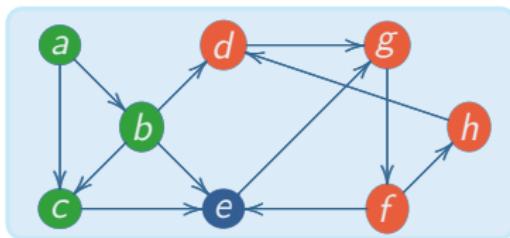


Um grafo é **fortemente conexo** ou f-conexo se todos os vértices são mutuamente alcançáveis

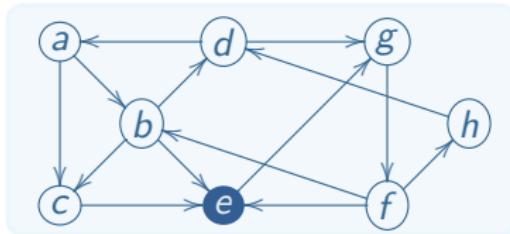


Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro

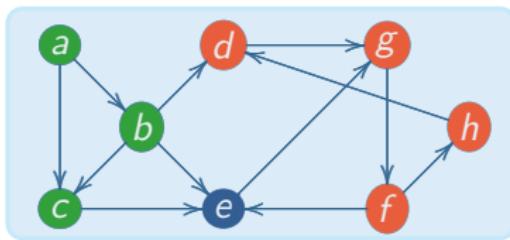


Um grafo é **fortemente conexo** ou f-conexo se todos os vértices são mutuamente alcançáveis

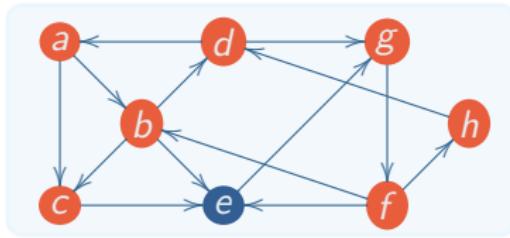


Conectividade em grafos direcionados

Um grafo é **semi-fortemente conexo** ou sf-conexo se para todo par de vértice, pelo menos um deles é alcançável a partir do outro



Um grafo é **fortemente conexo** ou f-conexo se todos os vértices são mutuamente alcançáveis



Categorias de conectividade

- ▶ C_0 : Grafos desconexos não s-conexos
- ▶ C_1 : Grafos s-conexos não sf-conexos
- ▶ C_2 : Grafos sf-conexos não f-conexos
- ▶ C_3 : Grafos f-conexos

Propriedade

- ▶ Se $G \in C_0$ então $C(G) \in C_3$
- ▶ Se $G \in C_1$ então $C(G) \notin C_0$
- ▶ Se $G \in C_2$ então $C(G) \notin C_0$
- ▶ Se $G \in C_3$ então $C(G) \in Ci$ ($i = 0,1,2,3$)

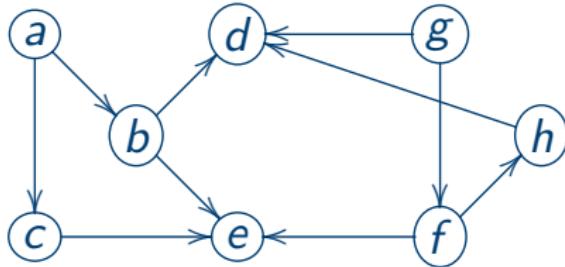
Atingibilidade

Seja $G = (V, E)$ um grafo dirigido.

Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $B \subseteq V$ é uma **base** de G se

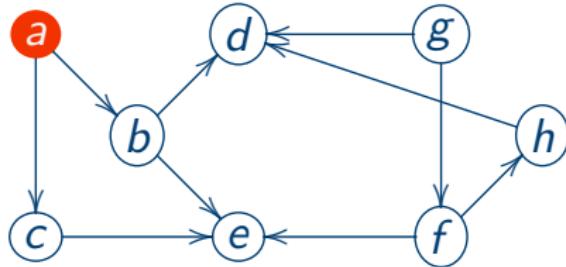
- ▶ não há caminho entre vértices de B
- ▶ todo vértice não pertencente a B pode ser **atingido** por algum vértice de B



Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $B \subseteq V$ é uma **base** de G se

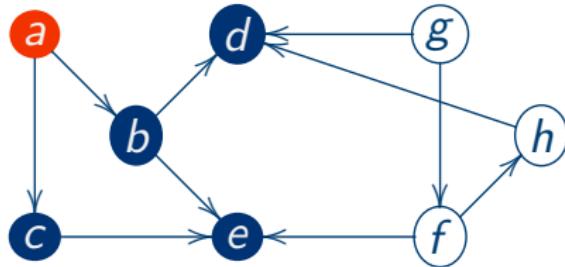
- ▶ não há caminho entre vértices de B
- ▶ todo vértice não pertencente a B pode ser atingido por algum vértice de B



Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $B \subseteq V$ é uma **base** de G se

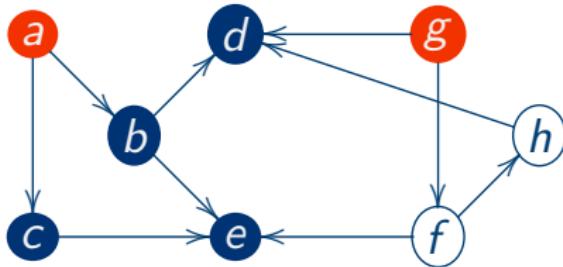
- ▶ não há caminho entre vértices de B
- ▶ todo vértice não pertencente a B pode ser atingido por algum vértice de B



Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $B \subseteq V$ é uma **base** de G se

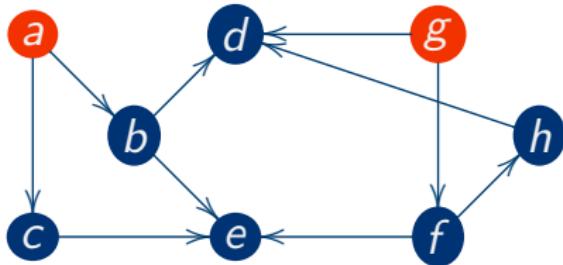
- ▶ não há caminho entre vértices de B
- ▶ todo vértice não pertencente a B pode ser atingido por algum vértice de B



Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $B \subseteq V$ é uma **base** de G se

- ▶ não há caminho entre vértices de B
- ▶ todo vértice não pertencente a B pode ser atingido por algum vértice de B



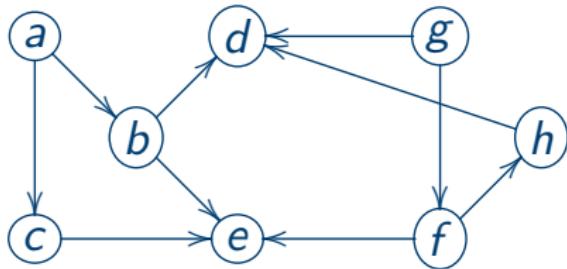
Atingibilidade

Seja $G = (V, E)$ um grafo dirigido.

Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $A \subseteq V$ é uma anti-base de G se

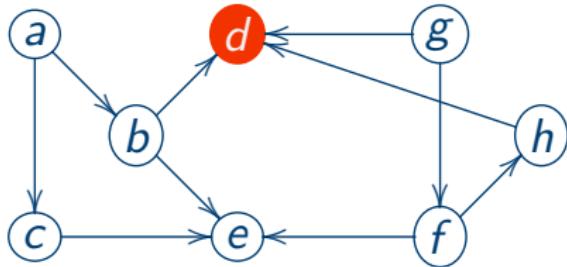
- ▶ não há caminho entre vértices de A
- ▶ todo vértice não pertencente a A pode atingir A por um caminho.



Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $A \subseteq V$ é uma anti-base de G se

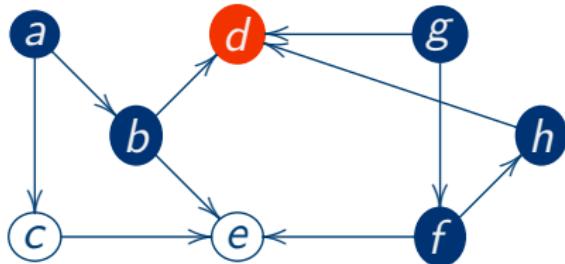
- ▶ não há caminho entre vértices de A
- ▶ todo vértice não pertencente a A pode atingir A por um caminho.



Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $A \subseteq V$ é uma anti-base de G se

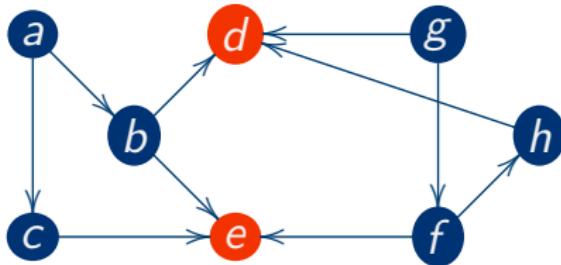
- ▶ não há caminho entre vértices de A
- ▶ todo vértice não pertencente a A pode atingir A por um caminho.



Atingibilidade

Seja $G = (V, E)$ um grafo dirigido. Um subconjunto $A \subseteq V$ é uma anti-base de G se

- ▶ não há caminho entre vértices de A
- ▶ todo vértice não pertencente a A pode atingir A por um caminho.



Atingibilidade

Sejam B uma base de G e A uma antibase de G

Raiz *Se B for um conjunto unitário, então dizemos que B é a RAIZ de G*

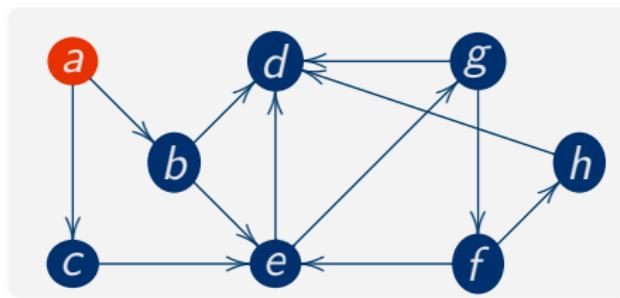
Anti-raiz *Se A for um conjunto unitário, então dizemos que A é a ANTIRAI Z de G*

Atingibilidade

Sejam B uma base de G e A uma antibase de G

Raiz *Se B for um conjunto unitário, então dizemos que B é a RAIZ de G*

Anti-raiz *Se A for um conjunto unitário, então dizemos que A é a ANTIRAI Z de G*

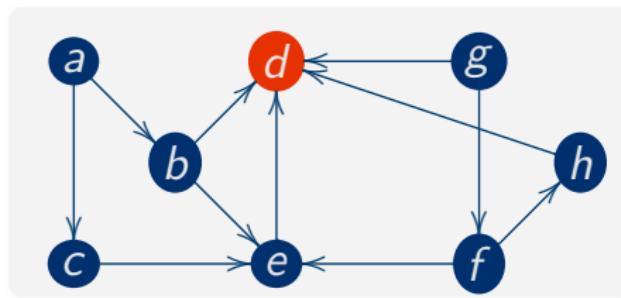


Atingibilidade

Sejam B uma base de G e A uma antibase de G

Raiz *Se B for um conjunto unitário, então dizemos que B é a RAIZ de G*

Anti-raiz *Se A for um conjunto unitário, então dizemos que A é a ANTIRAI Z de G*

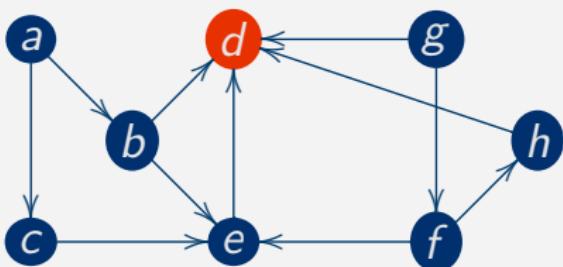
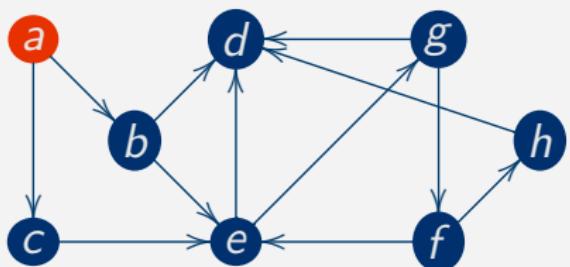


Atingibilidade

Sejam B uma base de G e A uma antibase de G

Raiz *Se B for um conjunto unitário, então dizemos que B é a RAIZ de G*

Anti-raiz *Se A for um conjunto unitário, então dizemos que A é a ANTIRAI Z de G*



Questions?

Connectivity
– Connectivity –

Teoria dos Grafos e Computabilidade

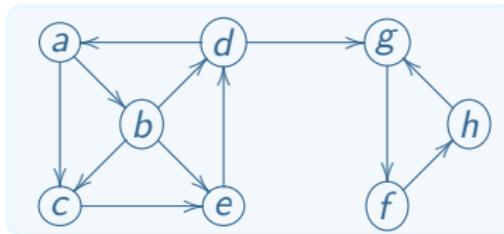
— Strongly connected components —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Componentes F-Conexas

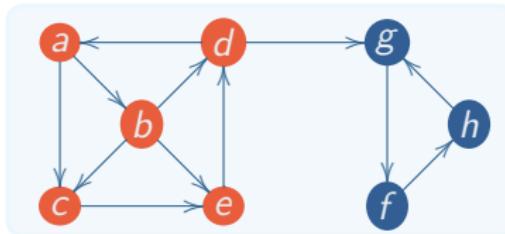
- ▶ Seja $G = (V, E)$ um grafo dirigido



Componentes F-Conexas

- ▶ Seja $G = (V, E)$ um grafo dirigido
- ▶ Considere a seguinte partição em V , em que cada S_i é f-conexo:

$$S = \{S_i | S_i \subseteq V, S_i \cap S_j = \emptyset, i, j = 1, \dots, n\}$$

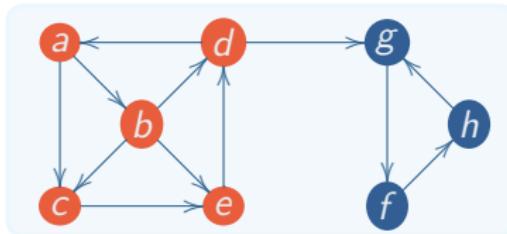


Componentes F-Conexas

- ▶ Seja $G = (V, E)$ um grafo dirigido
- ▶ Considere a seguinte partição em V , em que cada S_i é f-conexo:

$$S = \{S_i | S_i \subseteq V, S_i \cap S_j = \emptyset, i, j = 1, \dots, n\}$$

- ▶ Os elementos $S_i \in S$ são chamados de componentes f-conexas de G

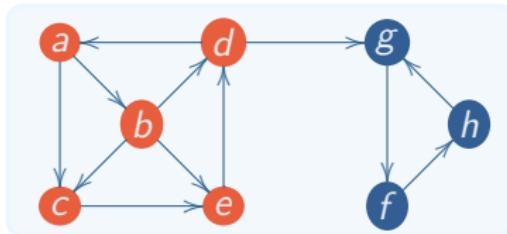


Componentes F-Conexas

- ▶ Seja $G = (V, E)$ um grafo dirigido
- ▶ Considere a seguinte partição em V , em que cada S_i é f-conexo:

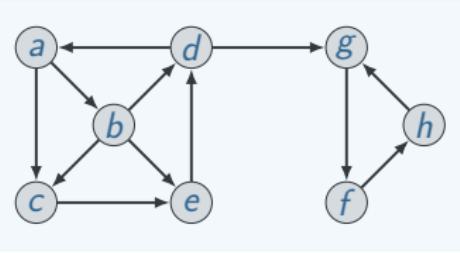
$$S = \{S_i | S_i \subseteq V, S_i \cap S_j = \emptyset, i, j = 1, \dots, n\}$$

- ▶ Os elementos $S_i \in S$ são chamados de componentes f-conexas de G
- ▶ Se G for f-conexo, então $S = V$



Strongly connected components

- Let $G = (V, E)$ be a directed graph

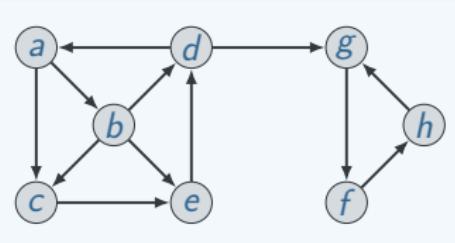


Strongly connected components

- ▶ Let $G = (V, E)$ be a directed graph
- ▶ Let S be a **partition** of V in which

$$S = \{S_i | S_i \subseteq V, S_i \cap S_j = \emptyset, i, j = 1, \dots, n\}$$

$$\bigcup_{i=1, \dots, n} S_i = V$$



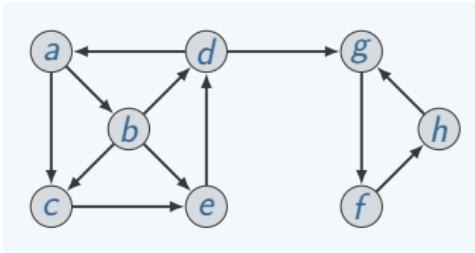
Strongly connected components

- ▶ Let $G = (V, E)$ be a directed graph
- ▶ Let S be a **partition** of V in which

$$S = \{S_i | S_i \subseteq V, S_i \cap S_j = \emptyset, i, j = 1, \dots, n\}$$

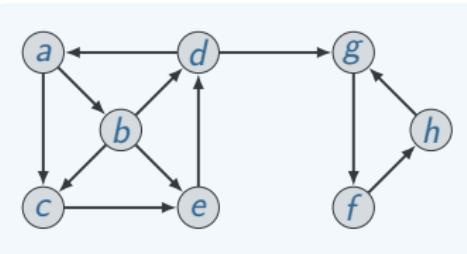
$$\bigcup_{i=1, \dots, n} S_i = V$$

- ▶ The induced subgraph of G by $S_i \in S$ is so-called **strongly connected component** if the subgraph is strongly connected.



Strongly connected components

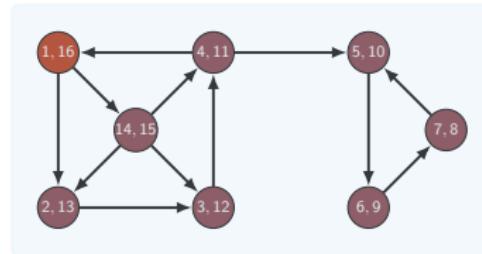
Let $G = (V, E)$ be a directed graph



Strongly connected components

Let $G = (V, E)$ be a directed graph

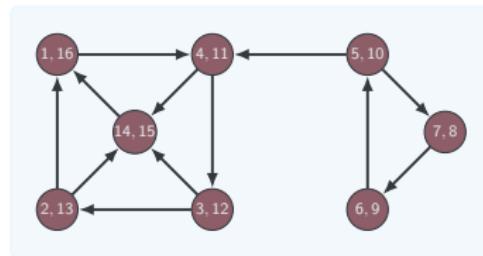
- ▶ Compute a **depth-first search** from a selected vertex indicating the discovery and last times



Strongly connected components

Let $G = (V, E)$ be a directed graph

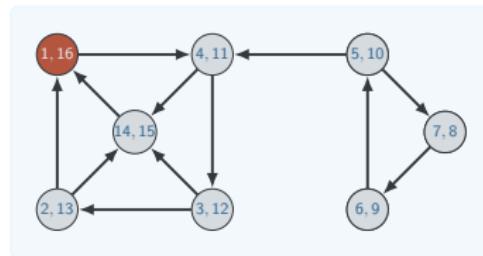
- ▶ Compute a **depth-first search** from a selected vertex indicating the discovery and last times
- ▶ Compute a **symmetric graph** $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.



Strongly connected components

Let $G = (V, E)$ be a directed graph

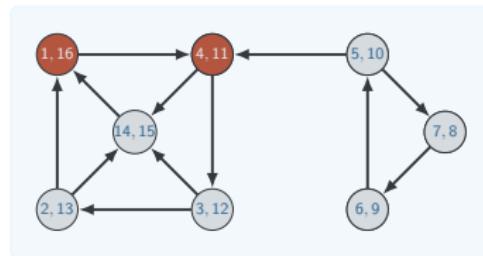
- ▶ Compute a depth-first search from a selected vertex indicating the discovery and last times
- ▶ Compute a symmetric graph $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a depth-first search in the inversal order of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

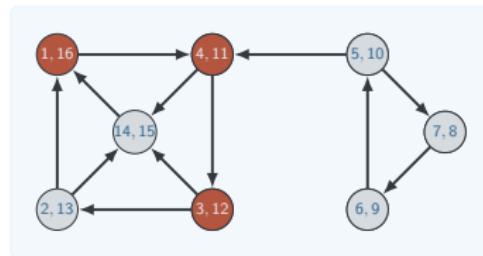
- ▶ Compute a **depth-first search** from a selected vertex indicating the discovery and last times
- ▶ Compute a **symmetric graph** $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a **depth-first search** in the **inversal order** of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

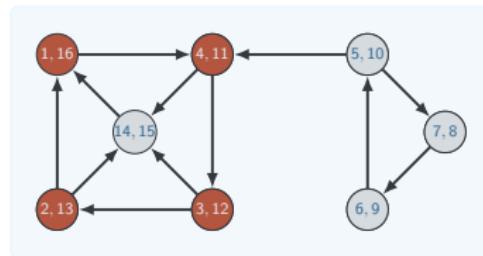
- ▶ Compute a depth-first search from a selected vertex indicating the discovery and last times
- ▶ Compute a symmetric graph $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a depth-first search in the inversal order of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

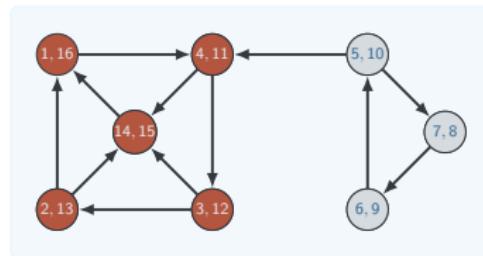
- ▶ Compute a **depth-first search** from a selected vertex indicating the discovery and last times
- ▶ Compute a **symmetric graph** $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a **depth-first search** in the **inversal order** of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

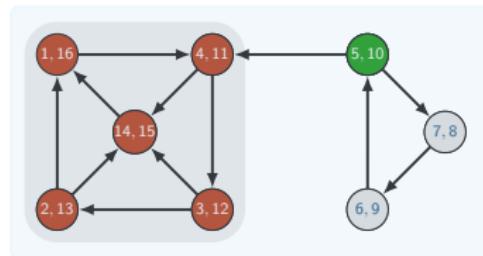
- ▶ Compute a depth-first search from a selected vertex indicating the discovery and last times
- ▶ Compute a symmetric graph $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a depth-first search in the inversal order of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

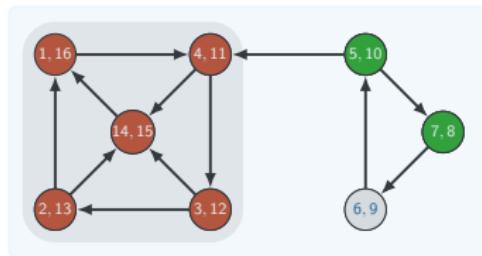
- ▶ Compute a depth-first search from a selected vertex indicating the discovery and last times
- ▶ Compute a symmetric graph $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a depth-first search in the inverse order of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

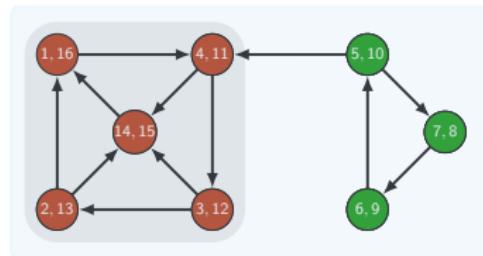
- ▶ Compute a depth-first search from a selected vertex indicating the discovery and last times
- ▶ Compute a symmetric graph $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a depth-first search in the inversal order of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

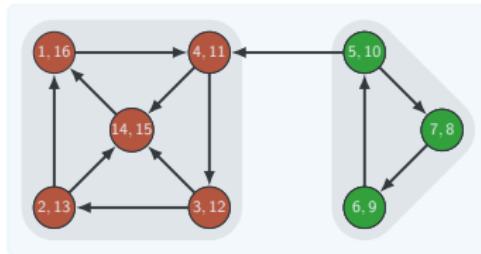
- ▶ Compute a depth-first search from a selected vertex indicating the discovery and last times
- ▶ Compute a symmetric graph $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a depth-first search in the inverse order of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

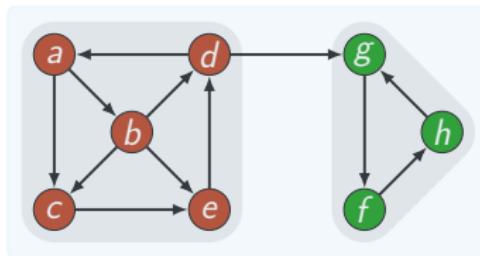
- ▶ Compute a **depth-first search** from a selected vertex indicating the discovery and last times
- ▶ Compute a **symmetric graph** $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a **depth-first search** in the **inversal order** of last time



Strongly connected components

Let $G = (V, E)$ be a directed graph

- ▶ Compute a depth-first search from a selected vertex indicating the discovery and last times
- ▶ Compute a symmetric graph $G^- = (V, E')$ in which there is an edge $e' \in E'$ if for each $e = (u, v) \in E$ then $e' = (v, u)$.
- ▶ Compute a depth-first search in the inversal order of last time





Teoria dos Grafos e Computabilidade

— Graph cut —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

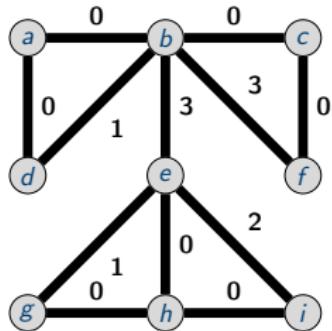
Pontifical Catholic University of Minas Gerais – PUC Minas

Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).

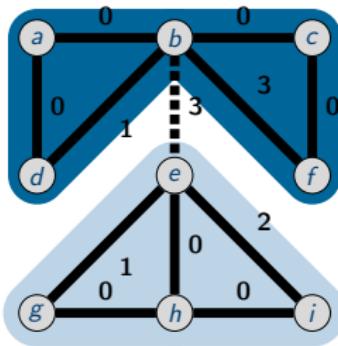
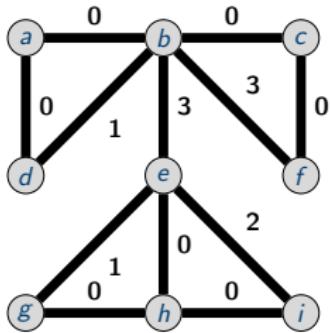
Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).



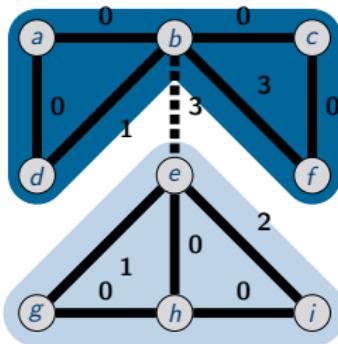
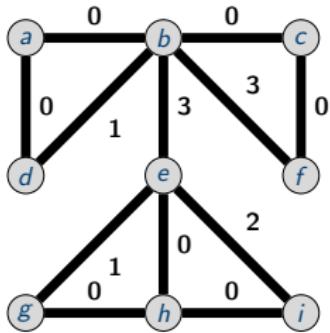
Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).



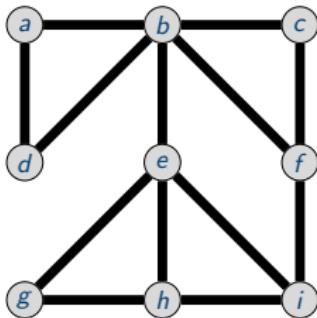
Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



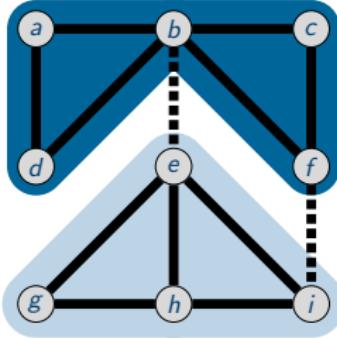
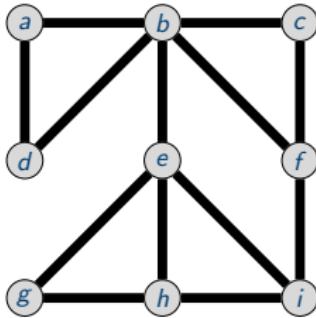
Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



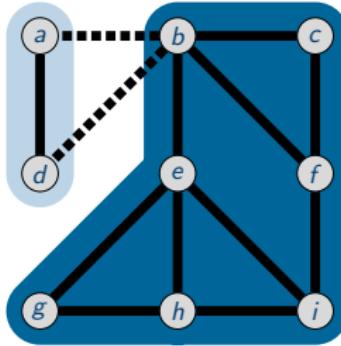
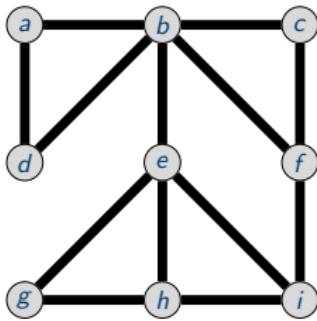
Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



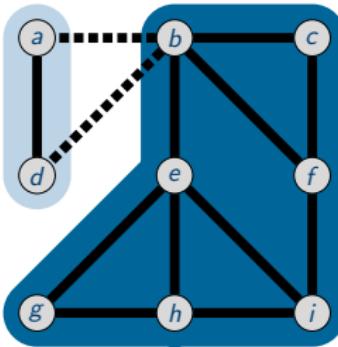
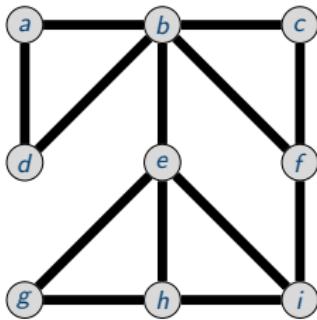
Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



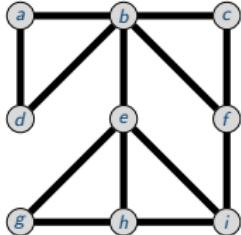
Graph Cuts

- ▶ A **cut** (or cut-set) in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.
- ▶ $\text{cut}(S)$ is a cut because deleting the edges in $\text{cut}(S)$ disconnects S from $V - S$.



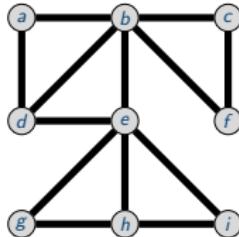
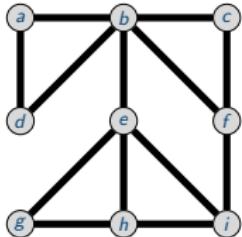
Connectivity and separability

- ▶ Edge-connectivity $\lambda(G)$ corresponds to the smallest number of edges of the graph in which their removal will disconnect the graph;



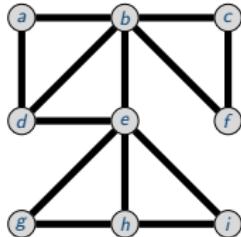
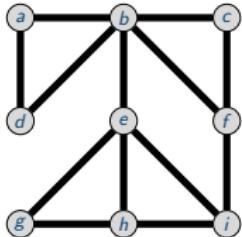
Connectivity and separability

- ▶ Edge-connectivity $\lambda(G)$ corresponds to the smallest number of edges of the graph in which their removal will disconnect the graph;



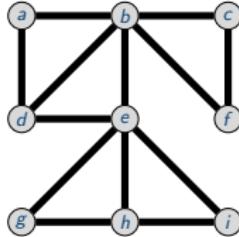
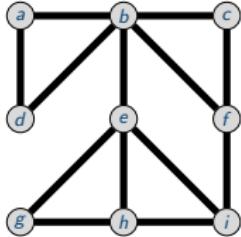
Connectivity and separability

- ▶ Edge-connectivity $\lambda(G)$ corresponds to the smallest number of edges of the graph in which their removal will disconnect the graph;



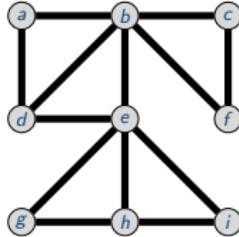
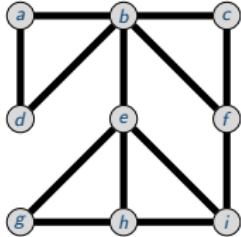
Connectivity and separability

- ▶ Edge-connectivity $\lambda(G)$ corresponds to the smallest number of edges of the graph in which their removal will disconnect the graph;
- ▶ Vertex-connectivity $K(G)$ corresponds to the smallest number of vertices in which their removal will disconnect the graph;



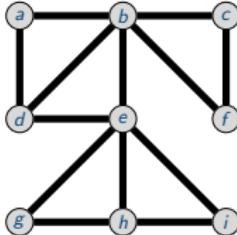
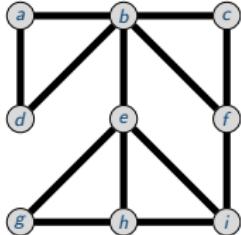
Connectivity and separability

- ▶ Edge-connectivity $\lambda(G)$ corresponds to the smallest number of edges of the graph in which their removal will disconnect the graph;
- ▶ Vertex-connectivity $K(G)$ corresponds to the smallest number of vertices in which their removal will disconnect the graph;



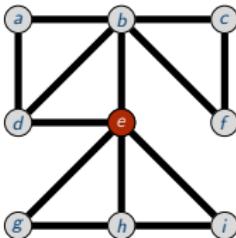
Connectivity and separability

- ▶ Edge-connectivity $\lambda(G)$ corresponds to the smallest number of edges of the graph in which their removal will disconnect the graph;
- ▶ Vertex-connectivity $K(G)$ corresponds to the smallest number of vertices in which their removal will disconnect the graph;
- ▶ K -connected graph is a graph with vertex-connectivity equal to K ;
- ▶ Separable-graph is a graph with vertex-connectivity equal to 1.



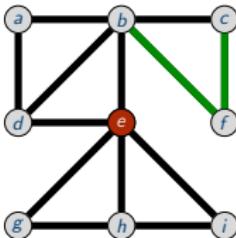
Connectivity and separability

- The vertex that disconnects the separable-graph is called articulation point (or cut-vertex)



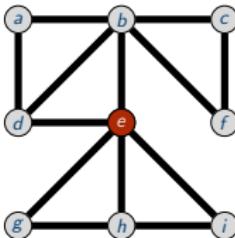
Connectivity and separability

- ▶ The vertex that disconnects the separable-graph is called **articulation point** (or cut-vertex)
- ▶ The edge-connectivity of a graph is smaller than or equal to the **smallest degree** between all vertex degree of the graph;



Connectivity and separability

- ▶ The vertex that disconnects the separable-graph is called **articulation point** (or cut-vertex)
- ▶ The edge-connectivity of a graph is smaller than or equal to the **smallest degree** between all vertex degree of the graph;
- ▶ The vertex-connectivity is smaller than or equal to the edge-connectivity;
- ▶ Let $\delta(G)$ be the smallest vertex degree of G . So,
 $K(G) \leq \lambda(G) \leq \delta(G)$.



Questions?

Connectivity
– Graph cut –

Robustness of a network – an example

Exemplo 1

Consider a network (transportation, computers, and so on) represented by a graph. How to measure the **robustness** of this network?

Robustness of a network – an example

Exemplo 1

Consider a network (transportation, computers, and so on) represented by a graph. How to measure the **robustness** of this network?

The robustness of a graph is based on the cut-sets.

Vulnerability of a network – an example

Exemplo 2

Let a private network containing eight computers. Consider that you have 16 lines to connect these computers. How to organize this network in order to decrease the vulnerability as low as possible?

Vulnerability of a network – an example

Exemplo 2

Let a private network containing eight computers. Consider that you have 16 lines to connect these computers. How to organize this network in order to decrease the vulnerability as low as possible?

How to model this problem as a graph problem?



Teoria dos Grafos e Computabilidade

— Shortest path —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Some graph fundamentals —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Graphs

- ▶ Model pairwise relationships (edges) between objects (nodes or vertices).
- ▶ **Undirected graph** $G = (V, E)$: set V of nodes and set E of edges, where $E \subseteq V \times V$. Elements of E are unordered pairs.
- ▶ **Directed graph** $G = (V, E)$: set V of nodes and set E of edges, where $E \subseteq V \times V$. Elements of E are ordered pairs.

Applications of Graphs

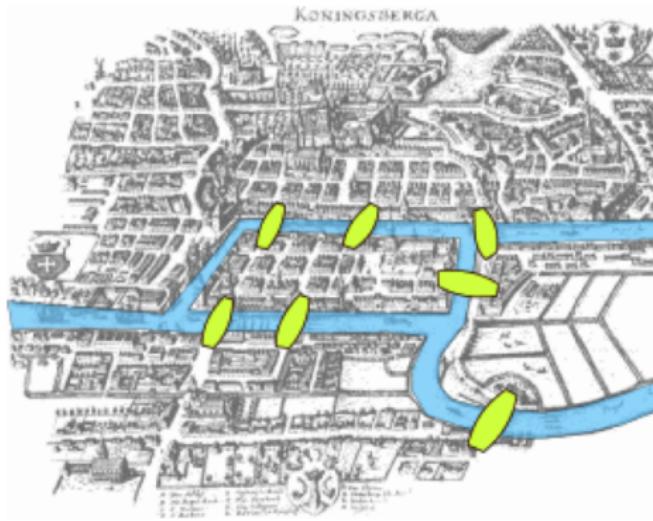
- ▶ Useful in a large number of applications:

Applications of Graphs

- ▶ Useful in a large number of applications: computer networks, the World Wide Web, ecology (food webs), social networks, software systems, job scheduling, VLSI circuits, cellular networks, ...
- ▶ Problems involving graphs have a rich history dating back to Euler.

Applications of Graphs

- ▶ Useful in a large number of applications: computer networks, the World Wide Web, ecology (food webs), social networks, software systems, job scheduling, VLSI circuits, cellular networks, ...
- ▶ Problems involving graphs have a rich history dating back to Euler.



Questions?

Shortest path

- Some graph fundamentals –

Teoria dos Grafos e Computabilidade

— Shortest Path Problem —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Shortest Path Problem

- ▶ $G = (V, E)$ is a connected directed graph. Each edge e has a length $l_e \geq 0$.
- ▶ V has n nodes and E has m edges.
- ▶ Length of a path P is the sum of lengths of the edges in P .
- ▶ Goal is to determine the shortest path from some start node s to each node in V .
- ▶ Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

Shortest Path Problem

- $G = (V, E)$ is a connected directed graph. Each edge e has a length $l_e \geq 0$.
- V has n nodes and E has m edges.
- Length of a path P is the sum of lengths of the edges in P .
- Goal is to determine the shortest path from some start node s to each node in V .
- Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

SHORTEST PATHS

INSTANCE A directed graph $G = (V, E)$, a function $l : E \rightarrow \mathbb{R}^+$, and a node $s \in V$

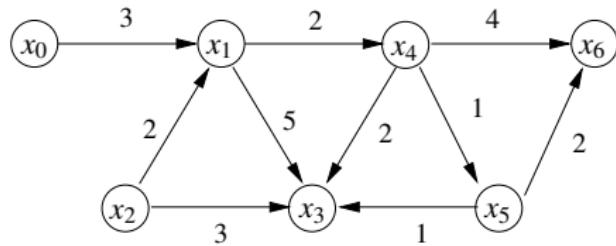
SOLUTION A set $\{P_u, u \in V\}$, where P_u is the shortest path in G from s to u .

Shortest paths

- ▶ Let $N = (G, W)$ be a positive length graph, let $x \in V$
- ▶ We define the map $L_x : V \rightarrow \mathbb{R} \cup \{\infty\}$ by:

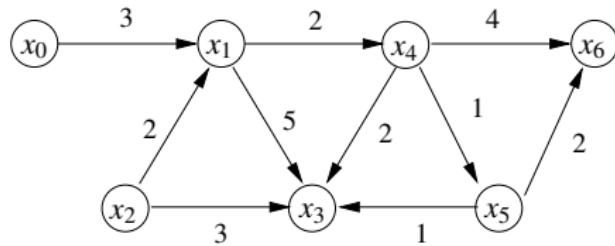
$$L_x(y) = \begin{cases} \text{the length of a shortest path from } x \text{ to } y, \text{ if such path exists} \\ \infty, \text{ otherwise} \end{cases}$$

Illustration: the map L_x



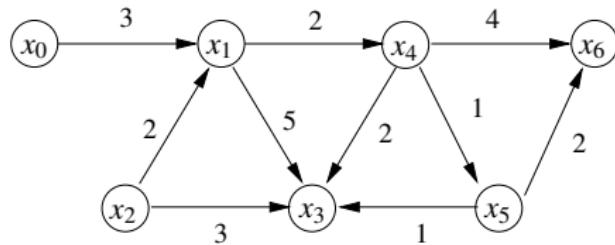
$$\frac{y =}{L_{x_0}(y) =} \begin{array}{c|ccccccc} & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ \hline \end{array}$$

Illustration: the map L_x



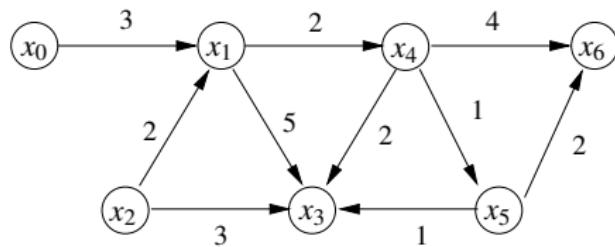
$$\begin{array}{c} y = \\ \hline L_{x_0}(y) = \end{array} \left| \begin{array}{ccccccc} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0 & & & & & & \end{array} \right.$$

Illustration: the map L_x



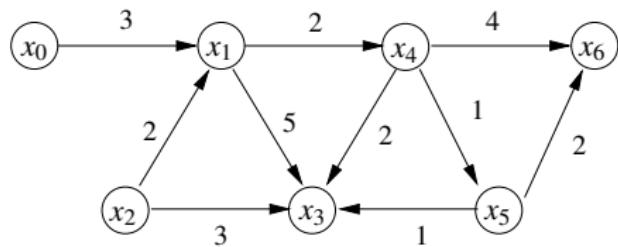
$$\begin{array}{c} y = \\ L_{x_0}(y) = \left| \begin{array}{ccccccc} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0 & 3 & & & & & \end{array} \right| \end{array}$$

Illustration: the map L_x



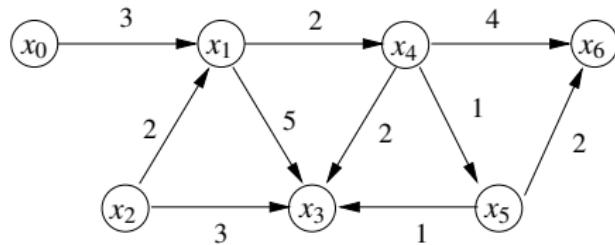
$$y = \begin{array}{c|ccccccc} & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ \hline L_{x_0}(y) = & 0 & 3 & \infty & & & & \end{array}$$

Illustration: the map L_x



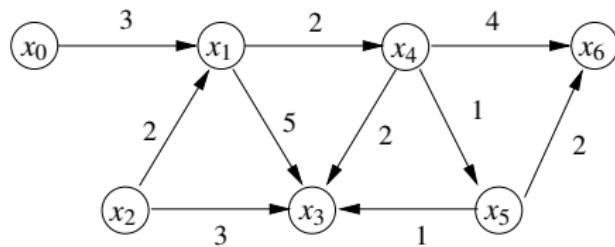
$$y = \begin{array}{c|ccccccc} & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ \hline L_{x_0}(y) = & 0 & 3 & \infty & 7 & & & \end{array}$$

Illustration: the map L_x



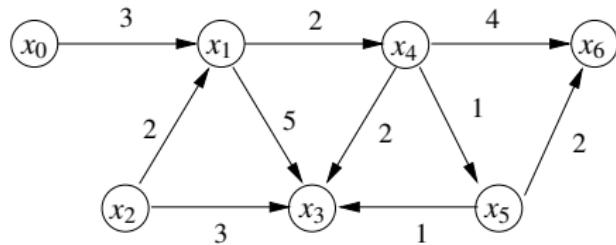
$$\frac{y =}{L_{x_0}(y) =} \begin{array}{c|ccccccc} & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ \hline 0 & 0 & 3 & \infty & 7 & 5 & & \end{array}$$

Illustration: the map L_x



$$y = \frac{L_{x_0}(y)}{| \begin{array}{ccccccc} x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ 0 & 3 & \infty & 7 & 5 & 6 \end{array} |}$$

Illustration: the map L_x



$$y = \frac{L_{x_0}(y)}{L_{x_0}(y) = \begin{array}{c|ccccccc} & x_0 & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ \hline 0 & 0 & 3 & \infty & 7 & 5 & 6 & 8 \end{array}}$$

Questions?

Shortest path

– Shortest Path Problem –

Teoria dos Grafos e Computabilidade

— Algorithms for Single Source Shortest Path —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Problems

1. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and two vertices x and y in V
 - ▶ Find a shortest path from x to y
 - ▶ Find the length $L_x(y)$ of a shortest path from x to y
2. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and a vertex x in V
 - ▶ Find, for each vertex y in V the length $L_x(y)$ of a shortest path from x to y
3. Given a graph $G = (V, \Gamma)$ and a network (G, ℓ)
 - ▶ Find, for each pair x, y of vertices in V , the length of a shortest path from x to y
4. Having solved problem 2
 - ▶ Solve problem 1

Dijkstra algorithm

1. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and two vertices x and y in V
 - ▶ Find a shortest path from x to y
 - ▶ Find the length $L_x(y)$ of a shortest path from x to y
2. Given a graph $G = (V, \Gamma)$, a network (G, ℓ) and a vertex x in V
 - ▶ Find for each vertex y in V the length $L_x(y)$ of a shortest path from x to y
3. Given a graph $G = (V, \Gamma)$ and a network (G, ℓ)
 - ▶ Find, for each pair x, y of vertices in V , the length of a shortest path from x to y
4. Having solved problem 2
 - ▶ Solve problem 1

Computing the lengths of shortest paths

Algorithm DIJKSTRA (**Data:** A graph $G = (V, \Gamma)$, a network (G, ℓ) , $n = |V|$, $x \in V$;

Result: L_x)

$\bar{S} := \emptyset$;

For each $y \in V$ **Do** $L_x[y] = \infty$; $\bar{S} := \bar{S} \cup \{y\}$;

$L_x[x] := 0$; $k := 0$; $\mu := 0$;

While $k < n$ and $\mu \neq \infty$ **Do**

- ▶ Extract a vertex $y^* \in \bar{S}$ such that $L_x[y^*] = \min\{L_x[y], y \in \bar{S}\}$

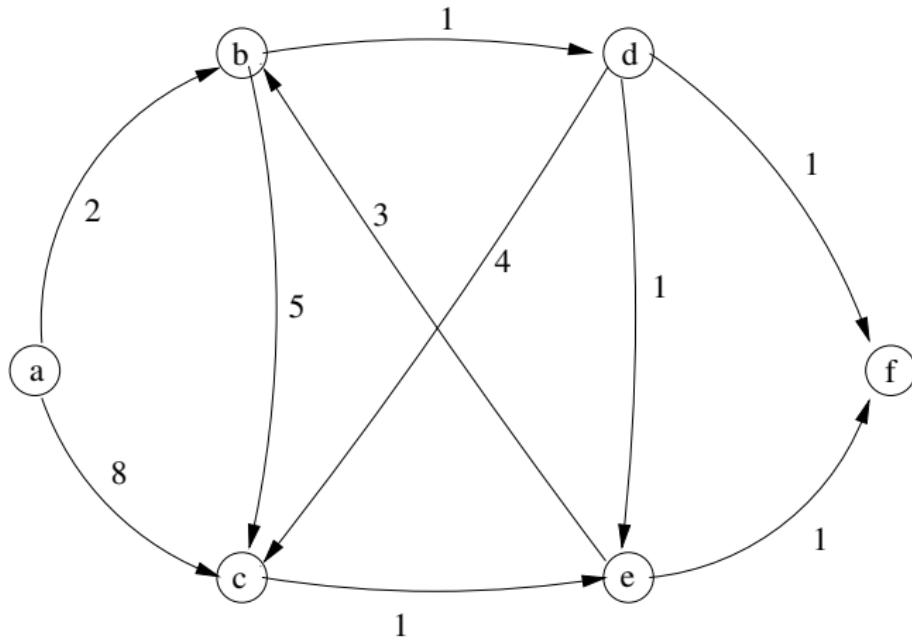
- ▶ $k++$; $\mu := L_x[y^*]$;

- ▶ **For each** $y \in \Gamma(y^*) \cap \bar{S}$ **Do**

- ▶ $L_x[y] := \min\{L_x[y], L_x[y^*] + \ell(y^*, y)\}$;

Computing the lengths of shortest paths

- Exercise. Execute “by hand” Dijkstra algorithm on the following network with $x = a$, and on any positive length network of your choice



Loop invariant of Dijkstra algorithm (# 1)

- ▶ Let $x \in V$ and $\mu \in \mathbb{R}$
- ▶ A subset S of V is called a μ -separating (for x) if the two following conditions hold true:

Loop invariant of Dijkstra algorithm (# 1)

- ▶ Let $x \in V$ and $\mu \in \mathbb{R}$
- ▶ A subset S of V is called a μ -separating (for x) if the two following conditions hold true:
 1. S contains any vertex y such that the length $L_x(y)$ of a shortest path from x to y is less than μ

Loop invariant of Dijkstra algorithm (# 1)

- ▶ Let $x \in V$ and $\mu \in \mathbb{R}$
- ▶ A subset S of V is called a μ -separating (for x) if the two following conditions hold true:
 1. S contains any vertex y such that the length $L_x(y)$ of a shortest path from x to y is less than μ
 2. $\bar{S} = V \setminus S$ contains any vertex y such that the length of a shortest path from x to y is greater than μ

Loop invariant of Dijkstra algorithm (# 2)

- ▶ Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ -separating for x
- ▶ An S -path is a path whose intermediary vertices are all in S

Loop invariant of Dijkstra algorithm (# 2)

- ▶ Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ -separating for x
- ▶ An S -path is a path whose intermediary vertices are all in S
- ▶ The length of a shortest S -path from x to y is denoted by $L_x^S(y)$

Loop invariant of Dijkstra algorithm (# 2)

- ▶ Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ -separating for x
- ▶ An S -path is a path whose intermediary vertices are all in S
- ▶ The length of a shortest S -path from x to y is denoted by $L_x^S(y)$

proof of Dijkstra algorithm

- ▶ Let $y^* \in \bar{S}$ such that $L_x^S(y^*) = \min\{L_x^S(y) \mid y \in \bar{S}\}$

Loop invariant of Dijkstra algorithm (# 2)

- ▶ Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ -separating for x
- ▶ An S -path is a path whose intermediary vertices are all in S
- ▶ The length of a shortest S -path from x to y is denoted by $L_x^S(y)$

proof of Dijkstra algorithm

- ▶ Let $y^* \in \bar{S}$ such that $L_x^S(y^*) = \min\{L_x^S(y) \mid y \in \bar{S}\}$
- ▶ Then, $L_x^S(y^*) = L_x(y^*)$

Loop invariant of Dijkstra algorithm (# 2)

- ▶ Let $x \in V$, let $\mu \in \mathbb{R}$, and let S be a set that is μ -separating for x
- ▶ An S -path is a path whose intermediary vertices are all in S
- ▶ The length of a shortest S -path from x to y is denoted by $L_x^S(y)$

proof of Dijkstra algorithm

- ▶ Let $y^* \in \bar{S}$ such that $L_x^S(y^*) = \min\{L_x^S(y) \mid y \in \bar{S}\}$
- ▶ Then, $L_x^S(y^*) = L_x(y^*)$
- ▶ Thus, $S \cup \{y^*\}$ is a set that is μ' -separating with $\mu' = L_x^S(y^*)$

Computing the lengths of shortest paths

Algorithm DIJKSTRA (**Data:** A graph $G = (V, \Gamma)$, a network (G, ℓ) , $n = |V|$, $x \in V$;

Result: L_x)

$\bar{S} := \emptyset$;

For each $y \in V$ **Do** $L_x[y] = \infty$; $\bar{S} := \bar{S} \cup \{y\}$;

$L_x[x] := 0$; $k := 0$; $\mu := 0$;

While $k < n$ and $\mu \neq \infty$ **Do**

- ▶ Extract a vertex $y^* \in \bar{S}$ such that $L_x[y^*] = \min\{L_x[y], y \in \bar{S}\}$

- ▶ $k++$; $\mu := L_x[y^*]$;

- ▶ **For each** $y \in \Gamma(y^*) \cap \bar{S}$ **Do**

- ▶ $L_x[y] := \min\{L_x[y], L_x[y^*] + \ell(y^*, y)\}$;

Dijkstra's Algorithm

- ▶ Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u .

Dijkstra's Algorithm

- ▶ Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u .
- ▶ Greedily add a node v to S that is closest to s .

Dijkstra's Algorithm

- ▶ Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u .
- ▶ Greedily add a node v to S that is closest to s .

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
 - 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
 - 3 Initially $d[s] = 0$ and $S = s$;
 - 4 **while** $S \neq V$ **do**
 - 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
 - 6 Add v to S and define $d[v] = d'[v]$;
 - 7 **end**
-

Dijkstra's Algorithm

- ▶ Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u .
- ▶ Greedily add a node v to S that is closest to s .

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

```
1 Let  $S$  be the set of explored nodes;  
2 foreach  $u \in S$  do store distance  $d[u] = \infty$ ;  
3 Initially  $d[s] = 0$  and  $S = s$ ;  
4 while  $S \neq V$  do  
5   Select a node  $v \notin S$  with at least one edge from  $S$  for which  
       $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$  is as small as possible;  
6   Add  $v$  to  $S$  and define  $d[v] = d'[v]$ ;  
7 end
```

Dijkstra's Algorithm

- ▶ Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u .
- ▶ Greedily add a node v to S that is closest to s .

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
 - 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
 - 3 Initially $d[s] = 0$ and $S = s$;
 - 4 **while** $S \neq V$ **do**
 - 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
 - 6 Add v to S and define $d[v] = d'[v]$;
 - 7 **end**
-

Dijkstra's Algorithm

- ▶ Maintain a set S of explored nodes: for each node $u \in S$, we have determined the length $d(u)$ of the shortest path from s to u .
- ▶ Greedily add a node v to S that is closest to s .

Algorithm: Shortest path algorithm – Dijkstra

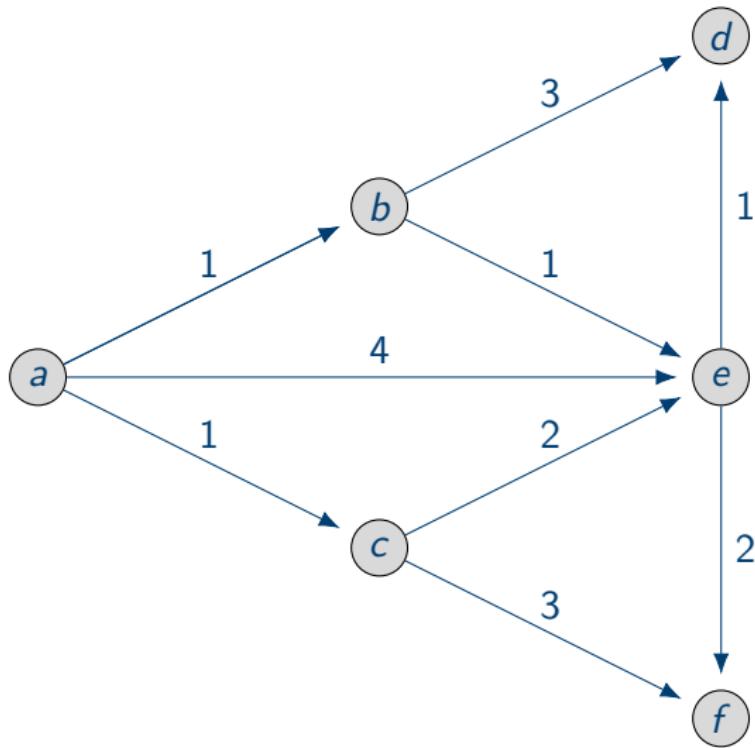
input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

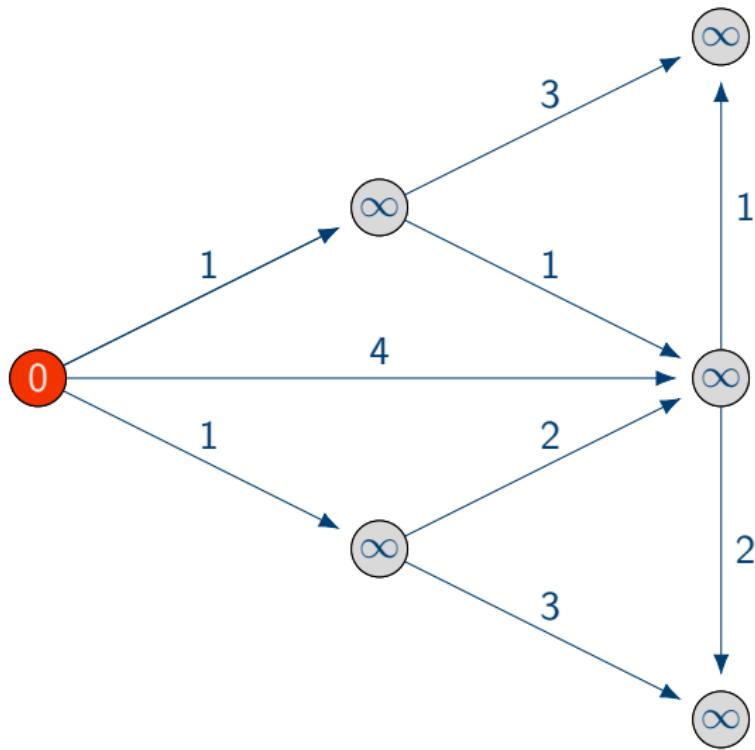
```
1 Let  $S$  be the set of explored nodes;  
2 foreach  $u \in S$  do store distance  $d[u] = \infty$ ;  
3 Initially  $d[s] = 0$  and  $S = s$ ;  
4 while  $S \neq V$  do  
5   Select a node  $v \notin S$  with at least one edge from  $S$  for which  
       $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$  is as small as possible;  
6   Add  $v$  to  $S$  and define  $d[v] = d'[v]$ ;  
7 end
```

- ▶ Can modify algorithm to compute the shortest paths themselves: record the predecessor u that minimizes $d'(v)$.

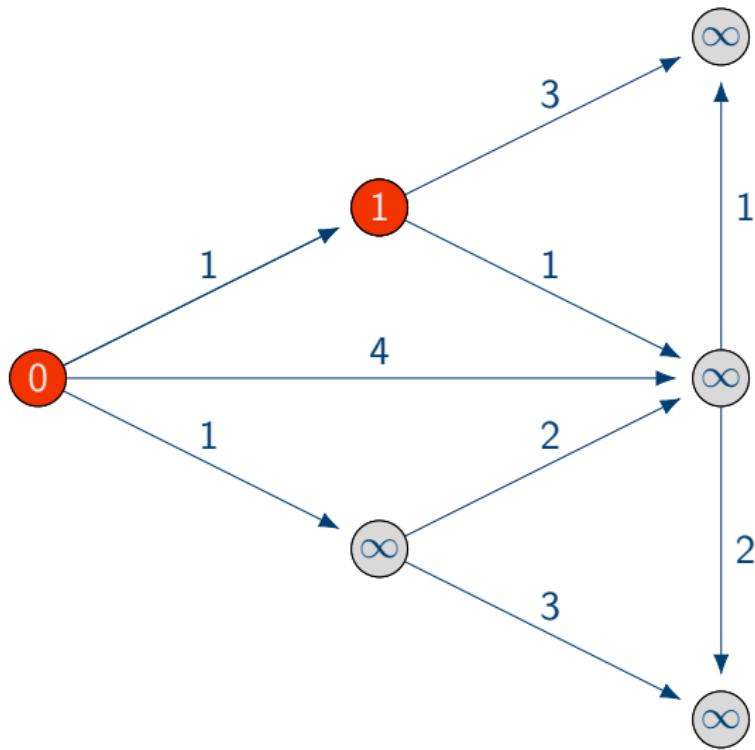
Example of Dijkstra's Algorithm



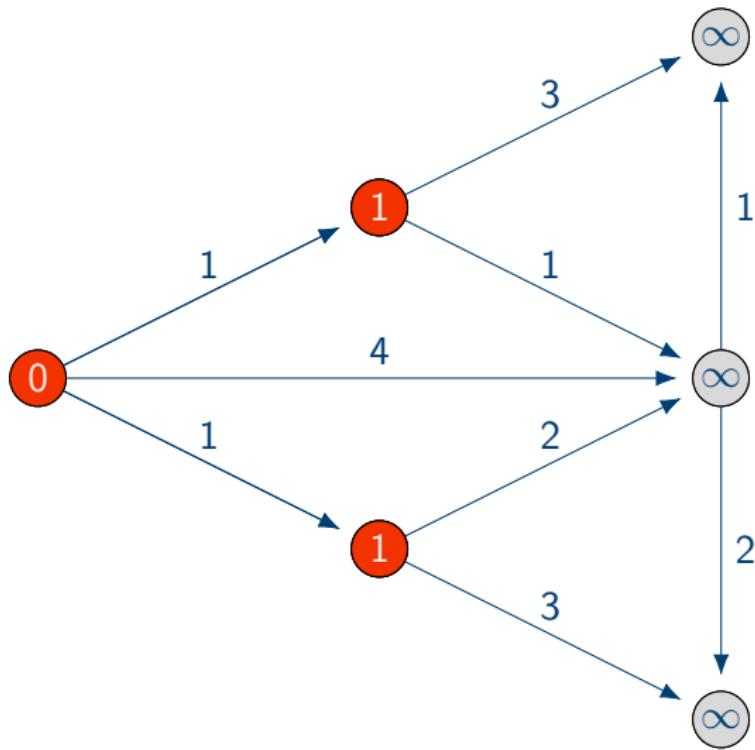
Example of Dijkstra's Algorithm



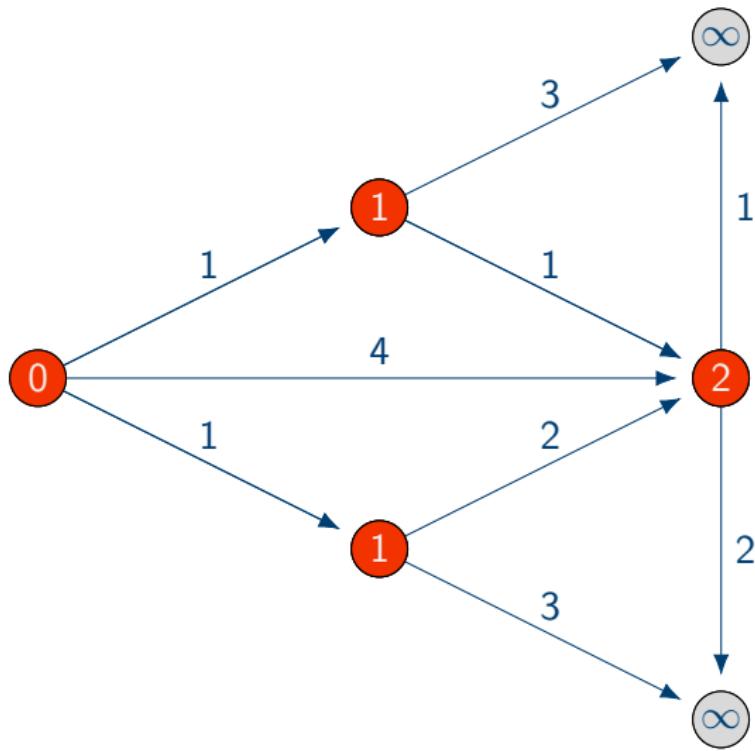
Example of Dijkstra's Algorithm



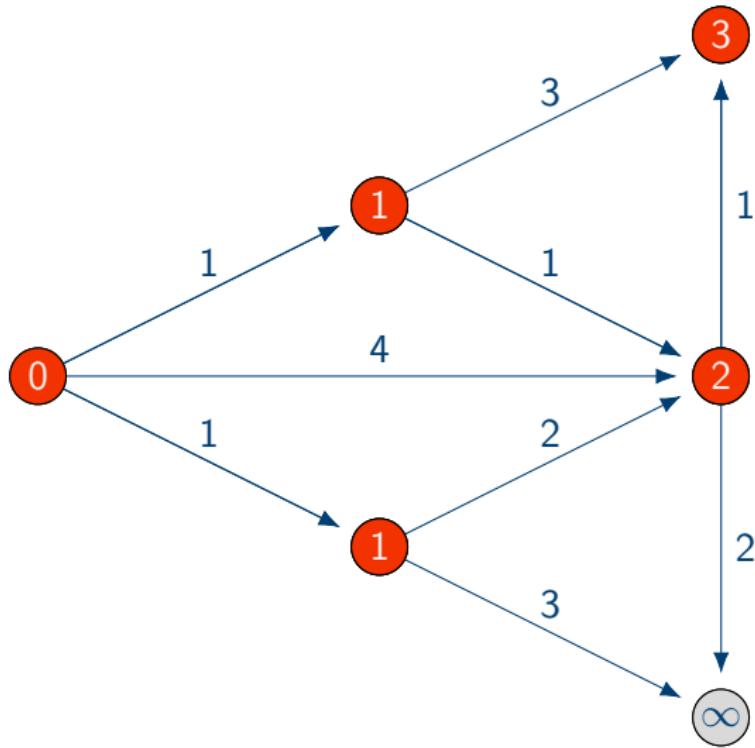
Example of Dijkstra's Algorithm



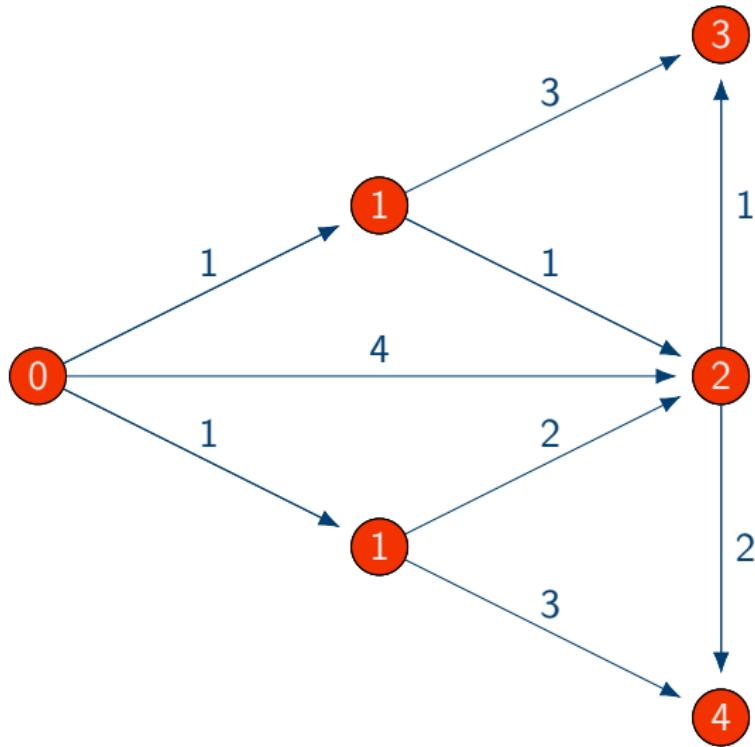
Example of Dijkstra's Algorithm



Example of Dijkstra's Algorithm



Example of Dijkstra's Algorithm

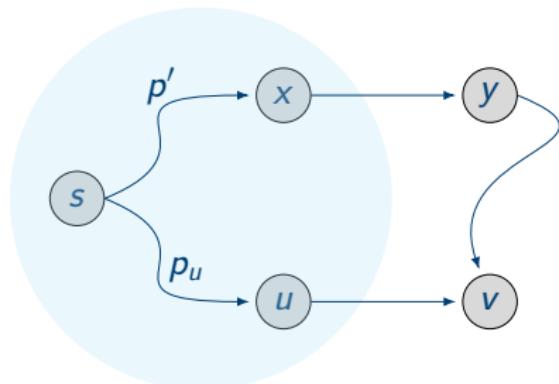


Proof of Correctness

- ▶ Let P_u be the shortest path computed for a node u .
- ▶ Claim: P_u is the shortest path from s to u .
- ▶ Prove by induction on the size of S .
 - ▶ Base case: $|S| = 1$. The only node in S is s .
 - ▶ Inductive step: we add the node v to S . Let u be the v 's predecessor on the path P_v . Could there be a shorter path P from s to v ?

Proof of Correctness

- Let P_u be the shortest path computed for a node u .
- Claim: P_u is the shortest path from s to u .
- Prove by induction on the size of S .
 - Base case: $|S| = 1$. The only node in S is s .
 - Inductive step: we add the node v to S . Let u be the v 's predecessor on the path P_v . Could there be a shorter path P from s to v ?



The alternate $s - v$ path P through x and y already too long by the time it had left the set S

Comments about Dijkstra's Algorithm

- ▶ Algorithm cannot handle negative edge lengths.
- ▶ Union of shortest paths output form a tree. Why?

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .
output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
- 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
- 3 Initially $d[s] = 0$ and $S = s$;
- 4 **while** $S \neq V$ **do**
 - 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
 - 6 Add v to S and define $d[v] = d'[v]$;
- 7 **end**

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .
output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
- 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
- 3 Initially $d[s] = 0$ and $S = s$;
- 4 **while** $S \neq V$ **do**
 - 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
 - 6 Add v to S and define $d[v] = d'[v]$;
- 7 **end**

- ▶ How many iterations are there of the while loop? .

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .
output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
- 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
- 3 Initially $d[s] = 0$ and $S = s$;
- 4 **while** $S \neq V$ **do**
 - 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
 - 6 Add v to S and define $d[v] = d'[v]$;
- 7 **end**

- ▶ How many iterations are there of the while loop? $n - 1$.

Implementing Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .
output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
- 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
- 3 Initially $d[s] = 0$ and $S = s$;
- 4 **while** $S \neq V$ **do**
 - 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
 - 6 Add v to S and define $d[v] = d'[v]$;
- 7 **end**

- ▶ How many iterations are there of the while loop? $n - 1$.
- ▶ In each iteration, for each node $v \notin S$, compute
$$\min_{e=(u,v), u \in S} d(u) + l_e.$$

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

```
1 Let  $S$  be the set of explored nodes;  
2 foreach  $u \in S$  do store distance  $d[u] = \infty$ ;  
3 Initially  $d[s] = 0$  and  $S = s$ ;  
4 while  $S \neq V$  do  
5   Select a node  $v \notin S$  with at least one edge from  $S$  for which  
       $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$  is as small as possible;  
6   Add  $v$  to  $S$  and define  $d[v] = d'[v]$ ;  
7 end
```

- ▶ Observation: If we add v to S , $d'(w)$ changes only for v 's neighbours.

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .
output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
- 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
- 3 Initially $d[s] = 0$ and $S = s$;
- 4 **while** $S \neq V$ **do**
- 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
- 6 Add v to S and define $d[v] = d'[v]$;
- 7 **end**

- ▶ Observation: If we add v to S , $d'(w)$ changes only for v 's neighbours.
- ▶ Store the minima $d'(v)$ for each node $v \in V - S$ in a priority queue .
- ▶ Determine the next node v to add to S using EXTRACTMIN.
- ▶ After adding v , for each neighbour w of v , compute $d(v) + l_{(v,w)}$.
- ▶ If $d(v) + l_{(v,w)} < d'(w)$, update w 's key using CHANGEKEY.

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .
output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
- 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
- 3 Initially $d[s] = 0$ and $S = s$;
- 4 **while** $S \neq V$ **do**
- 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
- 6 Add v to S and define $d[v] = d'[v]$;
- 7 **end**

- ▶ Observation: If we add v to S , $d'(w)$ changes only for v 's neighbours.
- ▶ Store the minima $d'(v)$ for each node $v \in V - S$ in a priority queue .
- ▶ Determine the next node v to add to S using EXTRACTMIN.
- ▶ After adding v , for each neighbour w of v , compute $d(v) + l_{(v,w)}$.
- ▶ If $d(v) + l_{(v,w)} < d'(w)$, update w 's key using CHANGEKEY.
- ▶ How many times are EXTRACTMIN and CHANGEKEY invoked?

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Dijkstra

input : A graph $G = (V, E)$, a weight map W and a source node s .
output: The distances of the vertices from s

- 1 Let S be the set of explored nodes;
- 2 **foreach** $u \in S$ **do** store distance $d[u] = \infty$;
- 3 Initially $d[s] = 0$ and $S = s$;
- 4 **while** $S \neq V$ **do**
- 5 Select a node $v \notin S$ with at least one edge from S for which
 $d'(v) = \min_{e=(u,v):u \in S} d[u] + W(e)$ is as small as possible;
- 6 Add v to S and define $d[v] = d'[v]$;
- 7 **end**

- ▶ Observation: If we add v to S , $d'(w)$ changes only for v 's neighbours.
- ▶ Store the minima $d'(v)$ for each node $v \in V - S$ in a priority queue .
- ▶ Determine the next node v to add to S using EXTRACTMIN.
- ▶ After adding v , for each neighbour w of v , compute $d(v) + l_{(v,w)}$.
- ▶ If $d(v) + l_{(v,w)} < d'(w)$, update w 's key using CHANGEKEY.
- ▶ How many times are EXTRACTMIN and CHANGEKEY invoked? $n - 1$ and m times, respectively.

Single Source Shortest Path Problem

- ▶ $G = (V, E)$ is a connected directed graph. Each edge e has a length l_e . Note that the weights may be negative.
- ▶ V has n nodes and E has m edges.
- ▶ Length of a path P is the sum of lengths of the edges in P .
- ▶ Goal is to determine the shortest path from some start node s to all other nodes in V .
- ▶ Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

Single Source Shortest Path Problem

- $G = (V, E)$ is a connected directed graph. Each edge e has a length l_e . Note that the weights may be negative.
- V has n nodes and E has m edges.
- Length of a path P is the sum of lengths of the edges in P .
- Goal is to determine the shortest path from some start node s to all other nodes in V .
- Aside: If G is undirected, convert to a directed graph by replacing each edge in G by two directed edges.

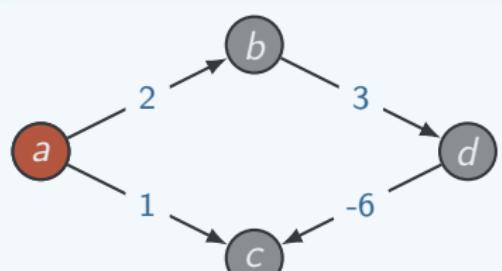
SHORTEST PATHS

INSTANCE A directed graph $G(V, E)$, a function $l : E \rightarrow \mathbb{R}$, and a node $s \in V$

SOLUTION A set $\{P_u, u \in V\}$, where P_u is the shortest path in G from s to u .

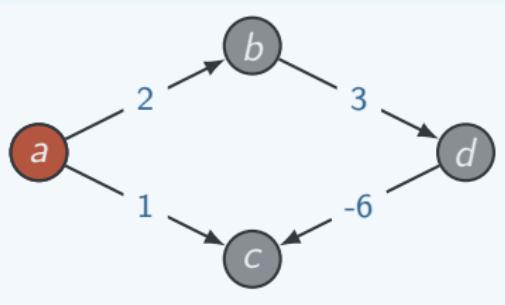
Bellman-Ford Algorithm

Dijkstra – Can fail if negative edge costs.

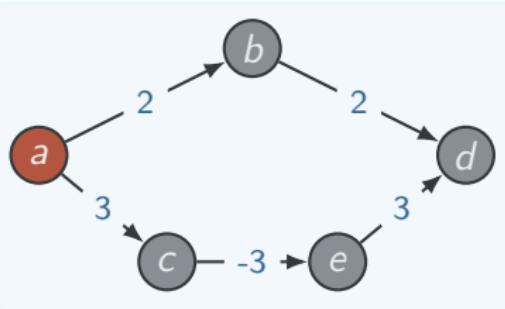


Bellman-Ford Algorithm

Dijkstra – Can fail if negative edge costs.

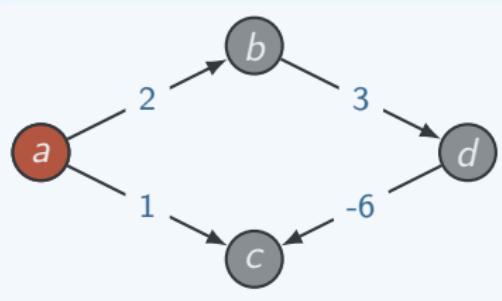


Re-weighting – Adding a constant to every edge weight can fail

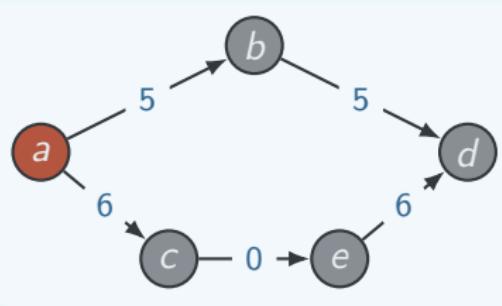


Bellman-Ford Algorithm

Dijkstra – Can fail if negative edge costs.

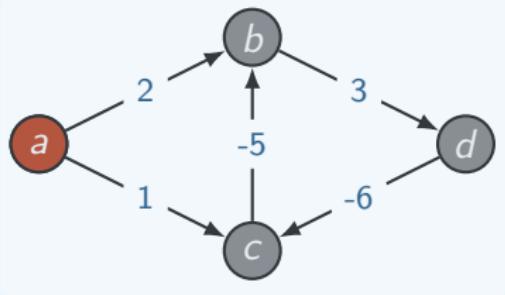


Re-weighting – Adding a constant to every edge weight can fail



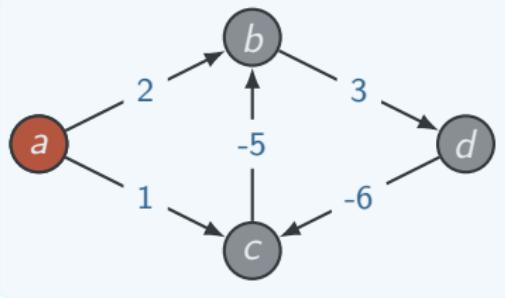
Bellman-Ford Algorithm

If some path from s to t contains a negative cost cycle,
there does not exist a shortest $s-t$ path; otherwise, there exists
one that is simple.



Bellman-Ford Algorithm

If some path from s to t contains a negative cost cycle,
there does not exist a shortest $s-t$ path; otherwise, there exists
one that is simple.



The Bellman-Ford algorithm is a way to find single source shortest paths in a graph with negative edge weights (but no negative cycles).

Bellman-Ford Algorithm

$\text{OPT}(i, v) = \text{length of shortest } v-t \text{ path } P \text{ using at most } i \text{ edges.}$

Bellman-Ford Algorithm

$OPT(i, v) = \text{length of shortest } v-t \text{ path } P \text{ using at most } i \text{ edges.}$

- ▶ Case 1 : P uses at most $i - 1$ edges.

$$OPT(i, v) = OPT(i - 1, v)$$

Bellman-Ford Algorithm

$\text{OPT}(i, v) = \text{length of shortest } v-t \text{ path } P \text{ using at most } i \text{ edges.}$

- ▶ Case 1 : P uses at most $i - 1$ edges.

$$\text{OPT}(i, v) = \text{OPT}(i - 1, v)$$

- ▶ Case 2 : P uses exactly i edges
 - ▶ if (v, w) is first edge, then OPT uses (v, w) , and then selects best $w-t$ path using at most $i - 1$ edges

Bellman-Ford Algorithm

$OPT(i, v) = \text{length of shortest } v-t \text{ path } P \text{ using at most } i \text{ edges.}$

- ▶ Case 1 : P uses at most $i - 1$ edges.

$$OPT(i, v) = OPT(i - 1, v)$$

- ▶ Case 2 : P uses exactly i edges

- ▶ if (v, w) is first edge, then OPT uses (v, w) , and then selects best $w-t$ path using at most $i - 1$ edges

$$OPT(i, v) = \begin{cases} 0, & \text{if } i = 0 \\ \min \left\{ \begin{array}{l} OPT(i - 1, v) \\ \min \{ OPT(i - 1, w) + c_{vw} \} \end{array} \right\}, & \text{otherwise} \end{cases}$$

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

```
1 foreach  $v \in V$  do  $d[0, v] = \infty$ ;  
2 Initially  $d[0, s] = 0$ ;  
3 for  $i = 1$  to  $n - 1$  do  
4   foreach  $v \in V$  do  
5     |  $d[i, v] = d[i - 1, v]$   
6   end  
7   foreach edge  $(w, v) \in E$  do  
8     |  $d[i, v] = \min\{d[i, v], d[i - 1, w] + c_{wv}\}$   
9   end  
10 end
```

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

```
1 foreach  $v \in V$  do  $d[0, v] = \infty$ ;  
2 Initially  $d[0, s] = 0$ ;  
3 for  $i = 1$  to  $n - 1$  do  
4   foreach  $v \in V$  do  
5     |  $d[i, v] = d[i - 1, v]$   
6   end  
7   foreach edge  $(w, v) \in E$  do  
8     |  $d[i, v] = \min\{d[i, v], d[i - 1, w] + c_{wv}\}$   
9   end  
10 end
```

- ▶ Computational cost: $O(mn)$

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s .

output: The distances of the vertices from s

```
1 foreach  $v \in V$  do  $d[0, v] = \infty$ ;  
2 Initially  $d[0, s] = 0$ ;  
3 for  $i = 1$  to  $n - 1$  do  
4   foreach  $v \in V$  do  
5     |  $d[i, v] = d[i - 1, v]$   
6   end  
7   foreach edge  $(w, v) \in E$  do  
8     |  $d[i, v] = \min\{d[i, v], d[i - 1, w] + c_{wv}\}$   
9   end  
10 end
```

- ▶ Computational cost: $O(mn)$
- ▶ For finding the shortest paths, it is necessary to maintain a **successor** for each table entry.

A Faster implementation of Dijkstra's Algorithm

Algorithm: Shortest path algorithm – Bellman-Ford

input : A graph $G = (V, E)$, a weight map W and a source node s .

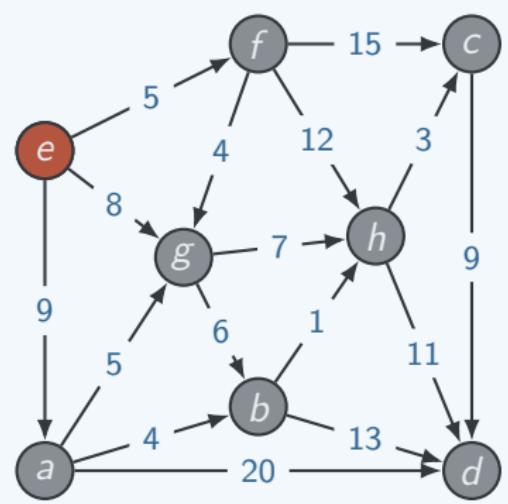
output: The distances of the vertices from s

```
1 foreach  $v \in V$  do  $d[0, v] = \infty$ ;  
2 Initially  $d[0, s] = 0$ ;  
3 for  $i = 1$  to  $n - 1$  do  
4   foreach  $v \in V$  do  
5     |  $d[i, v] = d[i - 1, v]$   
6   end  
7   foreach edge  $(w, v) \in E$  do  
8     |  $d[i, v] = \min\{d[i, v], d[i - 1, w] + c_{wv}\}$   
9   end  
10 end
```

- ▶ Computational cost: $O(mn)$
- ▶ For finding the shortest paths, it is necessary to maintain a successor for each table entry.

How to detect negative cycles?

Shortest path – an example



Compute the shortest path from e to all other nodes!

Questions?

Shortest path
– Bellman-Ford –

Teoria dos Grafos e Computabilidade

— Trees and spanning trees —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas



Teoria dos Grafos e Computabilidade

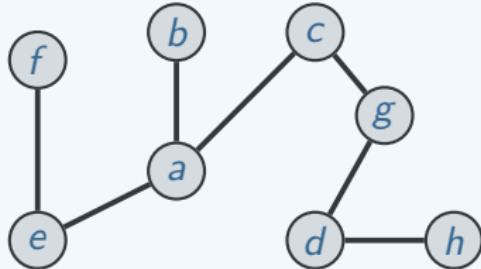
— Trees —

Silvio Jamil F. Guimarães

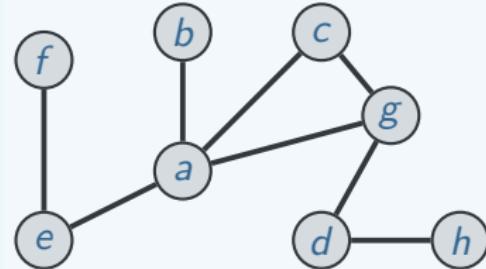
Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Trees

- ▶ A tree is an undirected connected graph with no cycles.
- ▶ Genealogical trees, evolutionary trees, decision trees, various data structures in Computer Science



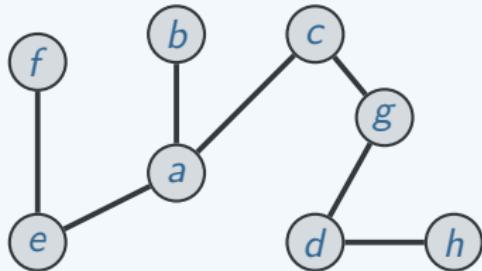
Tree



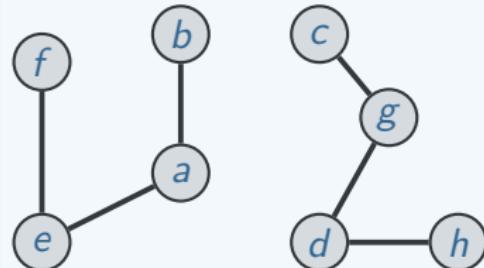
Not a tree (has cycle)

Trees

- ▶ A tree is an undirected connected graph with no cycles.
- ▶ Genealogical trees, evolutionary trees, decision trees, various data structures in Computer Science



Tree



Not a tree (not connected) –
this is a forest

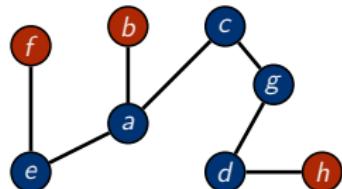
Theorem A tree has exactly one path between any pair of vertices

Theorem A tree has exactly one path between any pair of vertices

- ▶ A vertex of degree 1 is called a **leaf**.
- ▶ Sometimes, vertices of degree 0 are also counted as leaves
- ▶ A vertex with degree greater than 1 is an **internal** vertex.

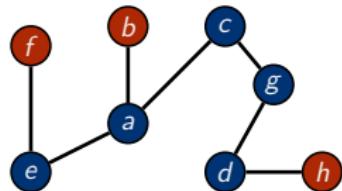
Theorem A tree has exactly one path between any pair of vertices

- ▶ A vertex of degree 1 is called a leaf.
- ▶ Sometimes, vertices of degree 0 are also counted as leaves
- ▶ A vertex with degree greater than 1 is an internal vertex.



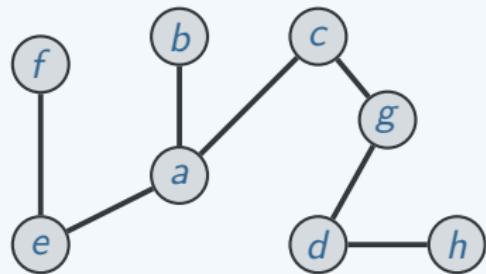
Theorem Every tree, with at least two vertices, has at least two leaves.

- ▶ A vertex of degree 1 is called a **leaf**.
- ▶ Sometimes, vertices of degree 0 are also counted as leaves
- ▶ A vertex with degree greater than 1 is an **internal** vertex.



Trees

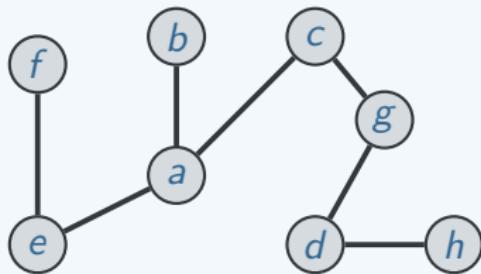
Theorem All trees on $n \geq 1$ vertices have exactly $n - 1$ edges



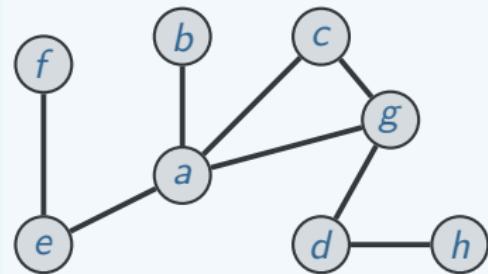
Tree

Trees

Theorem All trees on $n \geq 1$ vertices have exactly $n - 1$ edges



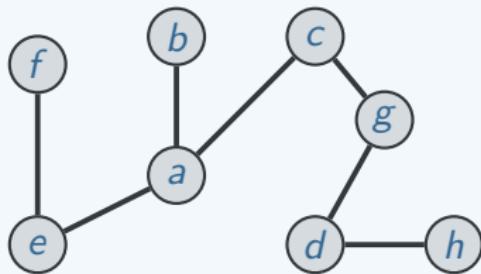
Tree



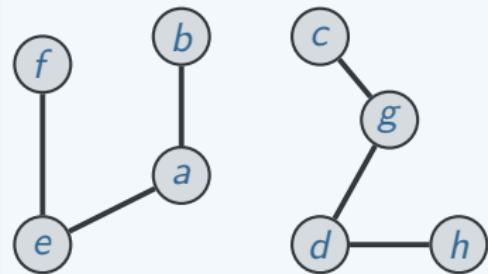
Not a tree (has cycle)

Trees

Theorem All trees on $n \geq 1$ vertices have exactly $n - 1$ edges



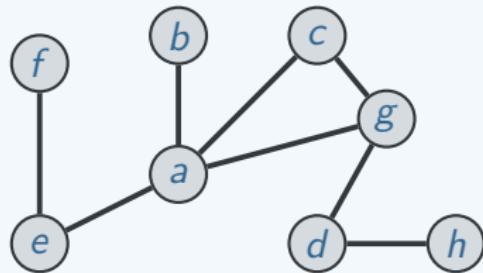
Tree



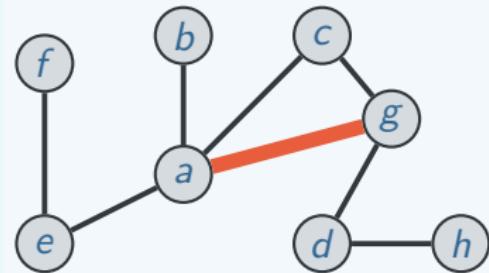
Not a tree (not connected) – this is a forest

Trees

Lemma Removing an edge from a cycle keeps connectivity



Not a tree (has cycle)



Still connected after removal

Spanning trees

A spanning tree of an undirected graph is a subgraph that is a tree and includes all vertices.

Spanning trees

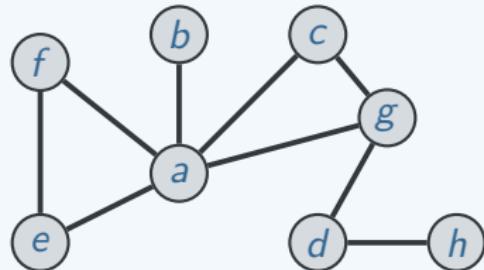
A spanning tree of an undirected graph is a subgraph that is a tree and includes all vertices.

A graph G has a spanning tree iff it is connected:

- ▶ If G has a spanning tree, it is connected: any two vertices have a path between them in the spanning tree and hence in G .
- ▶ If G is connected, we will construct a spanning tree

Spanning trees

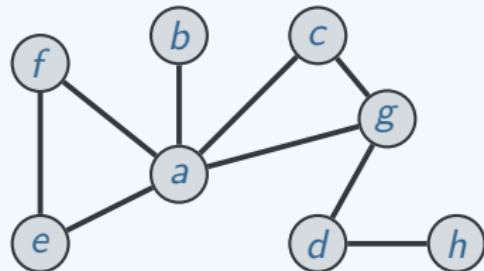
1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



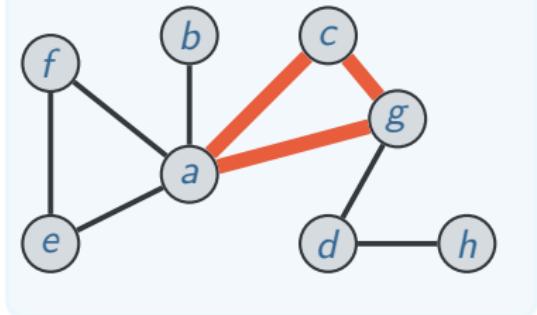
There exists a cycle?

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



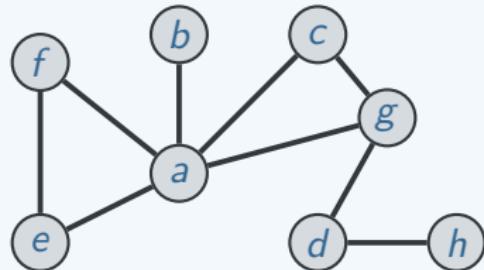
There exists a cycle?



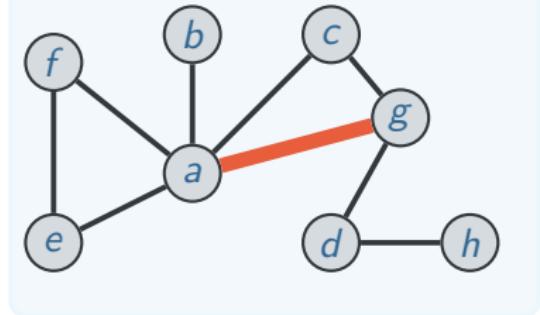
Cycle: a-c-g-a

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



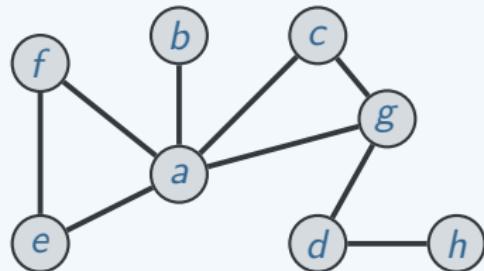
There exists a cycle?



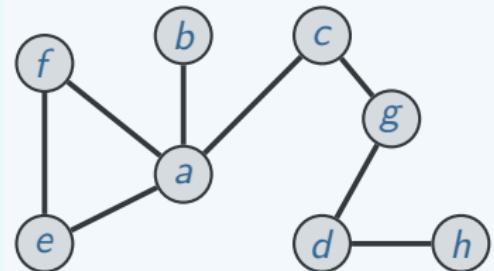
Remove the edge a-g

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are **any cycles**, **pick one** and **remove** any edge.
3. Repeat the item 2 until we arrive at a subgraph T with **no cycles**.



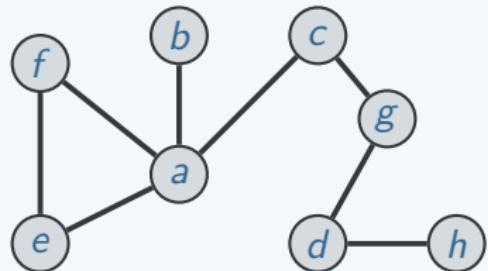
There exists a cycle?



Remove the edge a-g

Spanning trees

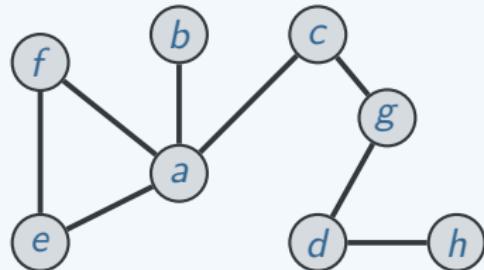
1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



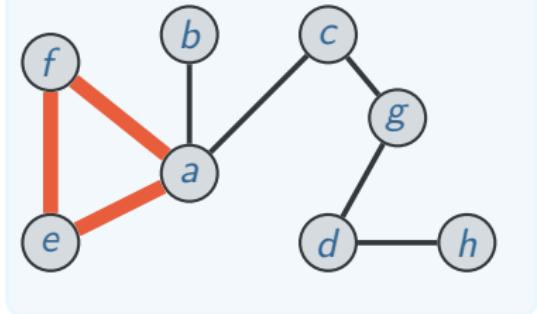
There exists a cycle?

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



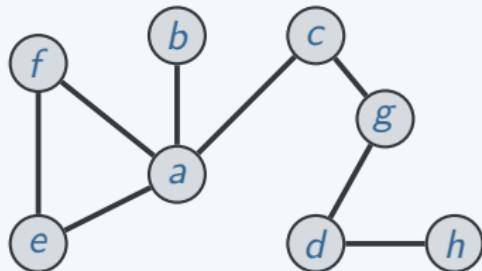
There exists a cycle?



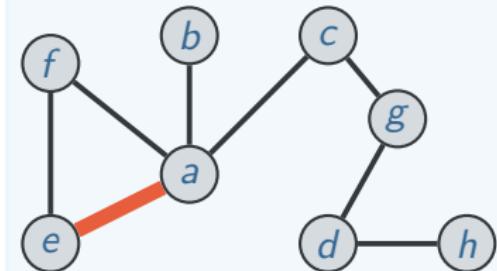
Cycle: a-f-e-a

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are **any cycles**, **pick one** and **remove** any edge.
3. Repeat the item 2 until we arrive at a subgraph T with **no cycles**.



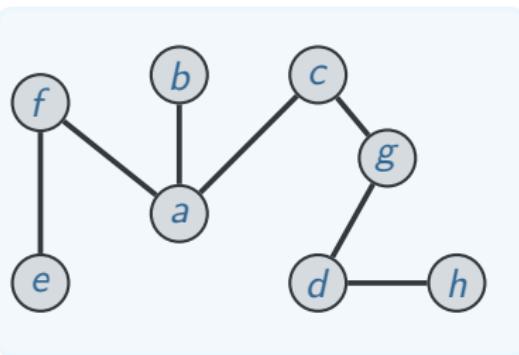
There exists a cycle?



Remove the edge a-e

Spanning trees

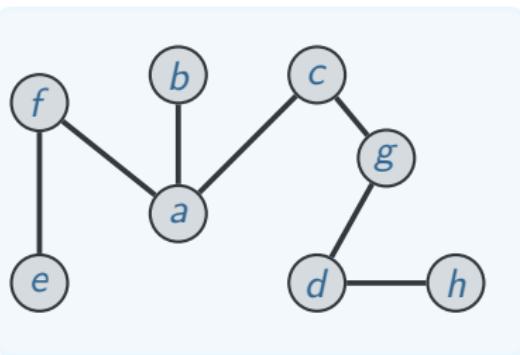
1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.
3. Repeat the item 2 until we arrive at a subgraph T with no cycles.



T is still connected, and has no cycles, so it's a tree!

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.
3. Repeat the item 2 until we arrive at a subgraph T with no cycles.



T is still connected, and has no cycles, so it's a tree!
It reaches all vertices, so it is a spanning tree

Spanning trees

Converse theorem

If a connected graph on n vertices has $n - 1$ edges, it is a **tree**

Spanning trees

Converse theorem

If a connected graph on n vertices has $n - 1$ edges, it is a tree

A forest is an undirected graph with no cycles and each connected component is a tree.

Spanning trees

Converse theorem

If a connected graph on n vertices has $n - 1$ edges, it is a tree

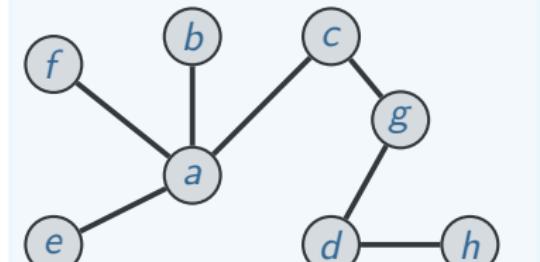
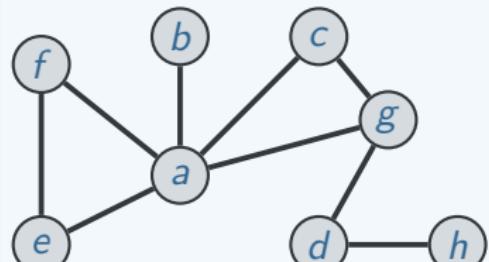
A forest is an undirected graph with no cycles and each connected component is a tree.

Theorem

A forest with n vertices and k trees has $n - k$ edges.

Spanning trees

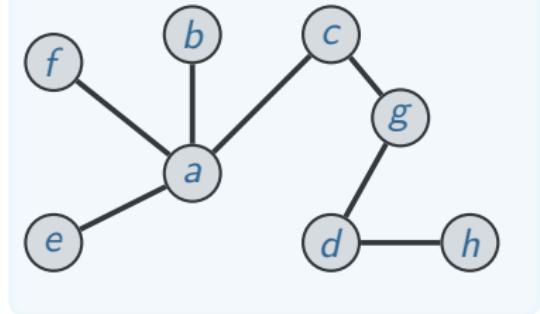
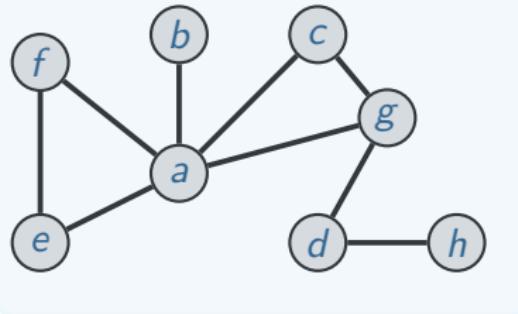
Let G be a connected graph on n vertices and T be a spanning tree computed from G



Spanning trees

Let G be a connected graph on n vertices and T be a spanning tree computed from G

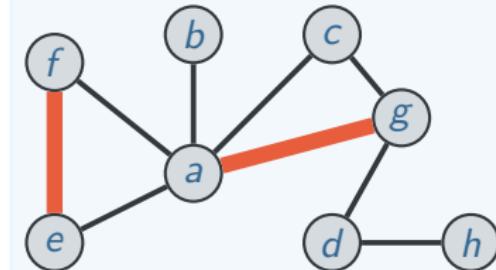
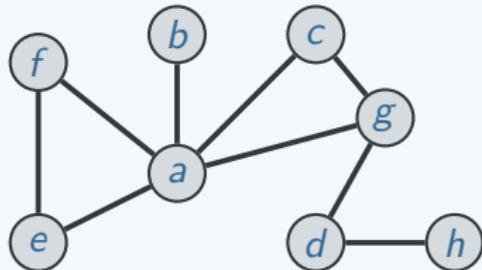
1. A **branch** is an edge in a spanning tree T .



Spanning trees

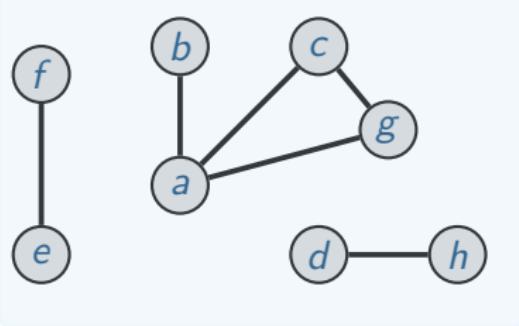
Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A **branch** is an edge in a spanning tree T .
2. A **chord** is an edge of the connected graph G that is not a branch of a spanning tree T .



Spanning forest

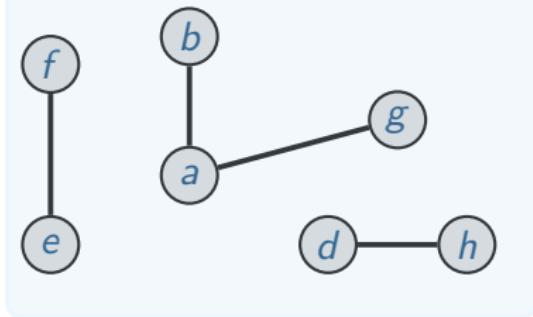
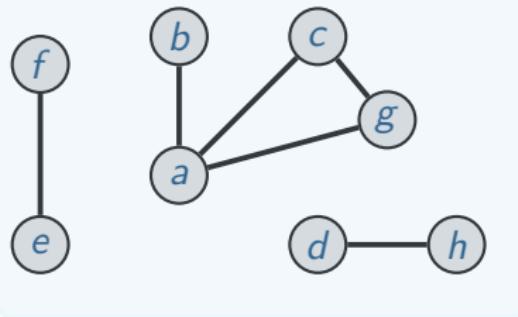
Let G be a graph on n vertices.



Spanning forest

Let G be a graph on n vertices.

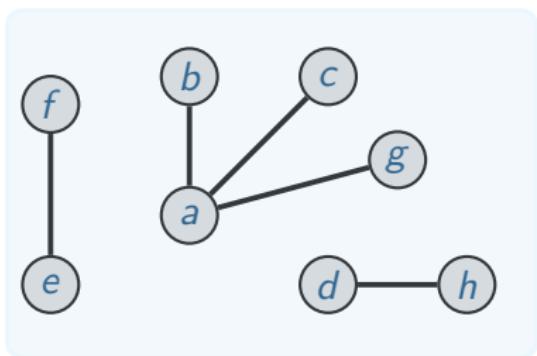
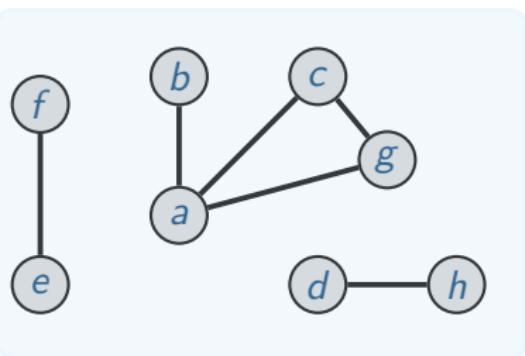
1. A **forest** is a collection of trees in the graph



Spanning forest

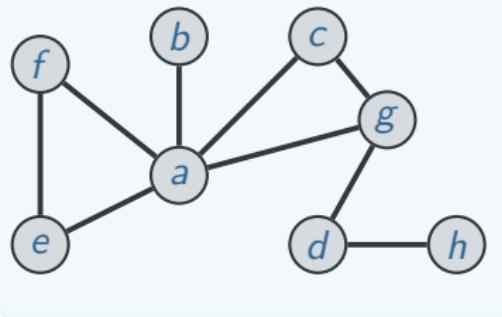
Let G be a graph on n vertices.

1. A **forest** is a collection of trees in the graph
2. A **spanning forest** is a collection of spanning trees.



Spanning trees

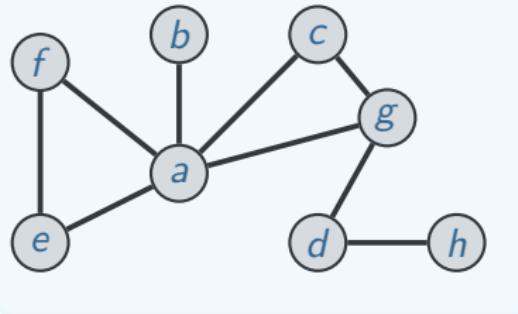
Let G be a connected graph on n vertices and T be a spanning tree computed from G



Spanning trees

Let G be a connected graph on n vertices and T be a spanning tree computed from G

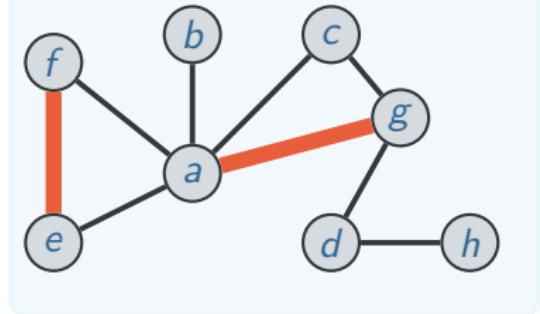
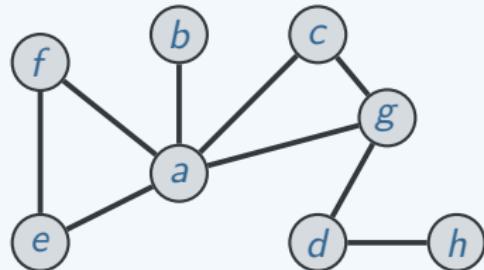
1. A **rank** of G is the number of branches of the spanning trees (or spanning forest) which is given by $r = n - k$



Spanning trees

Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A **rank** of G is the number of branches of the spanning trees (or spanning forest) which is given by $r = n - k$
2. A **nullity** is the number of chords related to the spanning trees (or spanning forest) which is given by $\mu = e - n - k$



Spanning trees

Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A **rank** of G is the number of branches of the spanning trees (or spanning forest) which is given by $r = n - k$
2. A **nullity** is the number of chords related to the spanning trees (or spanning forest) which is given by $\mu = e - n - k$

Remember that k is the number of connected component. For a spanning tree, $k = 1$, but for a spanning forest, k is the number of spanning trees which are in the spanning forest.

Questions?

Trees and spanning trees
– Trees –



Teoria dos Grafos e Computabilidade

— Minimum Spanning Trees —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Minimum Spanning Tree (MST)

- ▶ Given an undirected graph $G = (V, E)$ with a cost $c_e > 0$ associated with each edge $e \in E$.
- ▶ Find a subset T of edges such that the graph (V, T) is connected and the cost $\sum_{e \in T} c_e$ is as small as possible.

Minimum Spanning Tree (MST)

- ▶ Given an undirected graph $G = (V, E)$ with a cost $c_e > 0$ associated with each edge $e \in E$.
- ▶ Find a subset T of edges such that the graph (V, T) is connected and the cost $\sum_{e \in T} c_e$ is as small as possible.

MINIMUM SPANNING TREE

INSTANCE An undirected graph $G = (V, E)$ and a function $c : E \rightarrow \mathbb{R}^+$

SOLUTION A set $T \subseteq E$ of edges such that (V, T) is connected and the $\sum_{e \in T} c_e$ is as small as possible.

Minimum Spanning Tree (MST)

- Given an undirected graph $G = (V, E)$ with a cost $c_e > 0$ associated with each edge $e \in E$.
- Find a subset T of edges such that the graph (V, T) is connected and the cost $\sum_{e \in T} c_e$ is as small as possible.

MINIMUM SPANNING TREE

INSTANCE An undirected graph $G = (V, E)$ and a function $c : E \rightarrow \mathbb{R}^+$

SOLUTION A set $T \subseteq E$ of edges such that (V, T) is connected and the $\sum_{e \in T} c_e$ is as small as possible.

- Claim: If T is a minimum-cost solution to this network design problem then (V, T) is a tree.
- A subset T of E is a spanning tree of G if (V, T) is a tree.

Greedy Algorithm for the MST Problem

- ▶ Template: process edges in some order. Add an edge to T if tree property is not violated.

Greedy Algorithm for the MST Problem

- ▶ Template: process edges in some order. Add an edge to T if tree property is not violated.

Increasing cost order *Process edges in increasing order of cost.*

Discard an edge if it creates a cycle.

Dijkstra-like *Start from a node s and grow T outward from s : add the node that can be attached most cheaply to current tree.*

Decreasing cost order *Delete edges in order of decreasing cost as long as graph remains connected.*

Greedy Algorithm for the MST Problem

- ▶ Template: process edges in some order. Add an edge to T if tree property is not violated.

Increasing cost order *Process edges in increasing order of cost.
Discard an edge if it creates a cycle.*

Dijkstra-like *Start from a node s and grow T outward from s :
add the node that can be attached most cheaply to current
tree.*

Decreasing cost order *Delete edges in order of decreasing cost as
long as graph remains connected.*

- ▶ Which of these algorithms works?

Greedy Algorithm for the MST Problem

- ▶ Template: process edges in some order. Add an edge to T if tree property is not violated.

Increasing cost order *Process edges in increasing order of cost.*

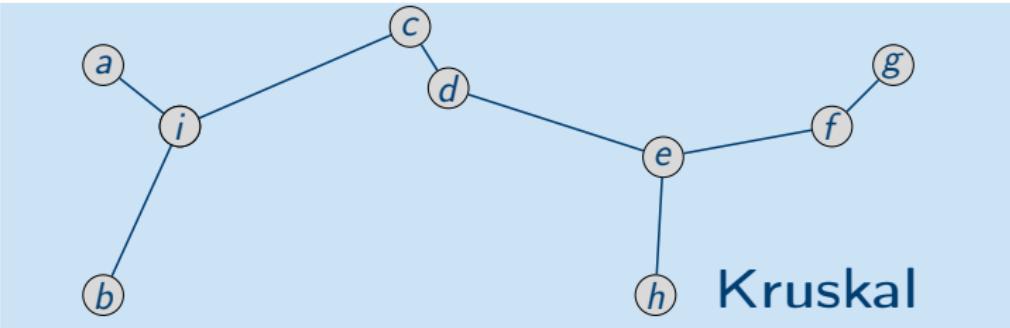
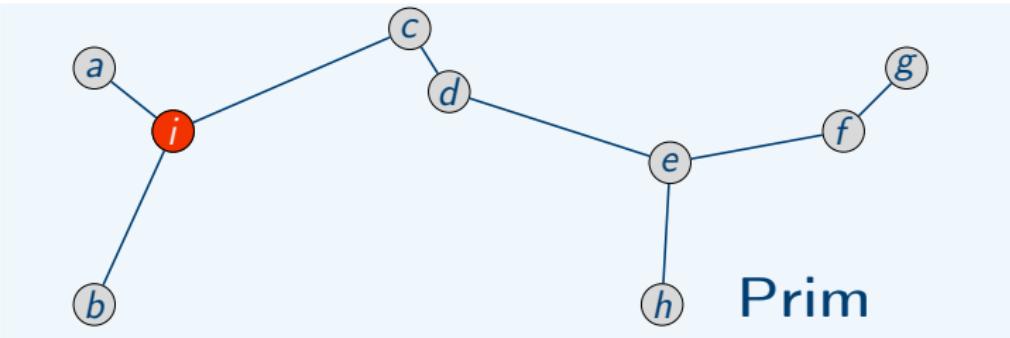
Discard an edge if it creates a cycle. **Kruskal's algorithm**

Dijkstra-like *Start from a node s and grow T outward from s : add the node that can be attached most cheaply to current tree.* **Prim's algorithm**

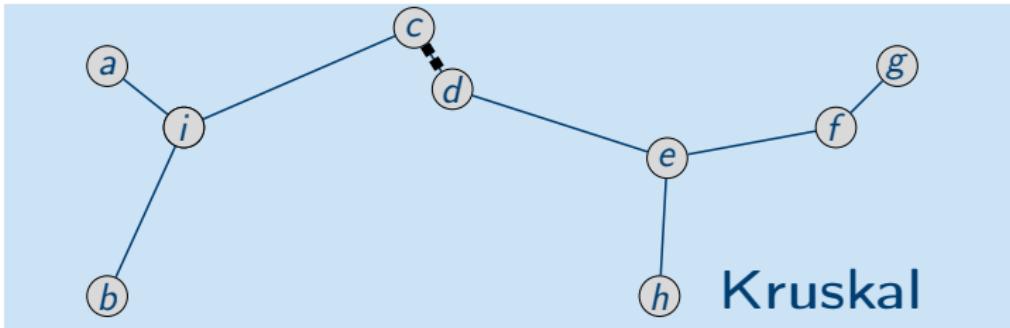
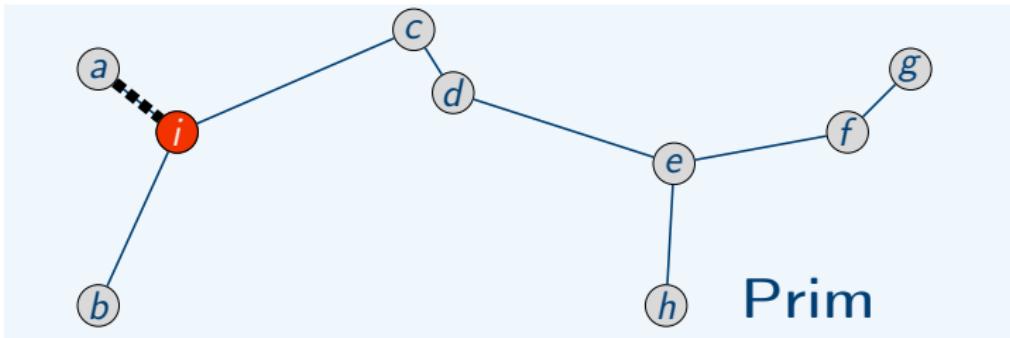
Decreasing cost order *Delete edges in order of decreasing cost as long as graph remains connected.* **Reverse-Delete algorithm**

- ▶ Which of these algorithms works? All of them!

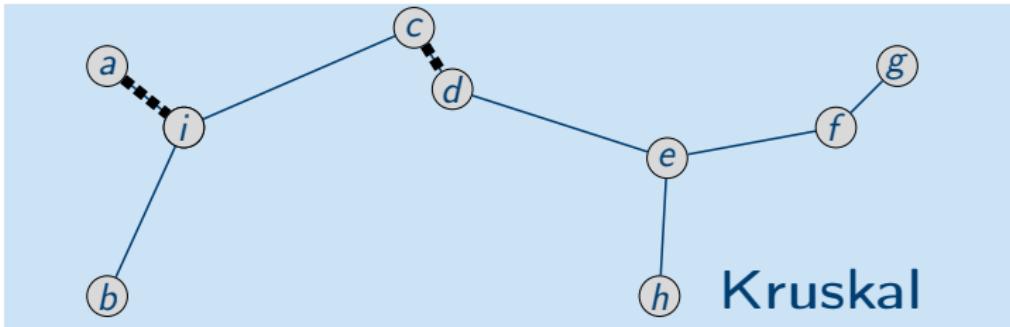
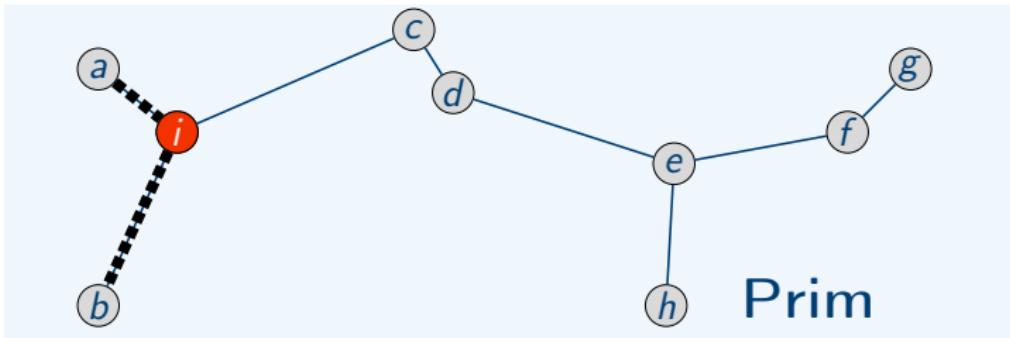
Example of Prim's and Kruskal's Algorithms



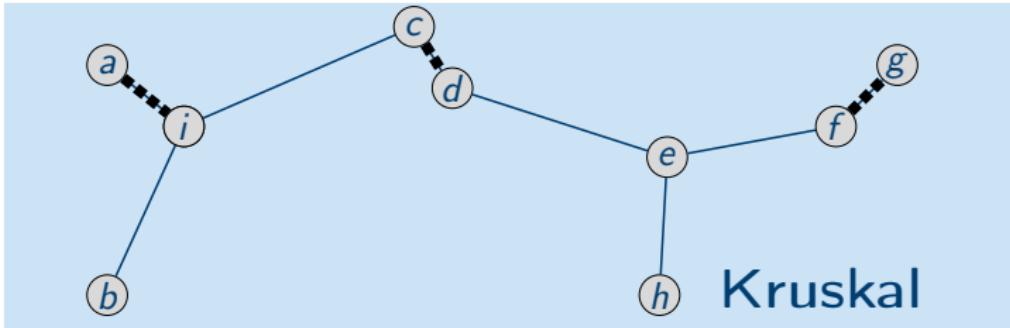
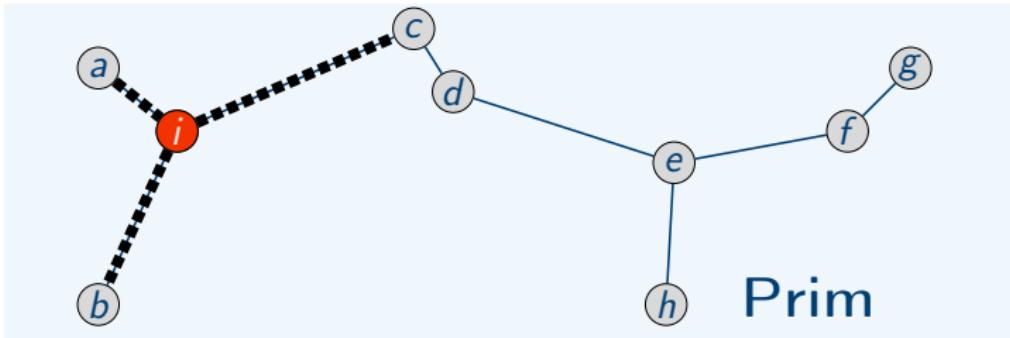
Example of Prim's and Kruskal's Algorithms



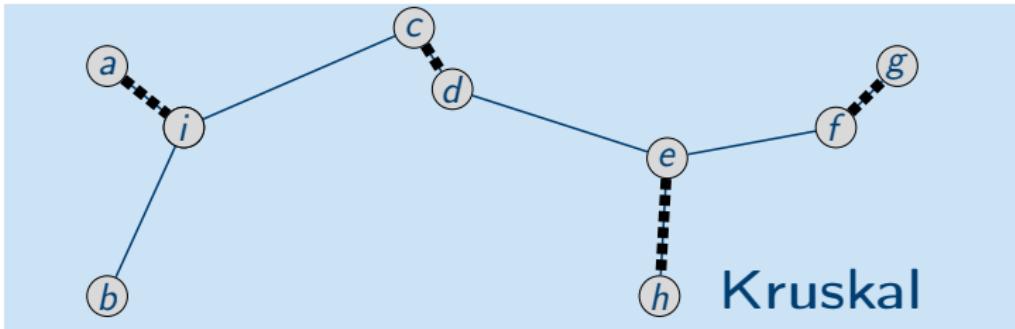
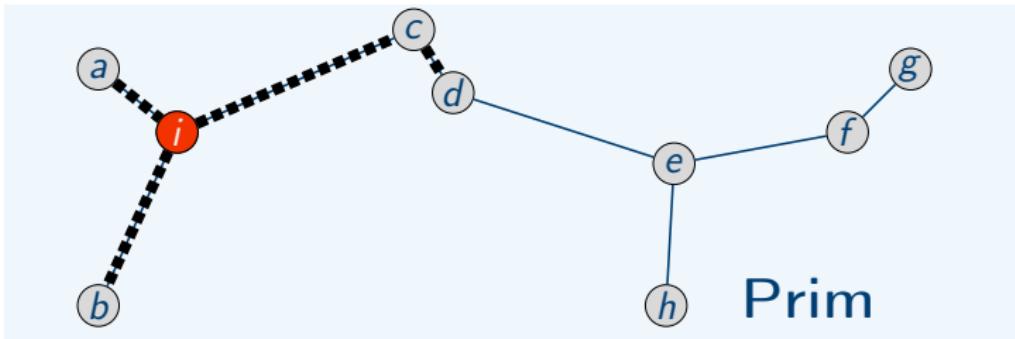
Example of Prim's and Kruskal's Algorithms



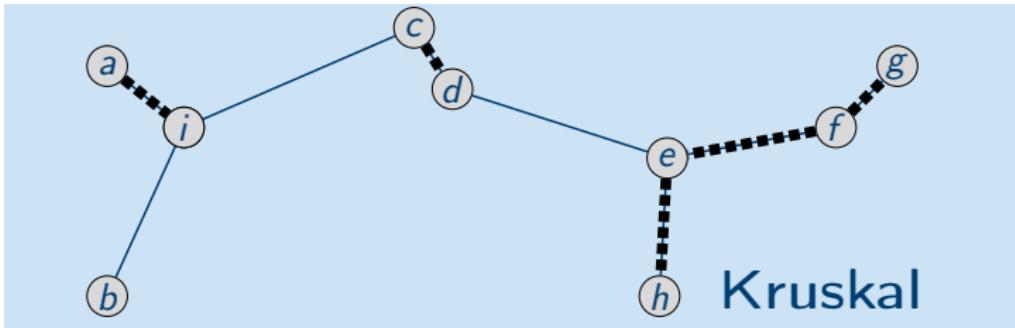
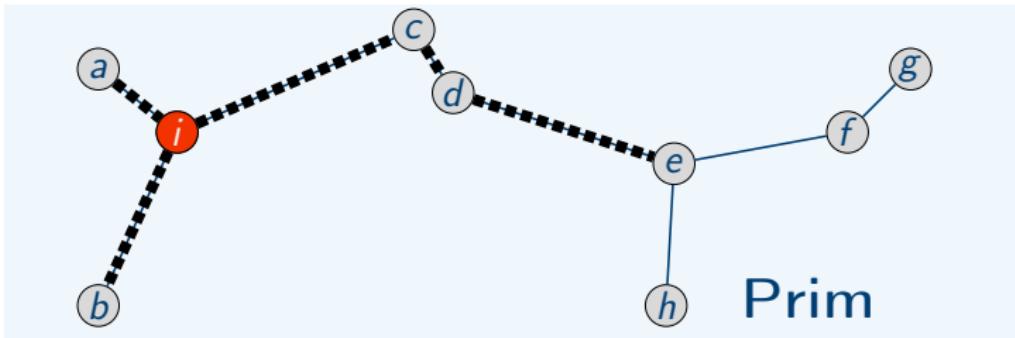
Example of Prim's and Kruskal's Algorithms



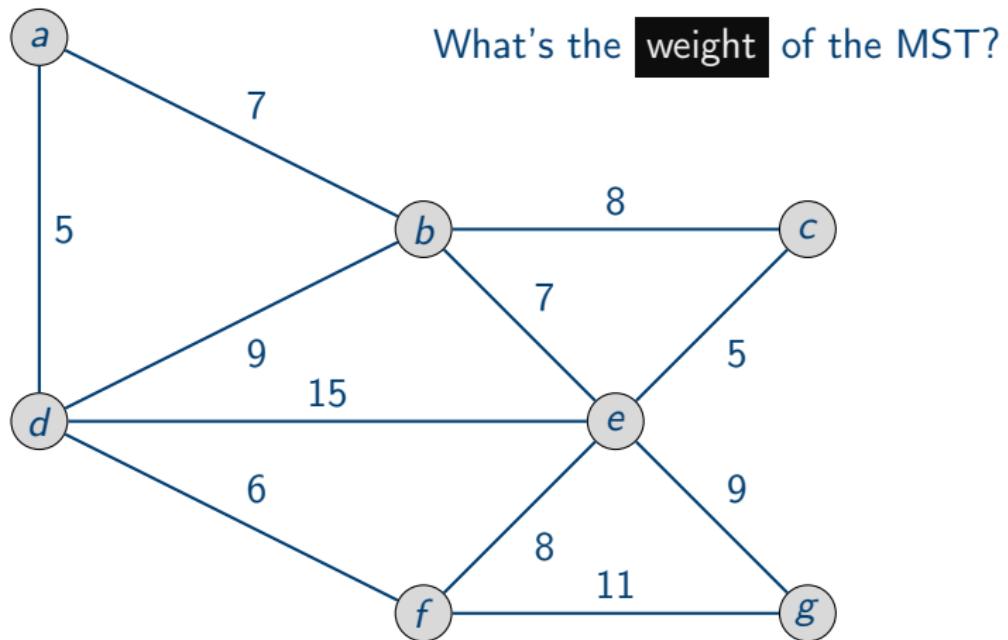
Example of Prim's and Kruskal's Algorithms



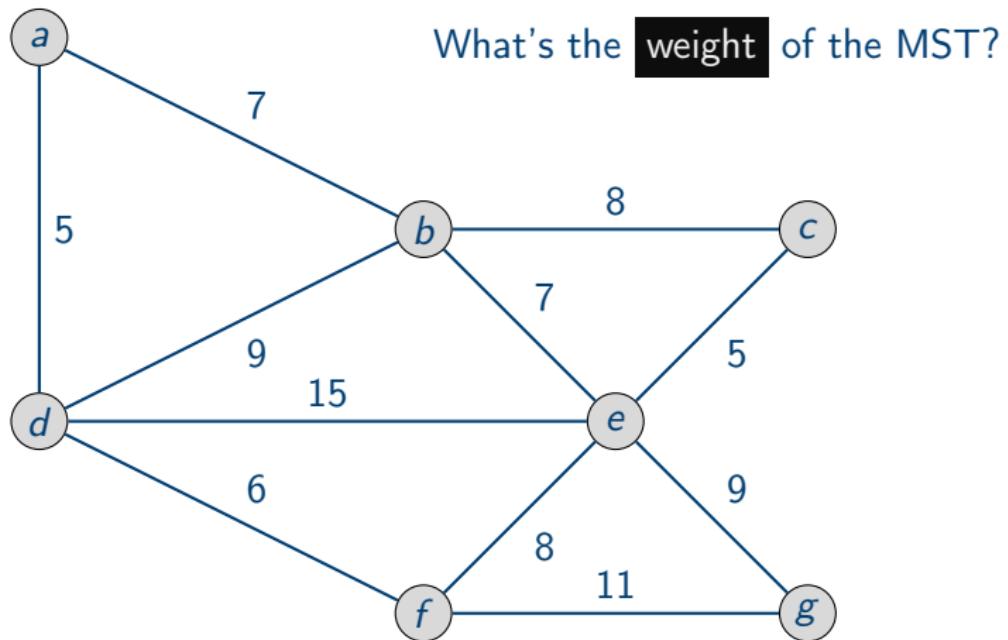
Example of Prim's and Kruskal's Algorithms



Example of Prim's Algorithm



Example of Kruskal's Algorithm

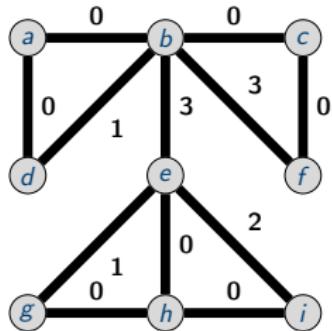


Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).

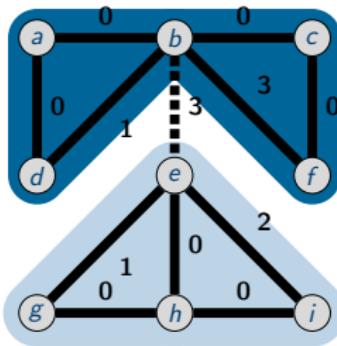
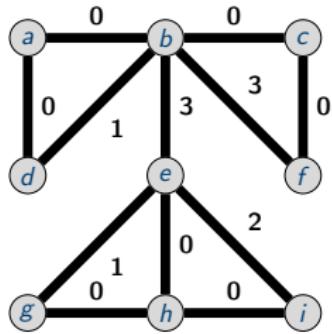
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).



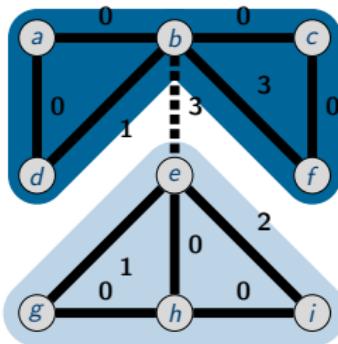
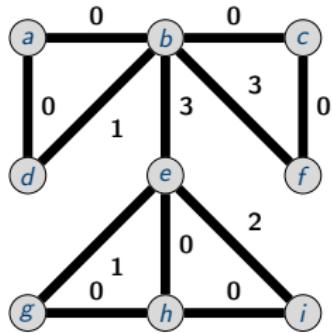
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).



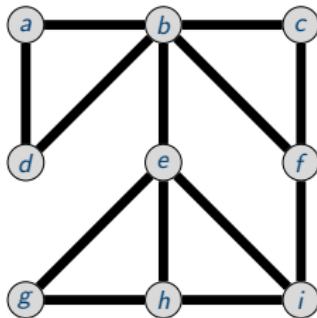
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



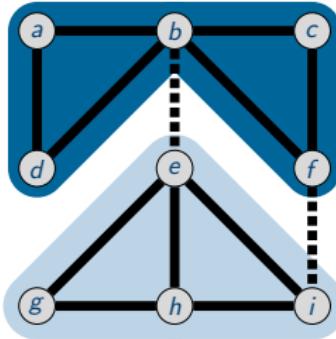
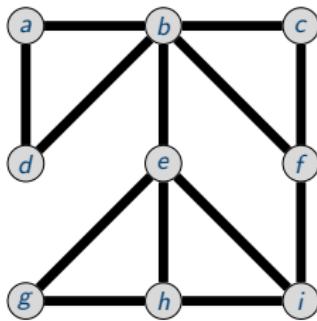
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



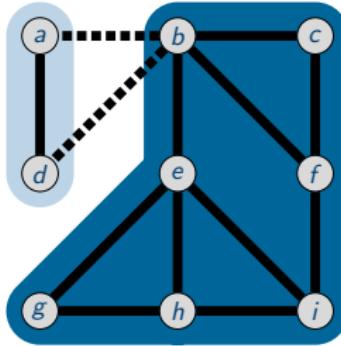
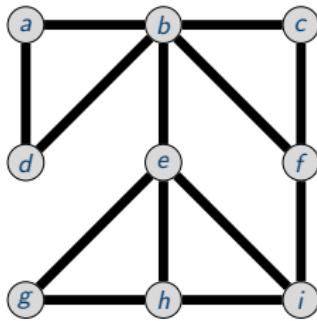
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



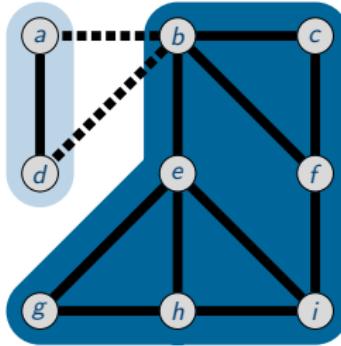
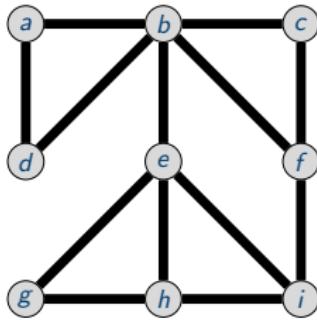
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.
- ▶ $\text{cut}(S)$ is a cut because deleting the edges in $\text{cut}(S)$ disconnects S from $V - S$.

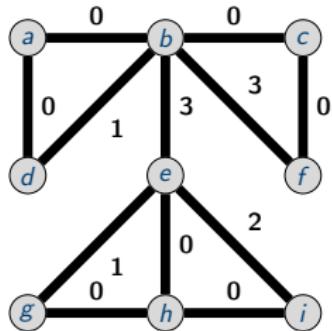


Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).

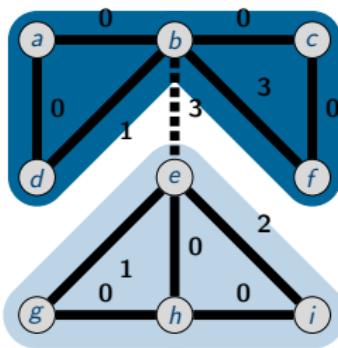
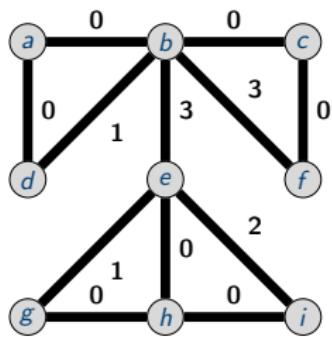
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).



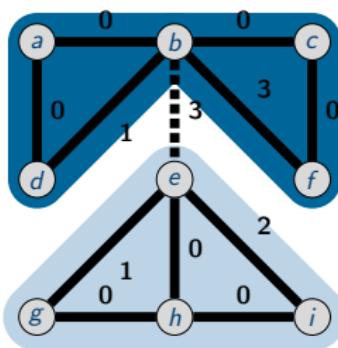
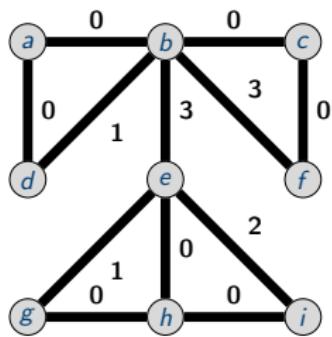
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).



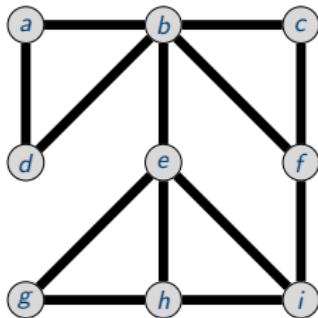
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



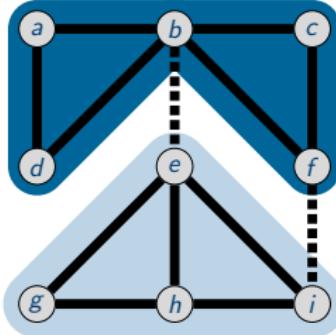
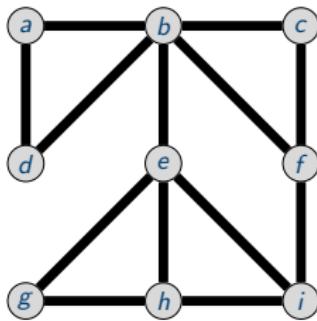
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



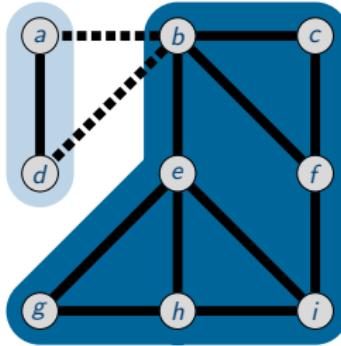
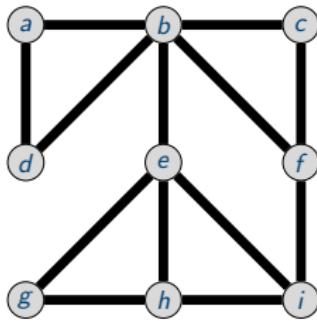
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



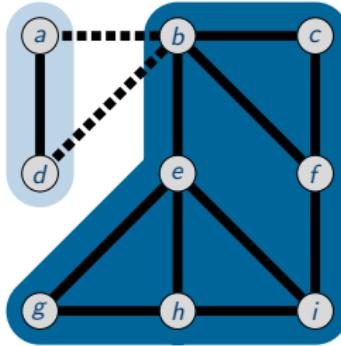
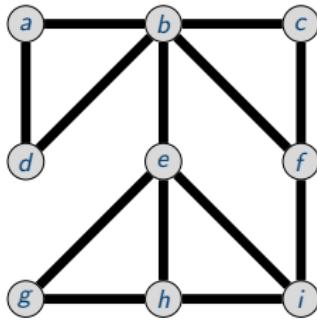
Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



Graph Cuts

- ▶ A **cut** in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- ▶ Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.
- ▶ $\text{cut}(S)$ is a cut because deleting the edges in $\text{cut}(S)$ **disconnects** S from $V - S$.



Cut Property

- When is it safe to include an edge in an MST?

Cut Property

- ▶ When is it safe to include an edge in an MST?
- ▶ Assume all edge costs are distinct.
- ▶ Let $S \subset V$, S is not empty or equal to V .
- ▶ Let e be the cheapest edge in $\text{cut}(S)$.
- ▶ Claim: every MST contains e .

Cut Property

- ▶ When is it safe to include an edge in an MST?
- ▶ Assume all edge costs are distinct.
- ▶ Let $S \subset V$, S is not empty or equal to V .
- ▶ Let e be the cheapest edge in $\text{cut}(S)$.
- ▶ Claim: every MST contains e .
- ▶ Proof: exchange argument. If a supposed MST T does not contain e , show that there is a tree with smaller cost than T that contains e .

Using the Cut Property

- ▶ Let F be the set of all edges that satisfy the cut property.
- ▶ Is the graph induced by F connected?
- ▶ Can the graph induced by F contain a cycle?
- ▶ How many edges can F contain?

Using the Cut Property

- ▶ Let F be the set of all edges that satisfy the cut property.
- ▶ Is the graph induced by F connected? Yes.
- ▶ Can the graph induced by F contain a cycle? No.
- ▶ How many edges can F contain? $n - 1$

Using the Cut Property

- ▶ Let F be the set of all edges that satisfy the cut property.
- ▶ Is the graph induced by F connected? Yes.
- ▶ Can the graph induced by F contain a cycle? No.
- ▶ How many edges can F contain? $n - 1$
- ▶ F is the unique MST.
- ▶ Kruskal's and Prim's algorithms compute F efficiently.

Optimality of Kruskal's Algorithm

- ▶ Kruskal's algorithm:
 - ▶ Start with an empty set T of edges.
 - ▶ Process edges in E in non decreasing order of cost.
 - ▶ Add the next edge e to T only if adding e does not create a cycle .
Discard e if it creates a cycle.
- ▶ Claim: Kruskal's algorithm outputs an MST.

Optimality of Kruskal's Algorithm

- ▶ Kruskal's algorithm:
 - ▶ Start with an empty set T of edges.
 - ▶ Process edges in E in non decreasing order of cost.
 - ▶ Add the next edge e to T only if adding e does not create a cycle .
Discard e if it creates a cycle.
- ▶ Claim: Kruskal's algorithm outputs an MST.
 1. For every edge e added, demonstrate the existence of S and $V - S$ such that e and S satisfy the cut property.
 2. Prove that the algorithm computes a spanning tree.

Optimality of Prim's Algorithm

- ▶ Prim's algorithm: Maintain a tree (S, U)
 - ▶ Start with an arbitrary node $s \in S$ and $U = \emptyset$.
 - ▶ Add the node v to S and the edge e to U that minimize

$$\min_{e=(u,v), u \in S, v \notin S} c_e \equiv \min_{e \in \text{cut}(S)} c_e.$$

- ▶ Stop when $S = V$.
- ▶ Claim: Prim's algorithm outputs an MST.

Optimality of Prim's Algorithm

- ▶ Prim's algorithm: Maintain a tree (S, U)
 - ▶ Start with an arbitrary node $s \in S$ and $U = \emptyset$.
 - ▶ Add the node v to S and the edge e to U that minimize

$$\min_{e=(u,v), u \in S, v \notin S} c_e \equiv \min_{e \in \text{cut}(S)} c_e.$$

- ▶ Stop when $S = V$.
- ▶ Claim: Prim's algorithm outputs an MST.
 1. Prove that every edge inserted satisfies the cut property.
 2. Prove that the graph constructed is a spanning tree.

Cycle Property

- When can we be sure that an edge cannot be in any MST?

Cycle Property

- ▶ When can we be sure that an edge cannot be in any MST?
- ▶ Let C be any cycle in G and let $e = (v, w)$ be the most expensive edge in C .
- ▶ Claim: e does not belong to any MST of G .

Cycle Property

- ▶ When can we be sure that an edge cannot be in any MST?
- ▶ Let C be any cycle in G and let $e = (v, w)$ be the most expensive edge in C .
- ▶ Claim: e does not belong to any MST of G .
- ▶ Proof: exchange argument. If a supposed MST T contains e , show that there is a tree with smaller cost than T that does not contain e .

Optimality of the Reverse-Delete Algorithm

- ▶ Reverse-Delete algorithm: Maintain a set E' of edges.
 - ▶ Start with $E' = E$.
 - ▶ Process edges in non increasing order of cost.
 - ▶ Delete the next edge e from E' only if (V, E') is connected after removal .
 - ▶ Stop after processing all the edges.
- ▶ Claim: the Reverse-Delete algorithm outputs an MST.

Optimality of the Reverse-Delete Algorithm

- ▶ Reverse-Delete algorithm: Maintain a set E' of edges.
 - ▶ Start with $E' = E$.
 - ▶ Process edges in non increasing order of cost.
 - ▶ Delete the next edge e from E' only if (V, E') is connected after removal .
 - ▶ Stop after processing all the edges.
- ▶ Claim: the Reverse-Delete algorithm outputs an MST.
 1. Show that every edge deleted belongs to no MST.
 2. Prove that the graph remaining at the end is a spanning tree.

Comments on MST Algorithms

- ▶ To handle multiple edges with the same weight, perturb each length by a random infinitesimal amount.
- ▶ Any algorithm that constructs a spanning tree by including edges that satisfy the cut property and deleting edges that satisfy the cycle property will yield an **MST!**

Questions?

Trees and spanning trees
– Graph cut –

Teoria dos Grafos e Computabilidade

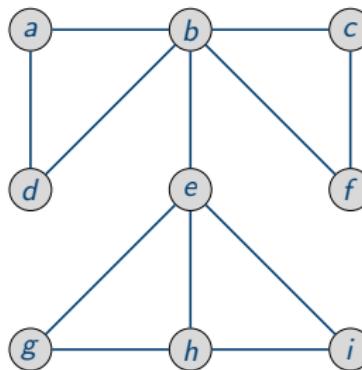
— Steiner Trees —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

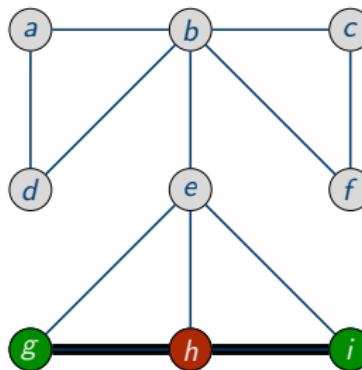
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimum size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



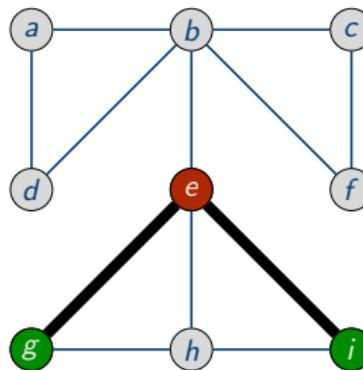
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimum size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



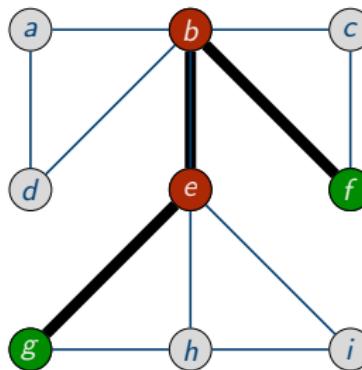
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimum size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



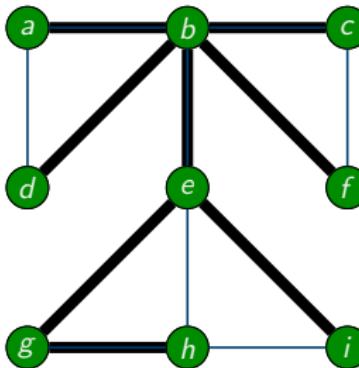
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimum size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimum size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



Is a spanning tree T' of G a Steiner tree?

Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimum size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree.

- ▶ The vertices in T are called terminals
- ▶ The vertices in $V \setminus T$ are called Steiner points
- ▶ Denote $n = |V|$, $m = |E|$ and $t = |T|$
- ▶ A minimum size:
 - ▶ Vertex cardinality: $|V'|$ or rather $|S| = |V' \setminus T|$ (default)
 - ▶ Edge cardinality: $|E'| = |V'| - 1$
 - ▶ Node weighted: Given $w : V \rightarrow \mathbb{N}$ minimize $w(S)$
 - ▶ Edge weighted: Given $w : E \rightarrow \mathbb{N}$ minimize $w(E')$

Questions?

Trees and spanning trees
– Steiner Trees –

Teoria dos Grafos e Computabilidade

— Network Flow —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Maximum Flow and Minimum Cut —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Maximum Flow and Minimum Cut

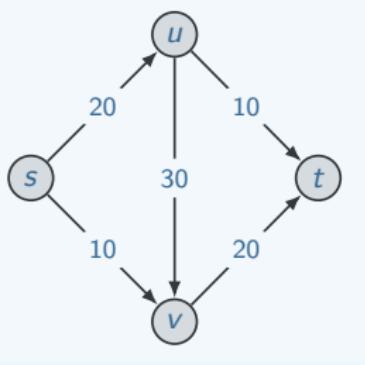
- ▶ Two rich algorithmic problems.
- ▶ Fundamental problems in combinatorial optimization.
- ▶ Beautiful mathematical duality between flows and cuts.
- ▶ Numerous non-trivial applications:
 - ▶ Bipartite matching .
 - ▶ Data mining.
 - ▶ Project selection.
 - ▶ Airline scheduling.
 - ▶ Baseball elimination .
 - ▶ Image segmentation .
 - ▶ Network connectivity .
 - ▶ Open-pit mining.
 - ▶ Network reliability.
 - ▶ Distributed computing.
 - ▶ Egalitarian stable matching.
 - ▶ Security of statistical data.
 - ▶ Network intrusion detection.
 - ▶ Multi-camera scene reconstruction.
 - ▶ Gene function prediction.

Flow Networks

- ▶ Use directed graphs to model transportation networks :
 - ▶ edges carry traffic and have capacities.
 - ▶ nodes act as switches.
 - ▶ *source* nodes generate traffic, *sink* nodes absorb traffic.

Flow Networks

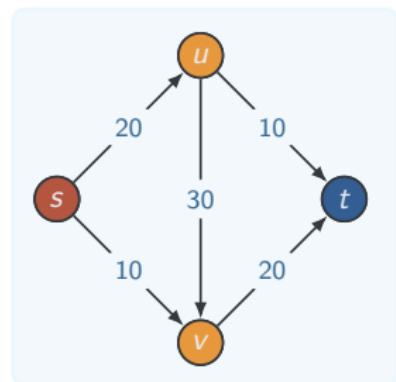
- ▶ Use directed graphs to model **transportation networks**:
 - ▶ edges carry traffic and have capacities.
 - ▶ nodes act as switches.
 - ▶ *source* nodes generate traffic, *sink* nodes absorb traffic.



- ▶ A **flow network** is a directed graph $G = (V, E)$
 - ▶ Each edge $e \in E$ has a capacity $c(e) > 0$.

Flow Networks

- ▶ Use directed graphs to model transportation networks:
 - ▶ edges carry traffic and have capacities.
 - ▶ nodes act as switches.
 - ▶ *source* nodes generate traffic, *sink* nodes absorb traffic.



- ▶ A **flow network** is a directed graph $G = (V, E)$
 - ▶ Each edge $e \in E$ has a capacity $c(e) > 0$.
 - ▶ There is a single **source** node $s \in V$.
 - ▶ There is a single **sink** node $t \in V$.
 - ▶ Nodes other than s and t are **internal**.

Defining Flow

- ▶ In a flow network $G = (V, E)$, an **s-t flow** is a function $f : E \rightarrow \mathbb{R}^+$ such that
 - (i) **Capacity conditions** For each $e \in E$, $0 \leq f(e) \leq c(e)$.
 - (ii) **Conservation conditions** For each internal node v ,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

- ▶ The **value** of a flow is $\nu(f) = \sum_{e \text{ out of } s} f(e)$.

Defining Flow

- ▶ In a flow network $G = (V, E)$, an **s-t flow** is a function $f : E \rightarrow \mathbb{R}^+$ such that
 - (i) **Capacity conditions** For each $e \in E$, $0 \leq f(e) \leq c(e)$.
 - (ii) **Conservation conditions** For each internal node v ,

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e)$$

- ▶ The **value** of a flow is $\nu(f) = \sum_{e \text{ out of } s} f(e)$.
- ▶ Useful notation:

$$f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$$

For $S \subseteq V$,

$$f^{\text{out}}(S) = \sum_{e \text{ out of } S} f(e)$$

$$f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e)$$

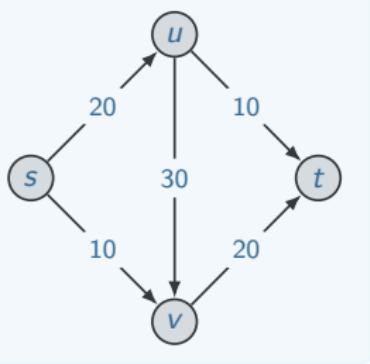
$$f^{\text{in}}(S) = \sum_{e \text{ into } S} f(e)$$

Maximum-Flow Problem

MAXIMUM FLOW

INSTANCE A flow network G

SOLUTION The flow with largest value in G



► Assumptions :

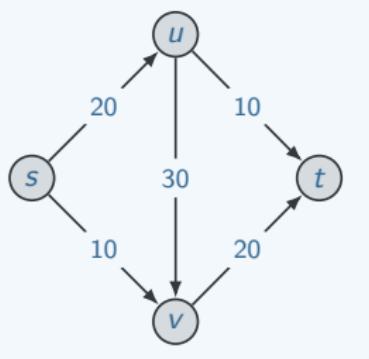
1. No edges enter s , no edges leave t .

Maximum-Flow Problem

MAXIMUM FLOW

INSTANCE A flow network G

SOLUTION The flow with largest value in G



► Assumptions :

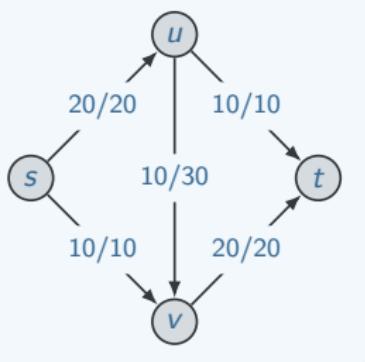
1. No edges enter s , no edges leave t .
2. There is at least one edge incident on each node.

Maximum-Flow Problem

MAXIMUM FLOW

INSTANCE A flow network G

SOLUTION The flow with largest value in G



► Assumptions :

1. No edges enter s , no edges leave t .
2. There is at least one edge incident on each node.
3. All edge capacities are integers .

Questions?

Network Flow

- Maximum Flow and Minimum Cut –



Teoria dos Grafos e Computabilidade

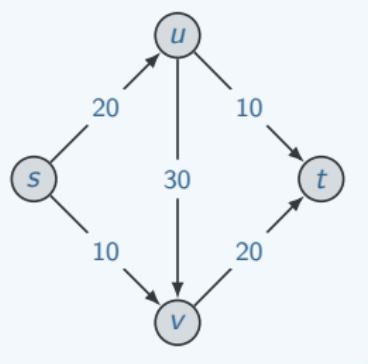
— Ford-Fulkerson Algorithm —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

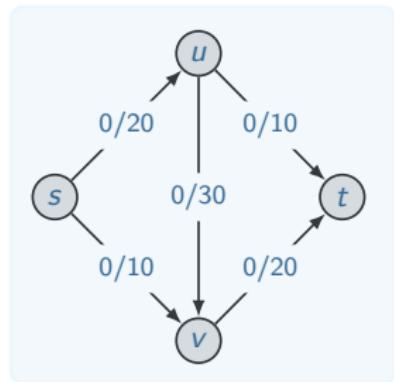
Developing the Algorithm

- A **flow network** is a directed graph $G = (V, E)$



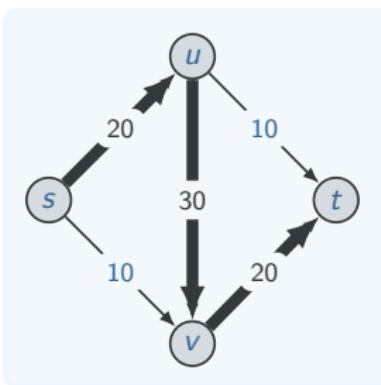
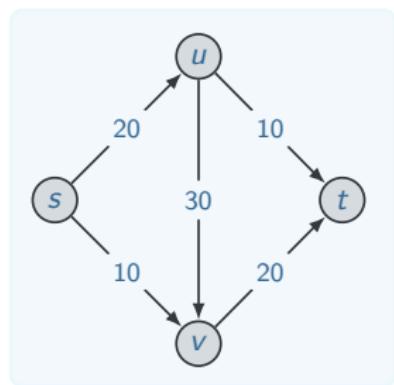
Developing the Algorithm

- ▶ A **flow network** is a directed graph $G = (V, E)$
- ▶ Let us take a greedy approach.
 1. Start with **zero flow** along all edges.



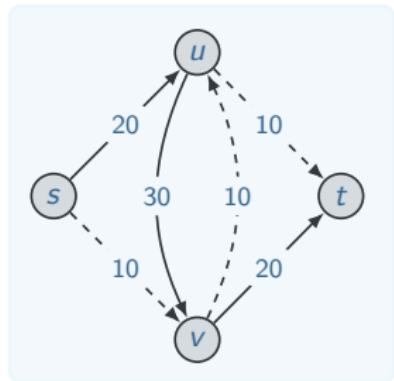
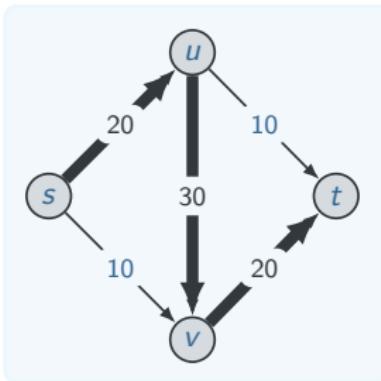
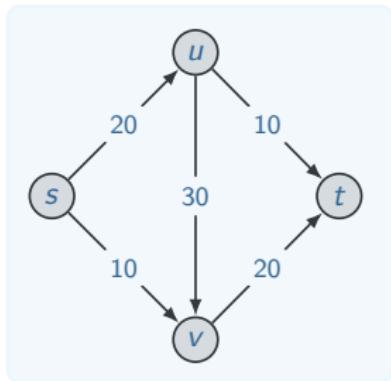
Developing the Algorithm

- ▶ A flow network is a directed graph $G = (V, E)$
- ▶ Let us take a greedy approach.
 1. Start with zero flow along all edges.
 2. Find an $s-t$ path and push as much flow along it as possible.



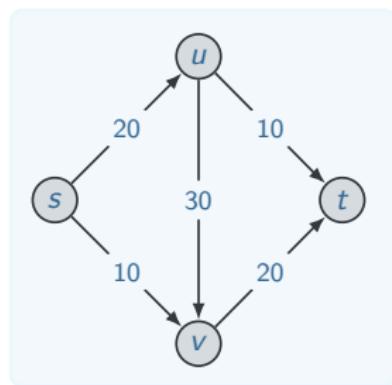
Developing the Algorithm

- ▶ A **flow network** is a directed graph $G = (V, E)$
- ▶ Let us take a greedy approach.
 1. Start with **zero flow** along all edges.
 2. Find an $s-t$ path and push **as much flow along it as possible**.
 3. **Key idea**: Push flow along edges with **leftover capacity** and **undo flow** on edges already carrying flow.



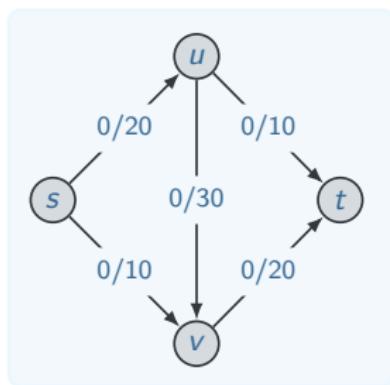
Residual Graph

- Given a flow network $G = (V, E)$ and a flow f on G , the residual graph G_f of G with respect to f is a directed graph such that
 - (i) Nodes – G_f has the same nodes as G .



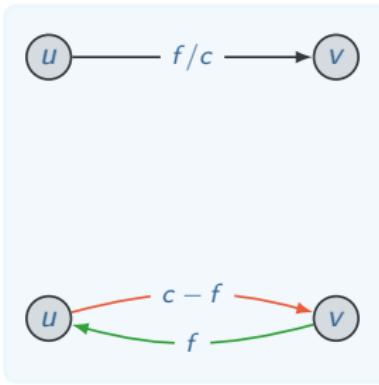
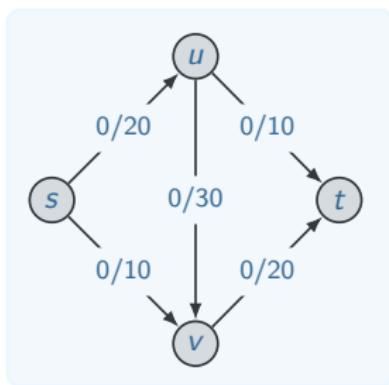
Residual Graph

- Given a flow network $G = (V, E)$ and a flow f on G , the residual graph G_f of G with respect to f is a directed graph such that
 - (i) Nodes – G_f has the same nodes as G .



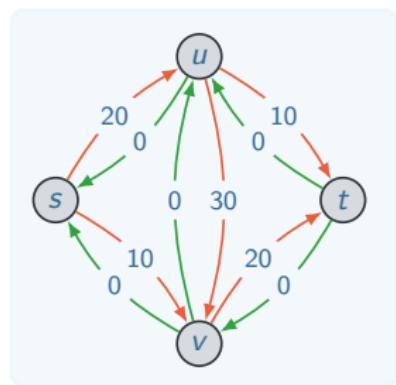
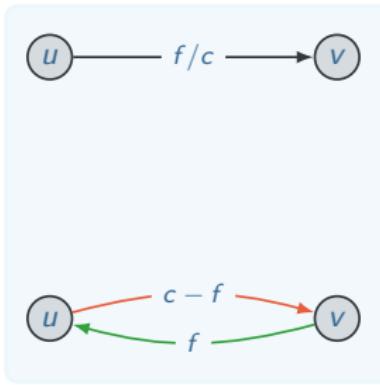
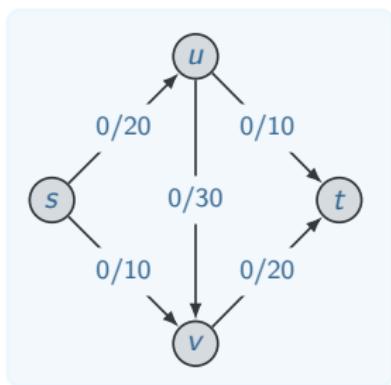
Residual Graph

- Given a flow network $G = (V, E)$ and a flow f on G , the residual graph G_f of G with respect to f is a directed graph such that
 - (i) Nodes – G_f has the same nodes as G .
 - (ii) Forward edges – For each edge $e = (u, v) \in E$ such that $f(e) < c(e)$, G_f contains the edge (u, v) with a residual capacity $c(e) - f(e)$.
 - (iii) Backward edges – For each edge $e \in E$ such that $f(e) > 0$, G_f contains the edge $e' = (v, u)$ with a residual capacity $f(e)$.



Residual Graph

- Given a flow network $G = (V, E)$ and a flow f on G , the residual graph G_f of G with respect to f is a directed graph such that
 - (i) Nodes – G_f has the same nodes as G .
 - (ii) Forward edges – For each edge $e = (u, v) \in E$ such that $f(e) < c(e)$, G_f contains the edge (u, v) with a residual capacity $c(e) - f(e)$.
 - (iii) Backward edges – For each edge $e \in E$ such that $f(e) > 0$, G_f contains the edge $e' = (v, u)$ with a residual capacity $f(e)$.



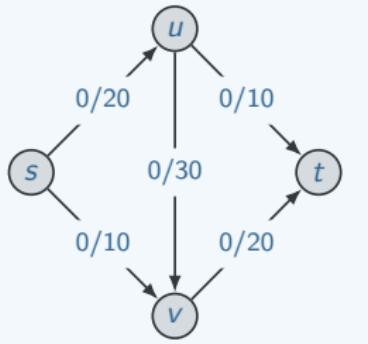
Augmenting Paths in a Residual Graph

- Let P be a simple $s-t$ path in G_f .
- $\text{bottleneck}(P, f)$ is the minimum residual capacity of any edge in P .
- The following operation $\text{augment}(f, P)$ yields a new flow f' in G :

Algorithm: Augmented path

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.

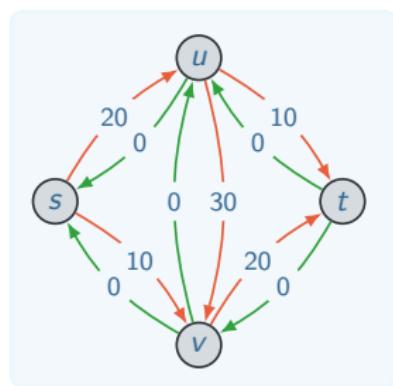
output: The distances of the vertices from s



```
1 Let  $b = \text{bottleneck}(P, f)$  ;  
2 foreach edge  $e = (u, v) \in P$  do  
3   | if  $e$  is a forward edge then  
4   |   | increase  $f(e)$  in  $G$  by  $b$   
5   | else if  $e$  is a backward edge then  
6   |   | decrease  $f(e)$  in  $G$  by  $b$ ;  
7 end  
8
```

Augmenting Paths in a Residual Graph

- Let P be a simple $s-t$ path in G_f .
- $\text{bottleneck}(P, f)$ is the minimum residual capacity of any edge in P .
- The following operation $\text{augment}(f, P)$ yields a new flow f' in G :

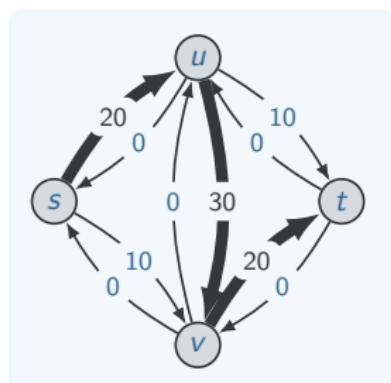


Algorithm: Augmented path

```
input : A graph  $G = (V, E)$ , a path  $P$  and a
       source  $s$  and a sink  $t$  nodes.
output: The distances of the vertices from  $s$ 
1 Let  $b = \text{bottleneck}(P, f)$  ;
2 foreach edge  $e = (u, v) \in P$  do
3   | if  $e$  is a forward edge then
4     |   | increase  $f(e)$  in  $G$  by  $b$ 
5   | else if  $e$  is a backward edge then
6     |   | decrease  $f(e)$  in  $G$  by  $b$ ;
7 end
8
```

Augmenting Paths in a Residual Graph

- Let P be a simple $s-t$ path in G_f .
- $\text{bottleneck}(P, f)$ is the minimum residual capacity of any edge in P .
- The following operation $\text{augment}(f, P)$ yields a new flow f' in G :



Algorithm: Augmented path

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.

output: The distances of the vertices from s

```
1 Let  $b = \text{bottleneck}(P, f)$  ;  
2 foreach edge  $e = (u, v) \in P$  do  
3   | if  $e$  is a forward edge then  
4   |   | increase  $f(e)$  in  $G$  by  $b$   
5   | else if  $e$  is a backward edge then  
6   |   | decrease  $f(e)$  in  $G$  by  $b$ ;  
7 end  
8
```

Augmenting Paths in a Residual Graph

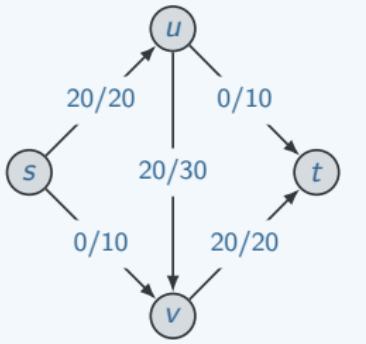
- Let P be a simple $s-t$ path in G_f .
- $\text{bottleneck}(P, f)$ is the minimum residual capacity of any edge in P .
- The following operation $\text{augment}(f, P)$ yields a new flow f' in G :

Algorithm: Augmented path

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.

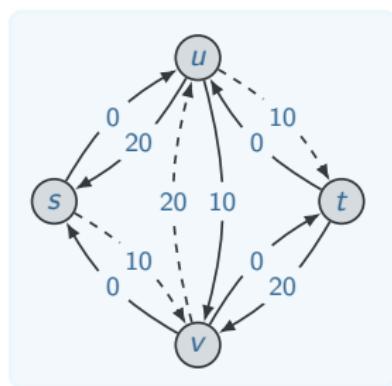
output: The distances of the vertices from s

```
1 Let  $b = \text{bottleneck}(P, f)$  ;  
2 foreach edge  $e = (u, v) \in P$  do  
3   | if  $e$  is a forward edge then  
4     |   | increase  $f(e)$  in  $G$  by  $b$   
5   | else if  $e$  is a backward edge then  
6     |   | decrease  $f(e)$  in  $G$  by  $b$ ;  
7 end  
8
```



Augmenting Paths in a Residual Graph

- Let P be a simple $s-t$ path in G_f .
- $\text{bottleneck}(P, f)$ is the minimum residual capacity of any edge in P .
- The following operation $\text{augment}(f, P)$ yields a new flow f' in G :



Algorithm: Augmented path

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.

output: The distances of the vertices from s

```
1 Let  $b = \text{bottleneck}(P, f)$  ;
2 foreach edge  $e = (u, v) \in P$  do
3   if  $e$  is a forward edge then
4     | increase  $f(e)$  in  $G$  by  $b$ 
5   else if  $e$  is a backward edge then
6     | decrease  $f(e)$  in  $G$  by  $b$ ;
7 end
8
```

Augmenting Paths in a Residual Graph

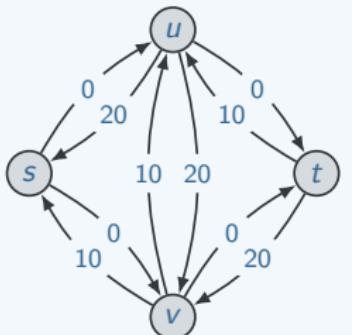
- Let P be a simple $s-t$ path in G_f .
- $\text{bottleneck}(P, f)$ is the minimum residual capacity of any edge in P .
- The following operation $\text{augment}(f, P)$ yields a new flow f' in G :

Algorithm: Augmented path

input : A graph $G = (V, E)$, a path P and a source s and a sink t nodes.

output: The distances of the vertices from s

```
1 Let  $b = \text{bottleneck}(P, f)$  ;  
2 foreach edge  $e = (u, v) \in P$  do  
3   | if  $e$  is a forward edge then  
4     |   | increase  $f(e)$  in  $G$  by  $b$   
5   | else if  $e$  is a backward edge then  
6     |   | decrease  $f(e)$  in  $G$  by  $b$ ;  
7 end  
8
```



Correctness of $\text{augment}(f, P)$

- ▶ A simple s - t path in the residual graph is an augmenting path.
- ▶ Let f' be the flow returned by $\text{augment}(f, P)$.
- ▶ Claim: f' is a flow. Verify capacity and conservation conditions.

Correctness of $\text{augment}(f, P)$

- ▶ A simple s - t path in the residual graph is an augmenting path.
- ▶ Let f' be the flow returned by $\text{augment}(f, P)$.
- ▶ Claim: f' is a flow. Verify capacity and conservation conditions.
 - ▶ Only need to check edges and internal nodes in P .

Correctness of $\text{augment}(f, P)$

- ▶ A simple s - t path in the residual graph is an augmenting path.
- ▶ Let f' be the flow returned by $\text{augment}(f, P)$.
- ▶ Claim: f' is a flow. Verify capacity and conservation conditions.
 - ▶ Only need to check edges and internal nodes in P .
 - ▶ Capacity condition on $e = (u, v) \in G_f$: Note that $\text{bottleneck}(P, f) \leq \text{residual capacity of } (u, v)$.

Correctness of $\text{augment}(f, P)$

- ▶ A simple s - t path in the residual graph is an augmenting path.
- ▶ Let f' be the flow returned by $\text{augment}(f, P)$.
- ▶ Claim: f' is a flow. Verify capacity and conservation conditions.
 - ▶ Only need to check edges and internal nodes in P .
 - ▶ Capacity condition on $e = (u, v) \in G_f$: Note that $\text{bottleneck}(P, f) \leq \text{residual capacity of } (u, v)$.
 - ▶ e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.

Correctness of $\text{augment}(f, P)$

- ▶ A simple s - t path in the residual graph is an augmenting path.
- ▶ Let f' be the flow returned by $\text{augment}(f, P)$.
- ▶ Claim: f' is a flow. Verify capacity and conservation conditions.
 - ▶ Only need to check edges and internal nodes in P .
 - ▶ Capacity condition on $e = (u, v) \in G_f$: Note that $\text{bottleneck}(P, f) \leq \text{residual capacity of } (u, v)$.
 - ▶ e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.
 - ▶ e is a backward edge: $c(e) \geq f(e) \geq f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$.

Correctness of $\text{augment}(f, P)$

- ▶ A simple s - t path in the residual graph is an augmenting path.
- ▶ Let f' be the flow returned by $\text{augment}(f, P)$.
- ▶ Claim: f' is a flow. Verify capacity and conservation conditions.
 - ▶ Only need to check edges and internal nodes in P .
 - ▶ Capacity condition on $e = (u, v) \in G_f$: Note that $\text{bottleneck}(P, f) \leq \text{residual capacity of } (u, v)$.
 - ▶ e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.
 - ▶ e is a backward edge: $c(e) \geq f(e) \geq f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$.
 - ▶ Conservation condition on internal node $v \in P$.

Correctness of $\text{augment}(f, P)$

- ▶ A simple s - t path in the residual graph is an augmenting path.
- ▶ Let f' be the flow returned by $\text{augment}(f, P)$.
- ▶ Claim: f' is a flow. Verify capacity and conservation conditions.
 - ▶ Only need to check edges and internal nodes in P .
 - ▶ Capacity condition on $e = (u, v) \in G_f$: Note that $\text{bottleneck}(P, f) \leq \text{residual capacity of } (u, v)$.
 - ▶ e is a forward edge: $0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(P, f) \leq f(e) + (c(e) - f(e)) = c(e)$.
 - ▶ e is a backward edge: $c(e) \geq f(e) \geq f'(e) = f(e) - \text{bottleneck}(P, f) \geq f(e) - f(e) = 0$.
 - ▶ Conservation condition on internal node $v \in P$. Four cases to work out.

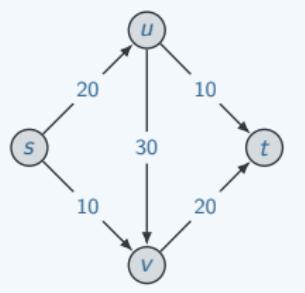
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

- 1 $f(e) = 0, \forall e \in E;$
 - 2 **while** there is a path $s-t$ in the residual graph G_f **do**
 - 3 Let P be a simple $s-t$ path in G_f ;
 - 4 $f' = \text{augment}(f, P);$
 - 5 Update f to be f' ;
 - 6 Update the residual graph G_f to be $G_{f'}$;
 - 7 **end**
 - 8 **return** f ;
-



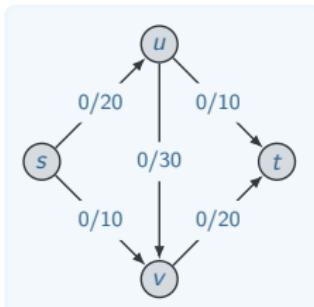
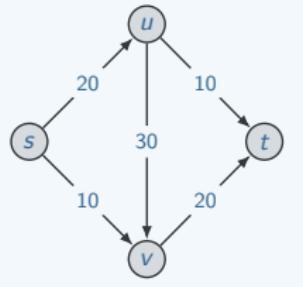
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
1  $f(e) = 0, \forall e \in E;$ 
2 while there is a path  $s-t$  in the residual graph  $G_f$  do
3   Let  $P$  be a simple  $s-t$  path in  $G_f$ ;
4    $f' = \text{augment}(f, P);$ 
5   Update  $f$  to be  $f'$ ;
6   Update the residual graph  $G_f$  to be  $G_{f'}$ ;
7 end
8 return  $f$ ;
```



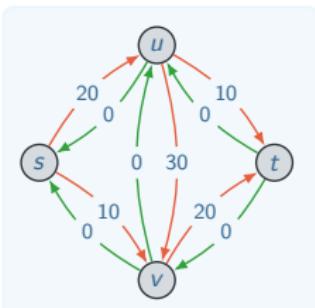
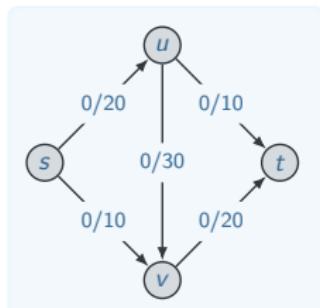
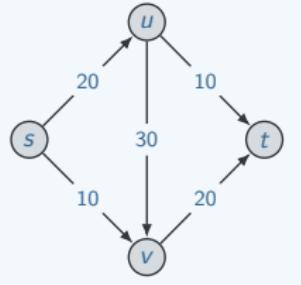
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
1  $f(e) = 0, \forall e \in E;$ 
2 while there is a path  $s-t$  in the residual graph  $G_f$  do
3   Let  $P$  be a simple  $s-t$  path in  $G_f$ ;
4    $f' = \text{augment}(f, P);$ 
5   Update  $f$  to be  $f'$ ;
6   Update the residual graph  $G_f$  to be  $G_{f'}$ ;
7 end
8 return  $f$ ;
```



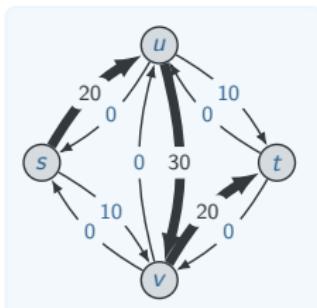
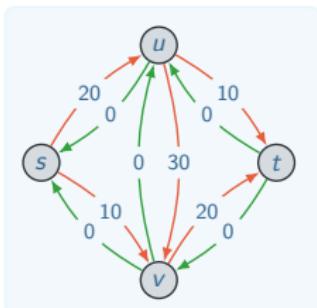
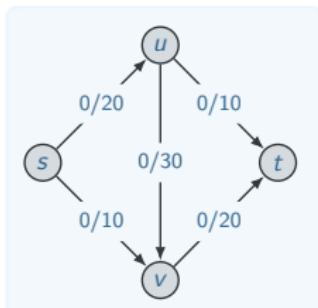
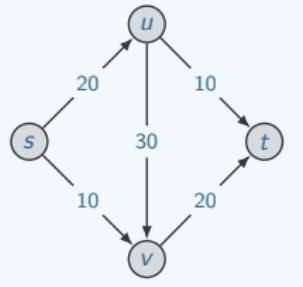
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

- 1 $f(e) = 0, \forall e \in E;$
 - 2 **while** there is a path $s-t$ in the residual graph G_f **do**
 - 3 Let P be a simple $s-t$ path in G_f ;
 - 4 $f' = \text{augment}(f, P);$
 - 5 Update f to be f' ;
 - 6 Update the residual graph G_f to be $G_{f'}$;
 - 7 **end**
 - 8 **return** f ;
-



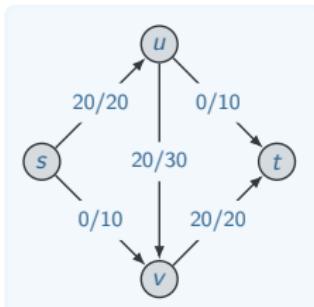
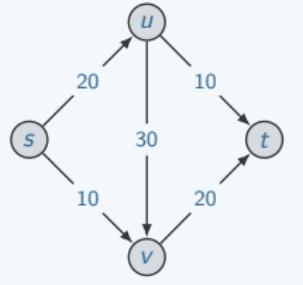
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
1  $f(e) = 0, \forall e \in E;$ 
2 while there is a path  $s-t$  in the residual graph  $G_f$  do
3   Let  $P$  be a simple  $s-t$  path in  $G_f$ ;
4    $f' = \text{augment}(f, P);$ 
5   Update  $f$  to be  $f'$ ;
6   Update the residual graph  $G_f$  to be  $G_{f'}$ ;
7 end
8 return  $f$ ;
```



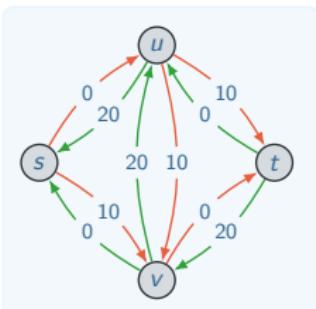
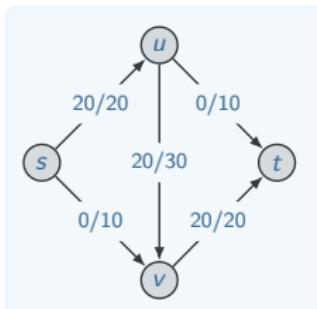
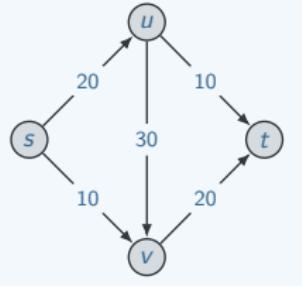
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
1  $f(e) = 0, \forall e \in E;$ 
2 while there is a path  $s-t$  in the residual graph  $G_f$  do
3   Let  $P$  be a simple  $s-t$  path in  $G_f$ ;
4    $f' = \text{augment}(f, P);$ 
5   Update  $f$  to be  $f'$ ;
6   Update the residual graph  $G_f$  to be  $G_{f'}$ ;
7 end
8 return  $f$ ;
```



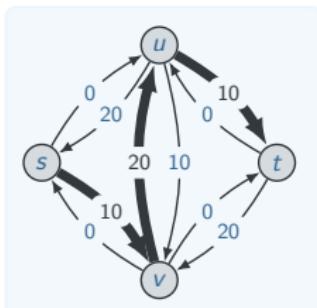
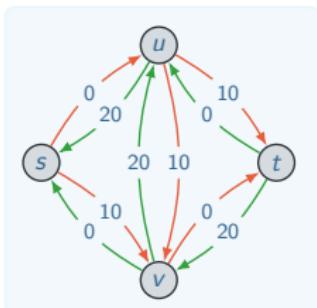
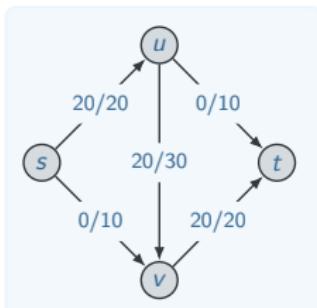
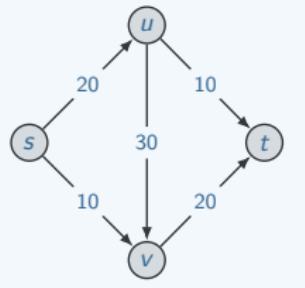
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

- 1 $f(e) = 0, \forall e \in E;$
 - 2 **while** there is a path $s-t$ in the residual graph G_f **do**
 - 3 Let P be a simple $s-t$ path in G_f ;
 - 4 $f' = \text{augment}(f, P);$
 - 5 Update f to be f' ;
 - 6 Update the residual graph G_f to be $G_{f'}$;
 - 7 **end**
 - 8 **return** f ;
-



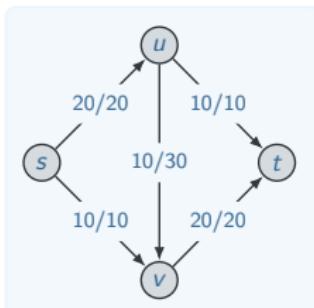
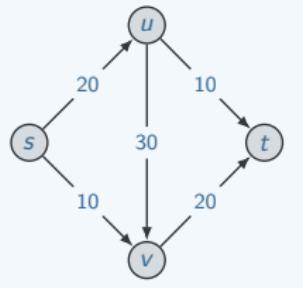
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
1  $f(e) = 0, \forall e \in E;$ 
2 while there is a path  $s-t$  in the residual graph  $G_f$  do
3   Let  $P$  be a simple  $s-t$  path in  $G_f$ ;
4    $f' = \text{augment}(f, P);$ 
5   Update  $f$  to be  $f'$ ;
6   Update the residual graph  $G_f$  to be  $G_{f'}$ ;
7 end
8 return  $f$ ;
```



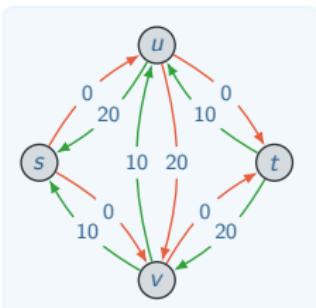
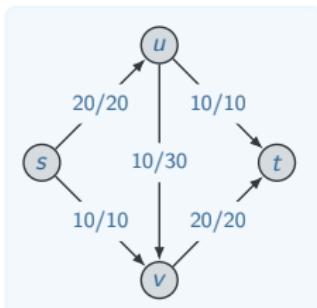
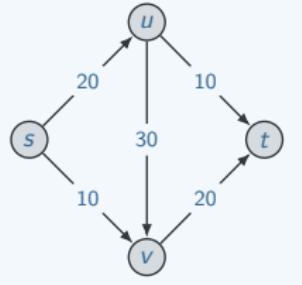
Ford-Fulkerson Algorithm

Algorithm: Ford-Fulkerson Algorithm

input : A graph $G = (V, E)$, a source s and a sink t nodes.

output: The flow f

```
1  $f(e) = 0, \forall e \in E;$ 
2 while there is a path  $s-t$  in the residual graph  $G_f$  do
3   Let  $P$  be a simple  $s-t$  path in  $G_f$ ;
4    $f' = \text{augment}(f, P);$ 
5   Update  $f$  to be  $f'$ ;
6   Update the residual graph  $G_f$  to be  $G_{f'}$ ;
7 end
8 return  $f$ ;
```



Termination of the Ford-Fulkerson Algorithm

- ▶ Claim: at each stage, flow values and residual capacities are integers.

Termination of the Ford-Fulkerson Algorithm

- ▶ Claim: at each stage, flow values and residual capacities are integers. Prove by induction.

Termination of the Ford-Fulkerson Algorithm

- ▶ Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- ▶ Claim: Flow value strictly increases when we apply $\text{augment}(f, P)$.

Termination of the Ford-Fulkerson Algorithm

- ▶ Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- ▶ Claim: Flow value strictly increases when we apply $\text{augment}(f, P)$.
 $v(f') = v(f) + \text{bottleneck}(P, f) > v(f).$

Termination of the Ford-Fulkerson Algorithm

- ▶ Claim: at each stage, flow values and residual capacities are integers. Prove by induction.
- ▶ Claim: Flow value strictly increases when we apply $\text{augment}(f, P)$.
 $v(f') = v(f) + \text{bottleneck}(P, f) > v(f)$.
- ▶ Claim: Maximum value of any flow is
 $C = \sum_{e \text{ out of } s} c(e)$.

Correctness of the Ford-Fulkerson Algorithm

- ▶ How large can the flow be?

Correctness of the Ford-Fulkerson Algorithm

- ▶ How large can the flow be?
- ▶ Can we characterise the magnitude of the flow in terms of the structure of the graph? For example, for every flow f , $\nu(f) \leq C = \sum_{e \text{ out of } s} c(e)$.
- ▶ Is there a better bound?

Correctness of the Ford-Fulkerson Algorithm

- ▶ How large can the flow be?
- ▶ Can we characterise the magnitude of the flow in terms of the structure of the graph? For example, for every flow f , $\nu(f) \leq C = \sum_{e \text{ out of } s} c(e)$.
- ▶ Is there a better bound?
- ▶ Idea: An **s - t cut** is a partition of V into sets A and B such that $s \in A$ and $t \in B$.
 - ▶ Capacity of the cut (A, B) is $c(A, B) = \sum_{e \text{ out of } A} c(e)$.
 - ▶ Intuition: For every flow f , $\nu(f) \leq c(A, B)$.

Fun Facts about Cuts

- ▶ Let f be any $s-t$ flow and (A, B) any $s-t$ cut.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
 - ▶ For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
 - ▶ For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
 - ▶ $\nu(f) = \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v))$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
 - ▶ For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
 - ▶ $\nu(f) = \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v))$.
 - ▶ An edge e that has both ends in A or both ends out of A does not contribute.
 - ▶ An edge e that has its tail in A contributes $f(e)$.
 - ▶ An edge e that has its head in A contributes $-f(e)$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
 - ▶ For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
 - ▶ $\nu(f) = \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v))$.
 - ▶ An edge e that has both ends in A or both ends out of A does not contribute.
 - ▶ An edge e that has its tail in A contributes $f(e)$.
 - ▶ An edge e that has its head in A contributes $-f(e)$.
 - ▶ $\sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v)) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
 - ▶ For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
 - ▶ $\nu(f) = \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v))$.
 - ▶ An edge e that has both ends in A or both ends out of A does not contribute.
 - ▶ An edge e that has its tail in A contributes $f(e)$.
 - ▶ An edge e that has its head in A contributes $-f(e)$.
 - ▶ $\sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v)) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
- ▶ Corollary: $\nu(f) = f^{\text{in}}(B) - f^{\text{out}}(B)$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
 - ▶ For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
 - ▶ $\nu(f) = \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v))$.
 - ▶ An edge e that has both ends in A or both ends out of A does not contribute.
 - ▶ An edge e that has its tail in A contributes $f(e)$.
 - ▶ An edge e that has its head in A contributes $-f(e)$.
 - ▶ $\sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v)) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
- ▶ Corollary: $\nu(f) = f^{\text{in}}(B) - f^{\text{out}}(B)$.
- ▶ $\nu(f) \leq c(A, B)$.

Fun Facts about Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut.
- ▶ Claim: $\nu(f) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
 - ▶ $\nu(f) = f^{\text{out}}(s)$ and $f^{\text{in}}(s) = 0 \Rightarrow \nu(f) = f^{\text{out}}(s) - f^{\text{in}}(s)$.
 - ▶ For every other node $v \in A$, $f^{\text{out}}(v) - f^{\text{in}}(v) = 0$.
 - ▶ $\nu(f) = \sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v))$.
 - ▶ An edge e that has both ends in A or both ends out of A does not contribute.
 - ▶ An edge e that has its tail in A contributes $f(e)$.
 - ▶ An edge e that has its head in A contributes $-f(e)$.
 - ▶ $\sum_{v \in A} (f^{\text{out}}(v) - f^{\text{in}}(v)) = \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ into } A} f(e) = f^{\text{out}}(A) - f^{\text{in}}(A)$.
- ▶ Corollary: $\nu(f) = f^{\text{in}}(B) - f^{\text{out}}(B)$.
- ▶ $\nu(f) \leq c(A, B)$.

$$\begin{aligned}\nu(f) &= f^{\text{out}}(A) - f^{\text{in}}(A) \leq f^{\text{out}}(A) = \sum_{e \text{ out of } A} f(e) \\ &\leq \sum_{e \text{ out of } A} c(e) = c(A, B).\end{aligned}$$

Max-Flows and Min-Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut. We proved $\nu(f) \leq c(A, B)$.

Max-Flows and Min-Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut. We proved $\nu(f) \leq c(A, B)$.
- ▶ Very strong statement: The value of every flow is \leq capacity of any cut.

Max-Flows and Min-Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut. We proved $\nu(f) \leq c(A, B)$.
- ▶ Very strong statement: The value of every flow is \leq capacity of any cut.
- ▶ Corollary: The maximum flow is, at most, the smallest capacity of a cut.

Max-Flows and Min-Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut. We proved $\nu(f) \leq c(A, B)$.
- ▶ Very strong statement: The value of every flow is \leq capacity of any cut.
- ▶ Corollary: The maximum flow is, at most, the smallest capacity of a cut.
- ▶ Question: Is the reverse true? Is the smallest capacity of a cut at most the maximum flow?

Max-Flows and Min-Cuts

- ▶ Let f be any s - t flow and (A, B) any s - t cut. We proved $\nu(f) \leq c(A, B)$.
- ▶ Very strong statement: The value of every flow is \leq capacity of any cut.
- ▶ Corollary: The maximum flow is, at most, the smallest capacity of a cut.
- ▶ Question: Is the reverse true? Is the smallest capacity of a cut at most the maximum flow?
- ▶ Answer: Yes, and the Ford-Fulkerson algorithm computes this cut!

Flows and Cuts

- ▶ Let \bar{f} denote the flow computed by the Ford-Fulkerson algorithm .
- ▶ Enough to show $\exists s-t$ cut (A^*, B^*) such that $\nu(\bar{f}) = c(A^*, B^*)$.
- ▶ When the algorithm terminates, the residual graph has no $s-t$ path .

Flows and Cuts

- ▶ Let \bar{f} denote the flow computed by the Ford-Fulkerson algorithm .
- ▶ Enough to show $\exists s-t$ cut (A^*, B^*) such that $\nu(\bar{f}) = c(A^*, B^*)$.
- ▶ When the algorithm terminates, the residual graph has no $s-t$ path .
- ▶ Claim: If f is an $s-t$ flow such that G_f has no $s-t$ path, then there is an $s-t$ cut (A^*, B^*) such that $\nu(f) = c(A^*, B^*)$.
 - ▶ Claim applies to *any* flow f such that G_f has no $s-t$ path, and not just to the flow \bar{f} computed by the Ford-Fulkerson algorithm.

Proof of Claim Relating Flows to Cuts

- ▶ Claim: f is an s - t flow and G_f has no s - t path $\Rightarrow \exists$ s - t cut (A^*, B^*) , $\nu(f) = c(A^*, B^*)$.
- ▶ A^* = set of nodes reachable from s in G_f ,
 $B^* = V - A^*$.

Proof of Claim Relating Flows to Cuts

- ▶ Claim: f is an s - t flow and G_f has no s - t path $\Rightarrow \exists$ s - t cut (A^*, B^*) , $\nu(f) = c(A^*, B^*)$.
- ▶ A^* = set of nodes reachable from s in G_f ,
 $B^* = V - A^*$.
- ▶ Claim: (A^*, B^*) is an s - t cut.

Proof of Claim Relating Flows to Cuts

- ▶ Claim: f is an s - t flow and G_f has no s - t path $\Rightarrow \exists$ s - t cut (A^*, B^*) , $\nu(f) = c(A^*, B^*)$.
- ▶ A^* = set of nodes reachable from s in G_f ,
 $B^* = V - A^*$.
- ▶ Claim: (A^*, B^*) is an s - t cut.
- ▶ Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then

.

Proof of Claim Relating Flows to Cuts

- ▶ Claim: f is an s - t flow and G_f has no s - t path $\Rightarrow \exists$ s - t cut (A^*, B^*) , $\nu(f) = c(A^*, B^*)$.
- ▶ A^* = set of nodes reachable from s in G_f ,
 $B^* = V - A^*$.
- ▶ Claim: (A^*, B^*) is an s - t cut.
- ▶ Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.

Proof of Claim Relating Flows to Cuts

- ▶ Claim: f is an s - t flow and G_f has no s - t path $\Rightarrow \exists$ s - t cut (A^*, B^*) , $\nu(f) = c(A^*, B^*)$.
- ▶ A^* = set of nodes reachable from s in G_f ,
 $B^* = V - A^*$.
- ▶ Claim: (A^*, B^*) is an s - t cut.
- ▶ Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.
- ▶ Claim: If $e' = (u', v')$ such that $u' \in B^*$, $v' \in A^*$, then .

Proof of Claim Relating Flows to Cuts

- ▶ Claim: f is an s - t flow and G_f has no s - t path $\Rightarrow \exists$ s - t cut (A^*, B^*) , $\nu(f) = c(A^*, B^*)$.
- ▶ A^* = set of nodes reachable from s in G_f ,
 $B^* = V - A^*$.
- ▶ Claim: (A^*, B^*) is an s - t cut.
- ▶ Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.
- ▶ Claim: If $e' = (u', v')$ such that $u' \in B^*$, $v' \in A^*$, then $f(e') = 0$.

Proof of Claim Relating Flows to Cuts

- ▶ Claim: f is an s - t flow and G_f has no s - t path $\Rightarrow \exists$ s - t cut (A^*, B^*) , $\nu(f) = c(A^*, B^*)$.
- ▶ A^* = set of nodes reachable from s in G_f ,
 $B^* = V - A^*$.
- ▶ Claim: (A^*, B^*) is an s - t cut.
- ▶ Claim: If $e = (u, v)$ such that $u \in A^*$, $v \in B^*$, then $f(e) = c(e)$.
- ▶ Claim: If $e' = (u', v')$ such that $u' \in B^*$, $v' \in A^*$, then $f(e') = 0$.
- ▶ Claim: $\nu(f) = c(A^*, B^*)$.

Max-Flow Min-Cut Theorem

- ▶ The flow \bar{f} computed by the Ford-Fulkerson algorithm is a maximum flow .
- ▶ Given a flow of maximum value, we can compute a minimum $s-t$ cut .

Max-Flow Min-Cut Theorem

- ▶ The flow \bar{f} computed by the Ford-Fulkerson algorithm is a maximum flow.
- ▶ Given a flow of maximum value, we can compute a minimum $s-t$ cut.
- ▶ In every flow network, there is a flow f and a cut (A, B) such that $\nu(f) = c(A, B)$.
- ▶ Max-Flow Min-Cut Theorem : in every flow network, the maximum value of an $s-t$ flow is equal to the minimum capacity of an $s-t$ cut.

Max-Flow Min-Cut Theorem

- ▶ The flow \bar{f} computed by the Ford-Fulkerson algorithm is a maximum flow.
- ▶ Given a flow of maximum value, we can compute a minimum $s-t$ cut.
- ▶ In every flow network, there is a flow f and a cut (A, B) such that $\nu(f) = c(A, B)$.
- ▶ Max-Flow Min-Cut Theorem : in every flow network, the maximum value of an $s-t$ flow is equal to the minimum capacity of an $s-t$ cut.
- ▶ Corollary: If all capacities in a flow network are integers, then there is a maximum flow f where every flow value $f(e)$ is an integer.

Questions?

Network Flow

– Ford-Fulkerson Algorithm –

Teoria dos Grafos e Computabilidade

— Scaling Max-Flow Algorithm —

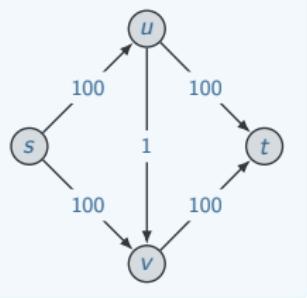
Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

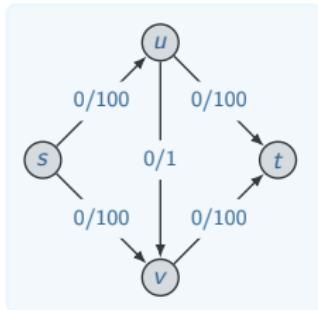
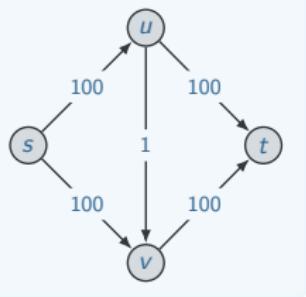
Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

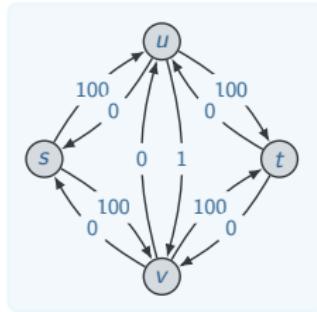
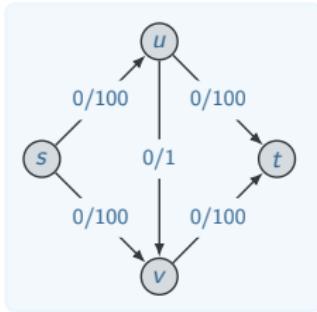
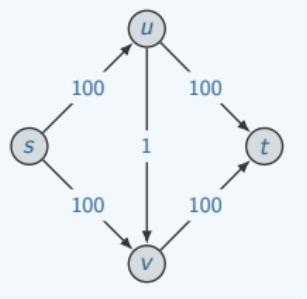
Bad Augmenting Paths



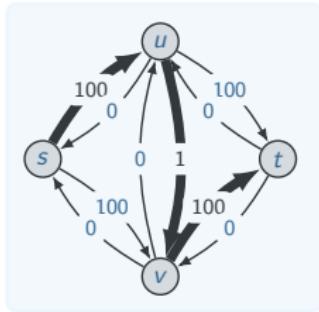
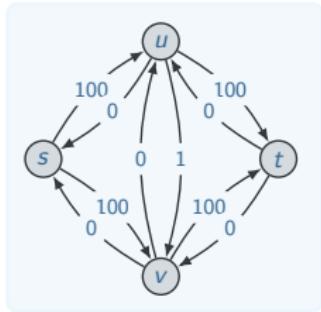
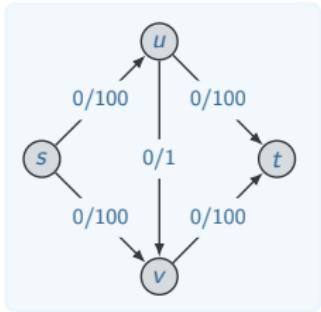
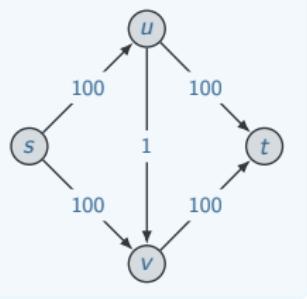
Bad Augmenting Paths



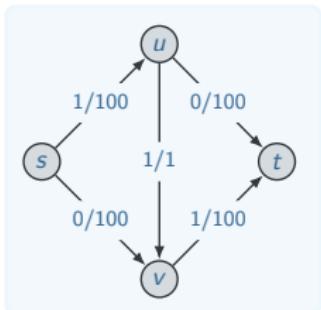
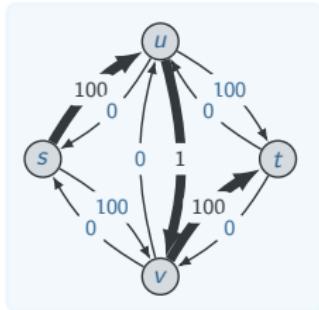
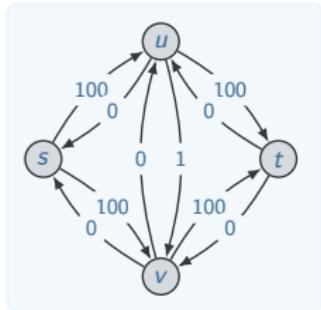
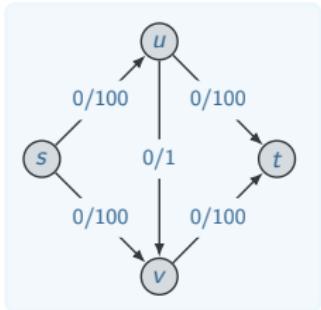
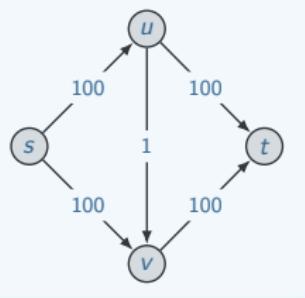
Bad Augmenting Paths



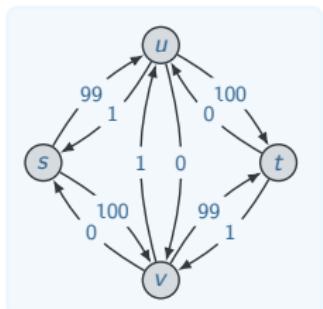
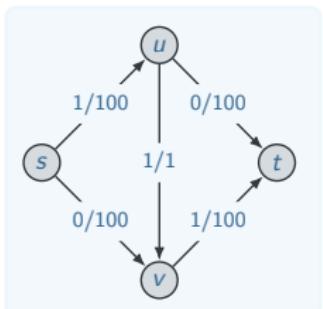
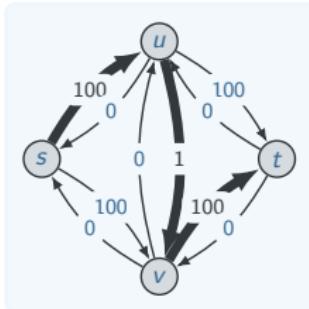
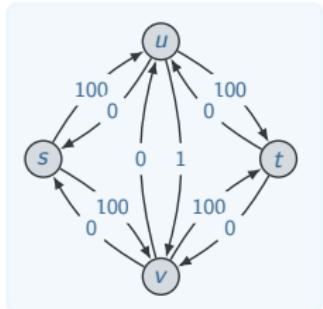
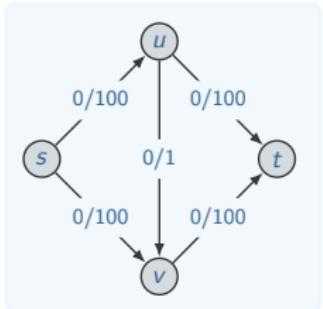
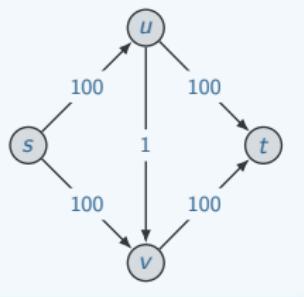
Bad Augmenting Paths



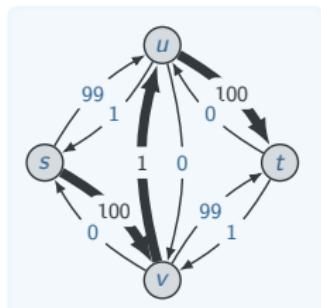
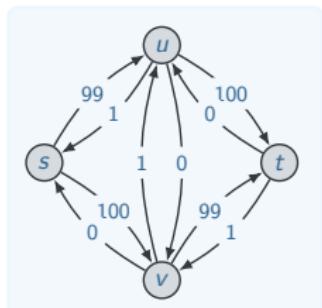
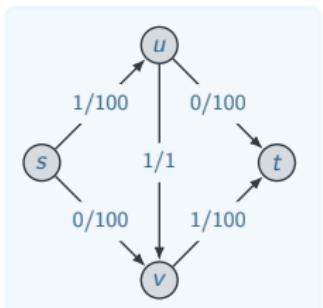
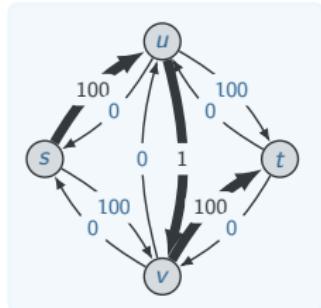
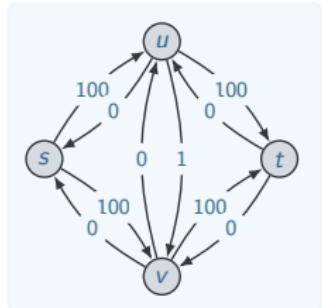
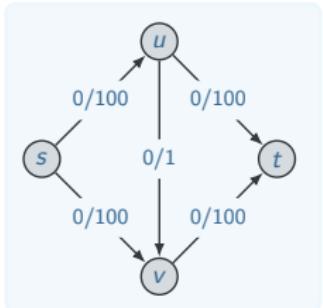
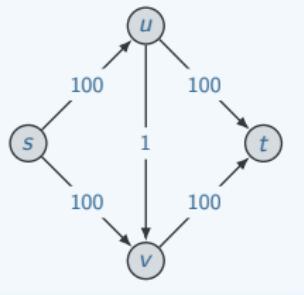
Bad Augmenting Paths



Bad Augmenting Paths



Bad Augmenting Paths



Improving Ford-Fulkerson Algorithm

- ▶ Bad case for Ford-Fulkerson algorithm is when the bottleneck edge is the augmenting path has a low capacity.
- ▶ Idea: decrease number of iterations by picking $s-t$ path with bottleneck edge of largest capacity.

Improving Ford-Fulkerson Algorithm

- ▶ Bad case for Ford-Fulkerson algorithm is when the bottleneck edge is the augmenting path has a low capacity.
- ▶ Idea: decrease number of iterations by picking $s-t$ path with bottleneck edge of largest capacity. Computing this path can slow down each iteration considerably.

Other Maximum Flow Algorithms

- Desire a **strongly polynomial** algorithm: running time depends only on the size of the graph and is *independent* of the numerical values of the capacities (as long as numerical operations).

Other Maximum Flow Algorithms

- ▶ Desire a **strongly polynomial** algorithm: running time depends only on the size of the graph and is *independent* of the numerical values of the capacities (as long as numerical operations).
- ▶ **Edmonds-Karp, Dinitz**: choose augmenting path to be the shortest path in G_f (use breadth-first search).

Questions?

Network Flow
– Scaling Max-Flow Algorithm –

Teoria dos Grafos e Computabilidade

— Exercises —

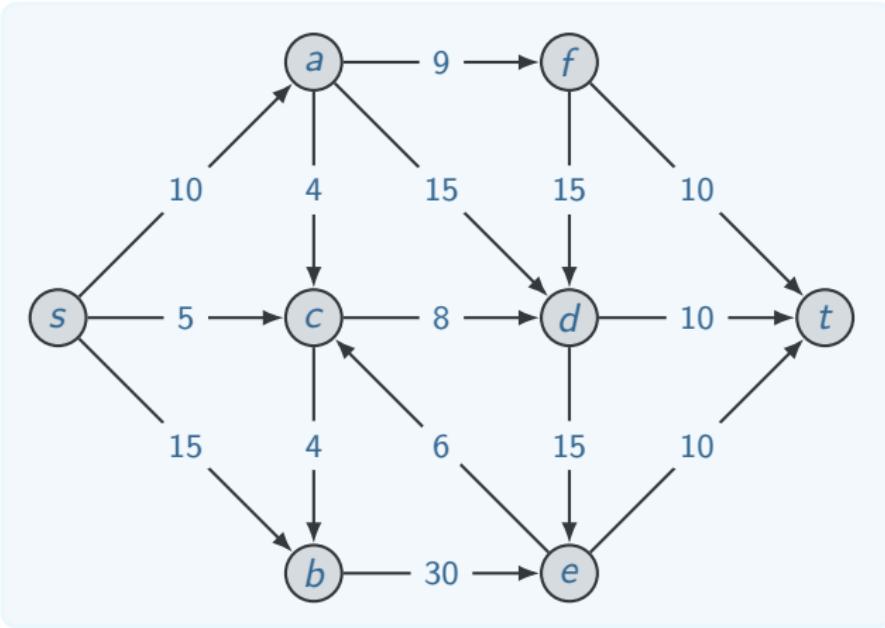
Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

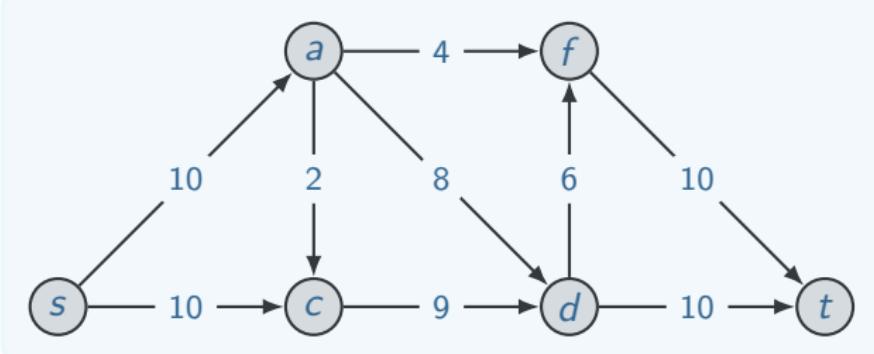
Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Compute the maximum flow



Compute the maximum flow

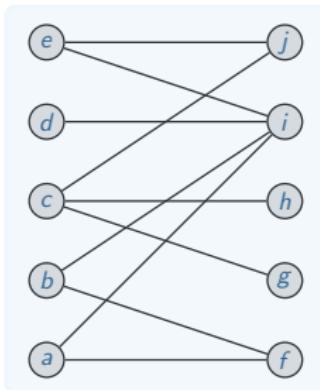


Bipartite graph matching

BIPARTITE GRAPH MATCHING

INSTANCE Let $G = (L \cup R, E)$ be an undirected graph. $M \subseteq E$ is a matching if each node appears in, at most, one edge in M .

SOLUTION Find a max cardinality matching.

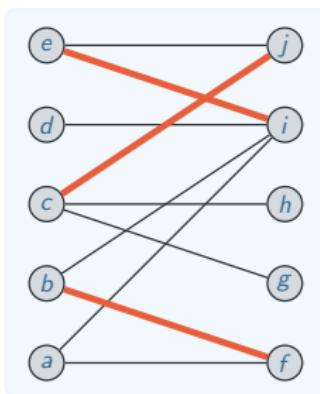


Bipartite graph matching

BIPARTITE GRAPH MATCHING

INSTANCE Let $G = (L \cup R, E)$ be an undirected graph. $M \subseteq E$ is a matching if each node appears in, at most, one edge in M .

SOLUTION Find a max cardinality matching.

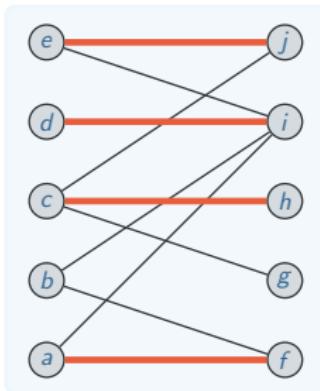


Bipartite graph matching

BIPARTITE GRAPH MATCHING

INSTANCE Let $G = (L \cup R, E)$ be an undirected graph. $M \subseteq E$ is a matching if each node appears in, at most, one edge in M .

SOLUTION Find a max cardinality matching.



Edge Disjoint Paths

DISJOINT PATH PROBLEM

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

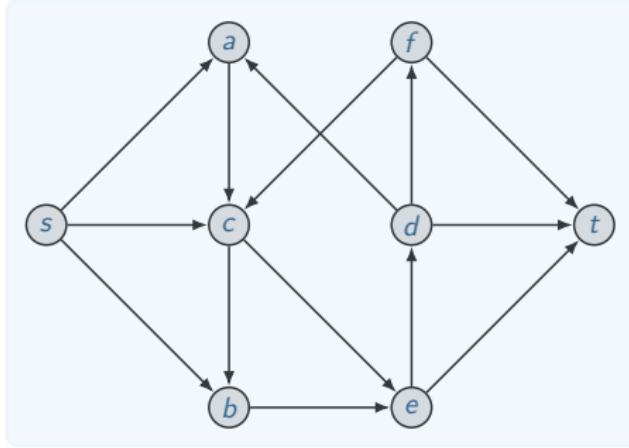
SOLUTION Find a **max number** of edge-disjoint $s-t$ paths.

Edge Disjoint Paths

DISJOINT PATH PROBLEM

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a **max number** of edge-disjoint $s-t$ paths.

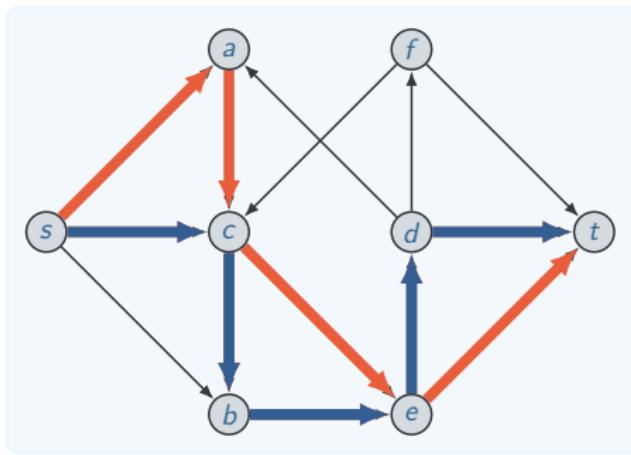


Edge Disjoint Paths

DISJOINT PATH PROBLEM

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a **max number** of edge-disjoint $s-t$ paths.



Network Connectivity

NETWORK CONNECTIVITY

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

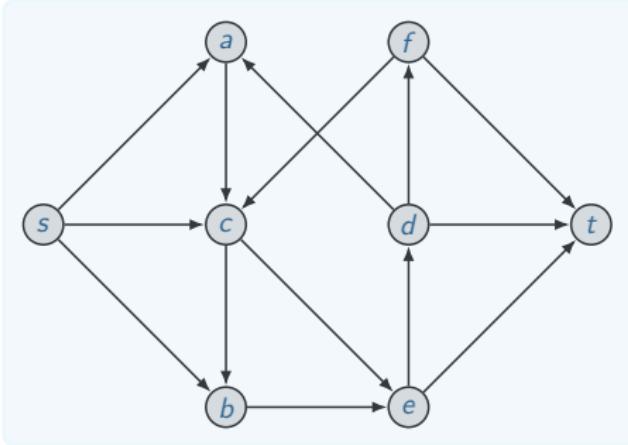
SOLUTION Find a **min number** of edges whose removal disconnects t from s

Network Connectivity

NETWORK CONNECTIVITY

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a **min number** of edges whose removal disconnects t from s

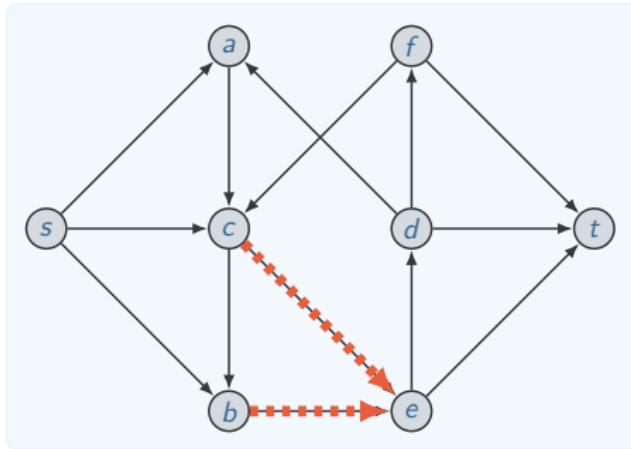


Network Connectivity

NETWORK CONNECTIVITY

INSTANCE Let $G = (G, E)$ be a directed graph and two vertices s and t

SOLUTION Find a **min number** of edges whose removal disconnects t from s





Teoria dos Grafos e Computabilidade

— Planar graphs —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Some concepts —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Bipartite graphs

If it is possible to partition the vertex set, V , into two disjoint sets, V_1 and V_2 , such that there are no edges between any two vertices in the same set, then the graph is Bipartite.

Bipartite graphs

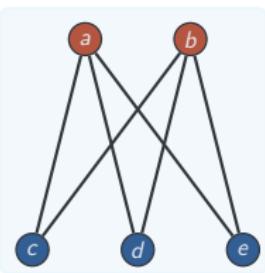
If it is possible to partition the vertex set, V , into two disjoint sets, V_1 and V_2 , such that there are no edges between any two vertices in the same set, then the graph is Bipartite.

When the bipartite graph is such that every vertex in V_1 is connected to every vertex in V_2 (and vice versa) the graph is called Complete Bipartite Graph. If $|V_1| = m$, and $|V_2| = n$, we denote it $K_{m,n}$.

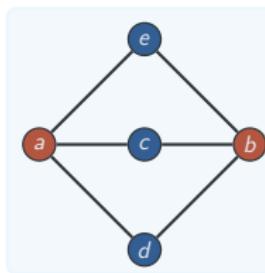
Bipartite graphs

If it is possible to partition the vertex set, V , into two disjoint sets, V_1 and V_2 , such that there are no edges between any two vertices in the same set, then the graph is Bipartite.

When the bipartite graph is such that every vertex in V_1 is connected to every vertex in V_2 (and vice versa) the graph is called Complete Bipartite Graph. If $|V_1| = m$, and $|V_2| = n$, we denote it $K_{m,n}$.



$K_{2,3}$

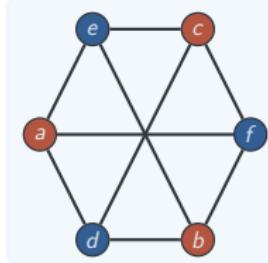
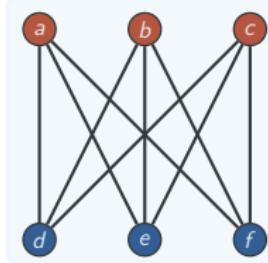
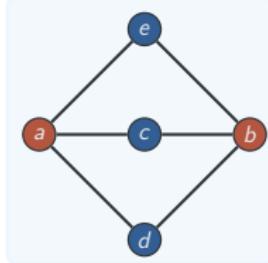
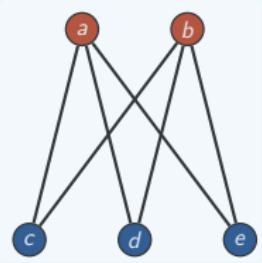


$K_{2,3}$

Bipartite graphs

If it is possible to partition the vertex set, V , into two disjoint sets, V_1 and V_2 , such that there are no edges between any two vertices in the same set, then the graph is Bipartite.

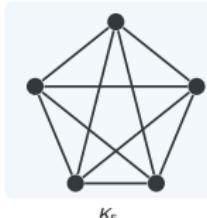
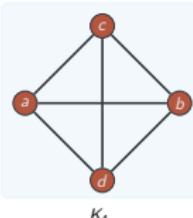
When the bipartite graph is such that every vertex in V_1 is connected to every vertex in V_2 (and vice versa) the graph is called Complete Bipartite Graph. If $|V_1| = m$, and $|V_2| = n$, we denote it $K_{m,n}$.



Some named graphs

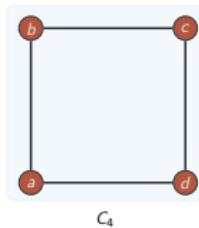
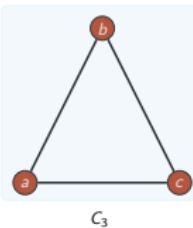
K_n

Complete graph of n vertices



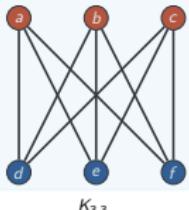
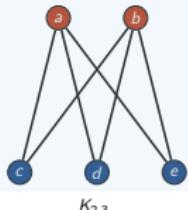
C_n

The cycle with n vertices



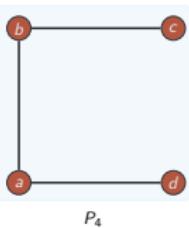
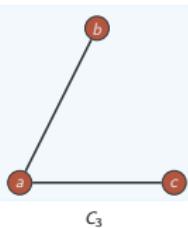
$K_{m,n}$

Complete bipartite graph of m and n vertices



P_n

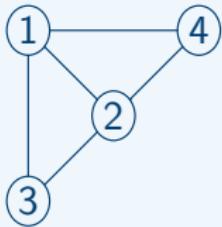
The path with n vertices



Sub-grafo

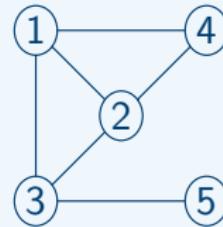
- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de H estão em G

H



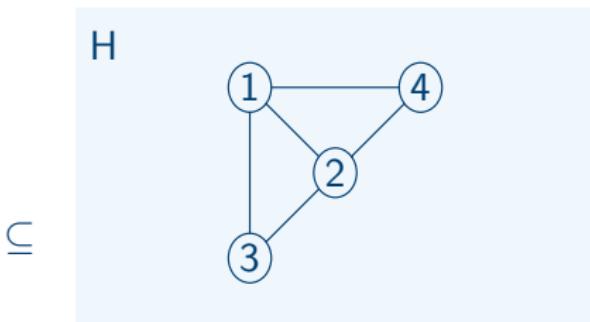
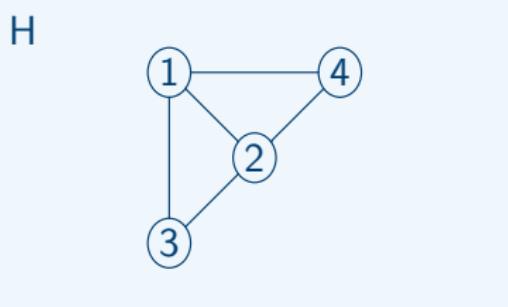
\subseteq

G



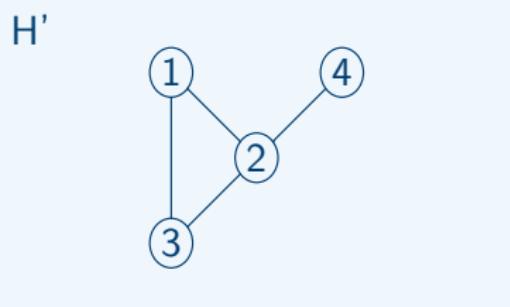
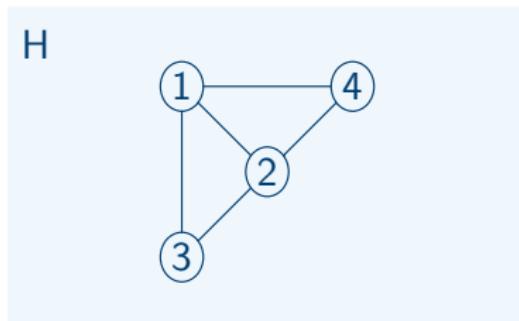
Sub-grafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio



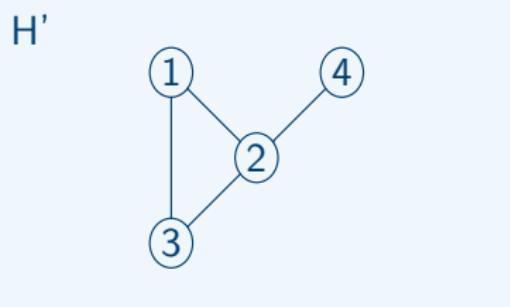
Sub-grafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G

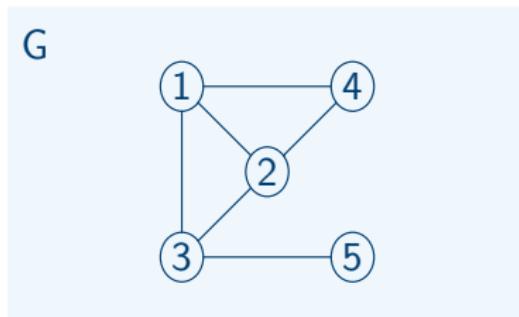
 \subseteq 

Sub-grafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G



\subseteq



Sub-grafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G
 - um vértice simples de G é um subgrafo de G



Sub-grafo

- Um grafo H é dito ser um subgrafo de um grafo G ($H \subseteq G$) se **todos os vértices** e todas as **arestas** de g estão em G
 - todo grafo é subgrafo de si próprio
 - o subgrafo de um subgrafo de G é subgrafo de G
 - um vértice simples de G é um subgrafo de G
 - uma aresta simples de G (juntamente com suas extremidades) é subgrafo de G



Questions?

Planar graphs
– Some concepts –



Teoria dos Grafos e Computabilidade

— Planar graphs —

Silvio Jamil F. Guimarães

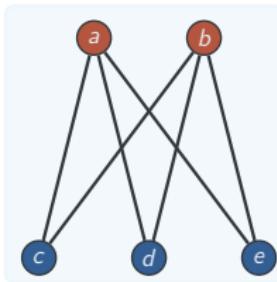
Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

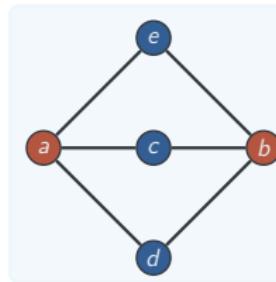
Pontifical Catholic University of Minas Gerais – PUC Minas

Planar graphs

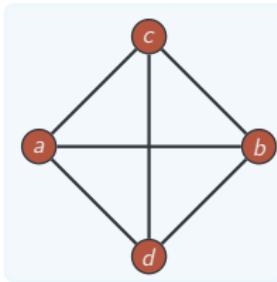
If you can sketch a graph so that none of its edges cross, then it is a planar graph.



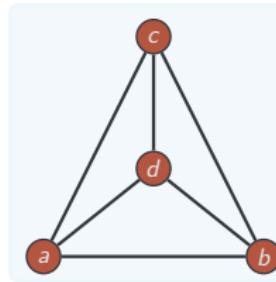
$K_{2,3}$



$K_{2,3}$



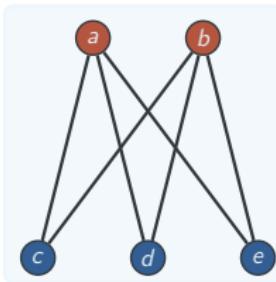
K_4



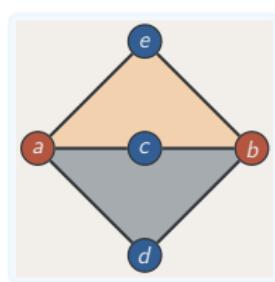
K_4

Planar graphs

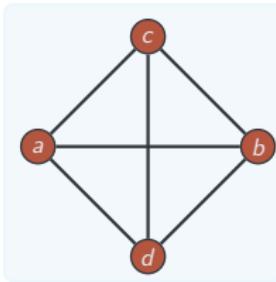
When a planar graph is drawn without edges crossing, the edges and vertices of the graph divide the plane into regions. Each region is called a face.



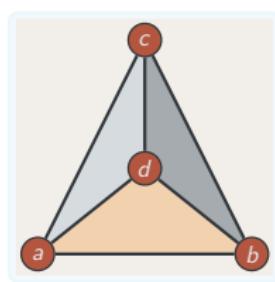
$K_{2,3}$



$K_{2,3} - 3$ faces



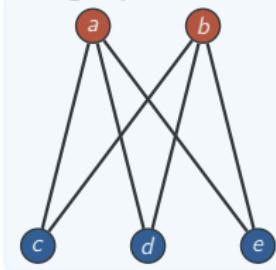
K_4



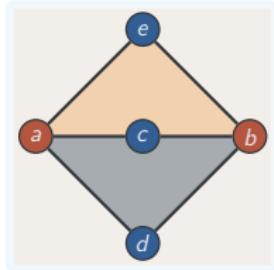
$K_4 - 4$ faces

Planar graphs

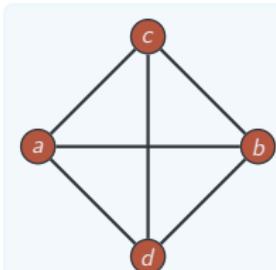
When a planar graph is drawn without edges crossing, the edges and vertices of the graph divide the plane into regions. Each region is called a face. The number of faces does not change no matter how you draw the graph, as long as no edges cross.



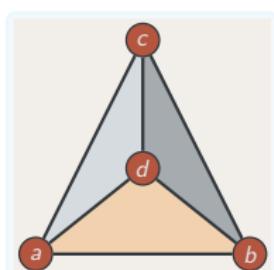
$K_{2,3}$



$K_{2,3} - 3$ faces



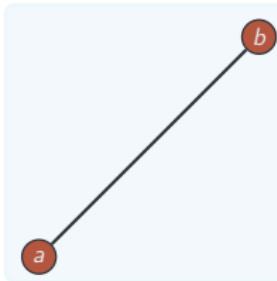
K_4



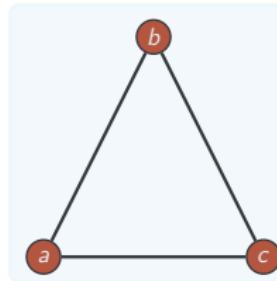
$K_4 - 4$ faces

Planar graphs

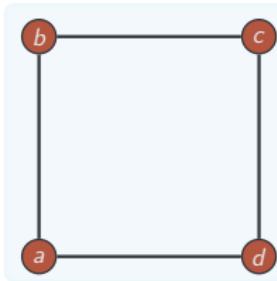
Count the number of edges, faces and vertices in the cycle graphs C_3 , C_4 and C_5 . What about C_k ?



C_2



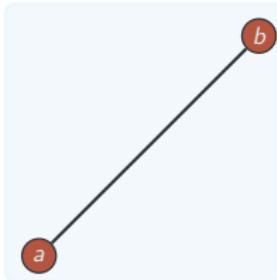
C_3



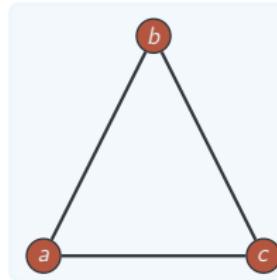
C_4

Planar graphs

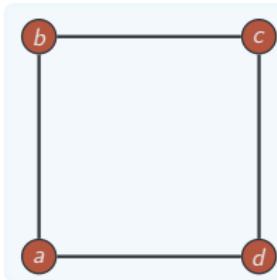
Count the number of edges, faces and vertices in the cycle graphs C_3 , C_4 and C_5 . What about C_k ? And what about P_k ?



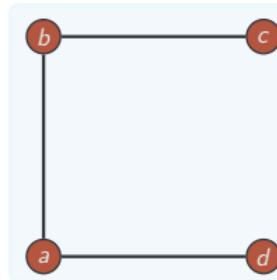
C_2



C_3



C_4



P_4

Planar graphs and Euler's formula

Let a list of some planar graphs, and count their vertices, edges, and faces, for example, K_3 , K_4 and C_5

Planar graphs and Euler's formula

Let a list of some planar graphs, and count their vertices, edges, and faces, for example, K_3 , K_4 and C_5 Is there a pattern?

Planar graphs and Euler's formula

Let a list of some planar graphs, and count their vertices, edges, and faces, for example, K_3 , K_4 and C_5 . Is there a pattern?

For any (connected) planar graph with v vertices, e edges and f faces, we have

$$v - e + f = 2$$

Planar graphs and Euler's formula

Let a list of some planar graphs, and count their vertices, edges, and faces, for example, K_3 , K_4 and C_5 . Is there a pattern?

For any (connected) planar graph with v vertices, e edges and f faces, we have

$$v - e + f = 2$$

Outline of the proof:

Planar graphs and Euler's formula

Let a list of some planar graphs, and count their vertices, edges, and faces, for example, K_3 , K_4 and C_5 . Is there a pattern?

For any (connected) planar graph with v vertices, e edges and f faces, we have

$$v - e + f = 2$$

Outline of the proof:

Consider the graph with a single vertex and no edges. So $v=1$, $e=0$ and $f=1$.

We can construct any other planar connected graph from this as follows:

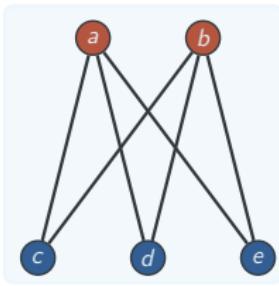
(i) - Let a K_3 be a complete graph with 3 vertices. Add one vertex and one edge. This will increase the number of vertices and edges by 1, and the number of faces will stay the same. So, $v - e + f$ is the same.

(ii) - Let the graph of (i). Add one edge but no new vertex. So, the number of vertices is unchanged, but the number of edges and faces will increase by 1. So, $v - e + f$ is the same.

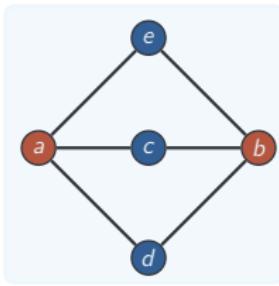
So, by induction, $v - e + f = 2$

Planar graphs and Euler's formula

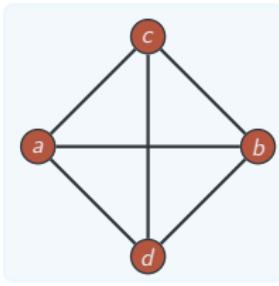
According to Fáry theorem (1947), every (simple) planar graph admits a straight line planar embedding (no edge crossings).



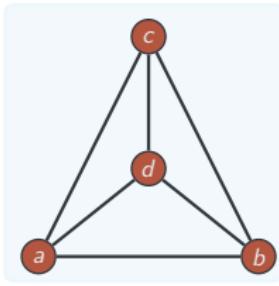
$K_{2,3}$



$K_{2,3}$



K_4



K_4

Questions?

Planar graphs
– Planar graphs –



Teoria dos Grafos e Computabilidade

— Non-planar graphs —

Silvio Jamil F. Guimarães

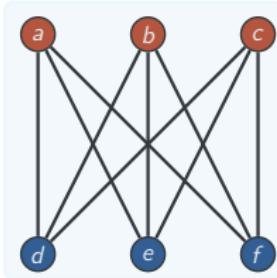
Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

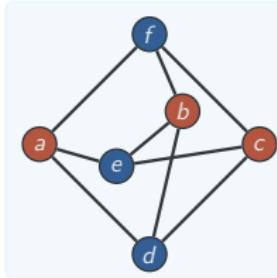
Pontifical Catholic University of Minas Gerais – PUC Minas

Non-planar graphs

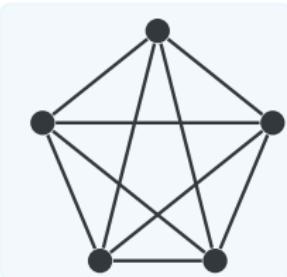
Most graphs **do not** have a planar representation. For example, the following two graphs **cannot** be drawn so no edges cross: K_5 and $K_{3,3}$.



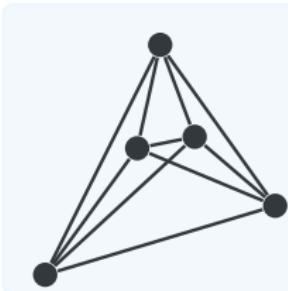
$K_{3,3}$



$K_{3,3}$



K_5

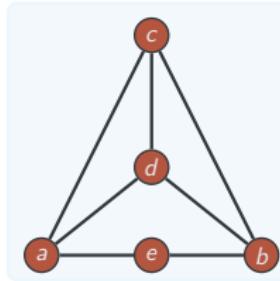
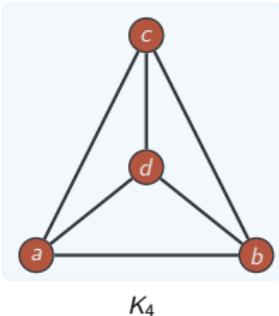


K_5

Homeomorphic graphs

Recall that a graph G' is a subgraph of G if it can be obtained by deleting some vertices and/or edges of G .

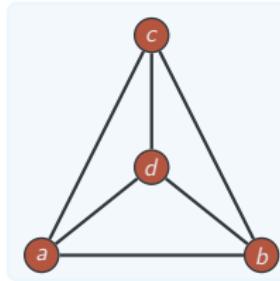
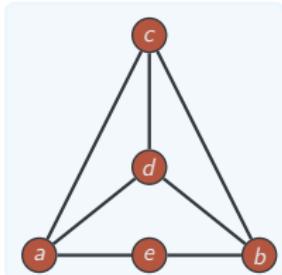
- ▶ A subdivision of an edge is obtained by adding a new vertex of degree 2 to the middle of the edge.
- ▶ A subdivision of a graph is obtained by subdividing one or more of its edges .



Homeomorphic graphs

Recall that a graph G' is a subgraph of G if it can be obtained by deleting some vertices and/or edges of G .

- ▶ Smoothing of the pair of edges $\{a, b\}$ and $\{b, c\}$, in which the degree of vertex b is equal to 2, means to remove these two edges, and add $\{a, c\}$.

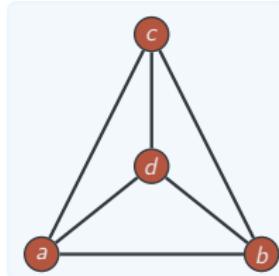
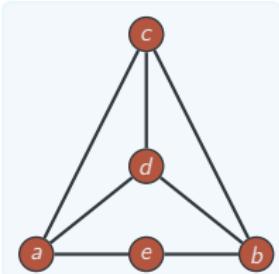


K_4

Homeomorphic graphs

Recall that a graph G' is a subgraph of G if it can be obtained by deleting some vertices and/or edges of G .

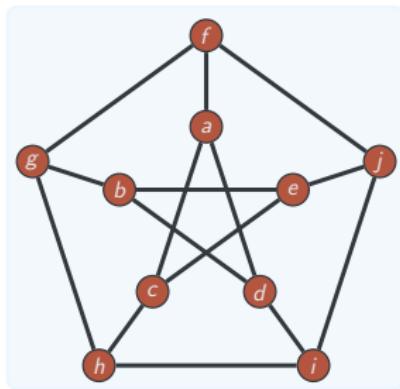
- The graphs G_1 and G_2 are homeomorphic if there is some subdivision of G_1 that is isomorphic to some subdivision of G_2 .



K_4

Kuratowski's theorem

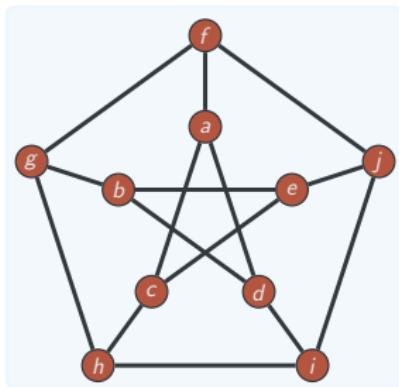
The Kuratowski's theorem says that a graph is planar if and only if it does not contain a subgraph that is homeomorphic to K_5 or $K_{3,3}$.



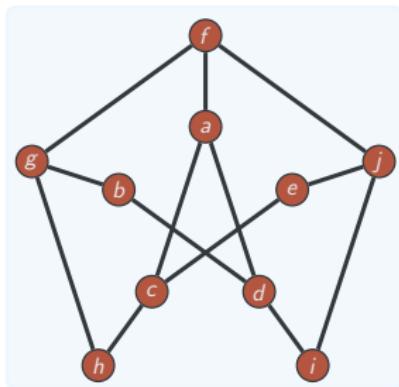
Petersen Graph

Kuratowski's theorem

The Kuratowski's theorem says that a graph is planar if and only if it does not contain a subgraph that is homeomorphic to K_5 or $K_{3,3}$.



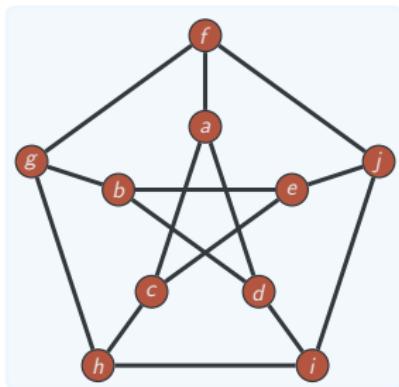
Petersen Graph



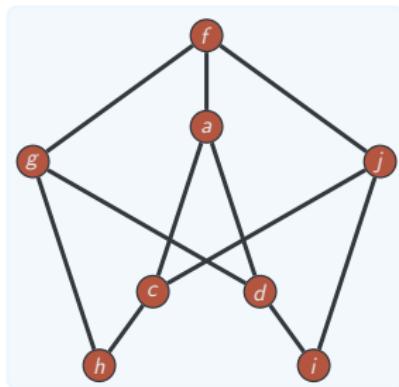
Subgraph of Petersen Graph

Kuratowski's theorem

The Kuratowski's theorem says that a graph is planar if and only if it does not contain a subgraph that is homeomorphic to K_5 or $K_{3,3}$.



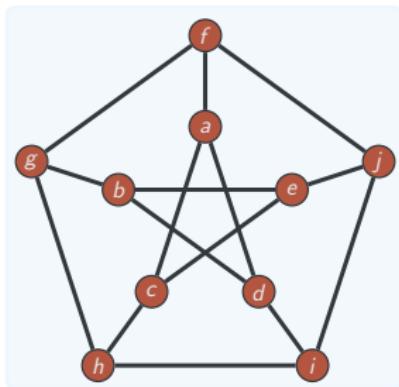
Petersen Graph



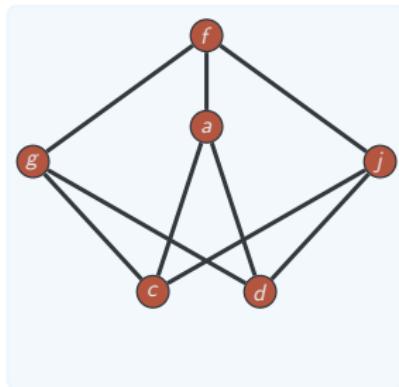
Petersen Graph – smoothing out

Kuratowski's theorem

The Kuratowski's theorem says that a graph is planar if and only if it does not contain a subgraph that is homeomorphic to K_5 or $K_{3,3}$.



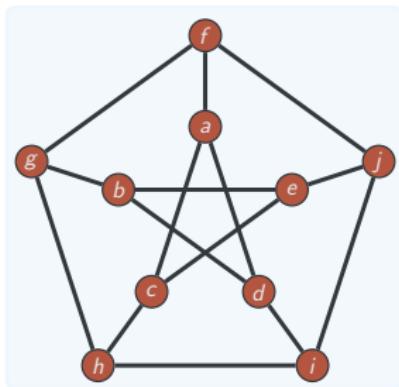
Petersen Graph



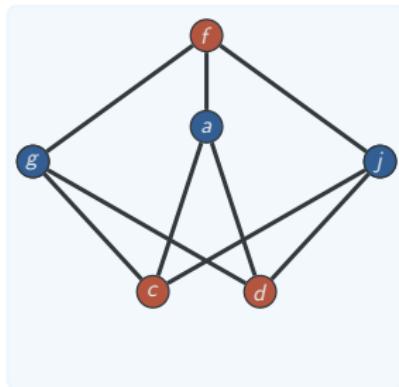
Petersen Graph – smoothing out

Kuratowski's theorem

The Kuratowski's theorem says that a graph is planar if and only if it does not contain a subgraph that is homeomorphic to K_5 or $K_{3,3}$.



Petersen Graph

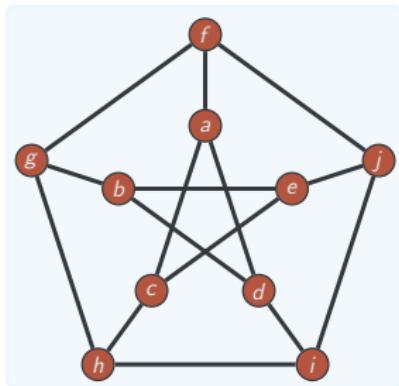


$K_{3,3}$

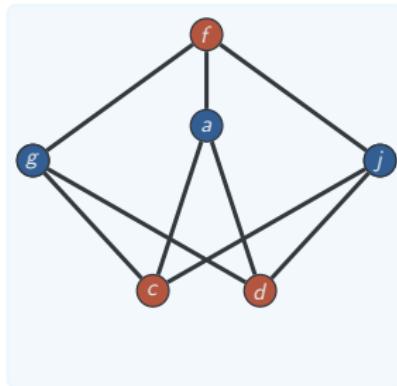
Kuratowski's theorem

The Kuratowski's theorem says that a graph is planar if and only if it does not contain a subgraph that is homeomorphic to K_5 or $K_{3,3}$.

- What this really means is that every non-planar graph has some smoothing that contains a copy of K_5 or $K_{3,3}$ somewhere inside it.



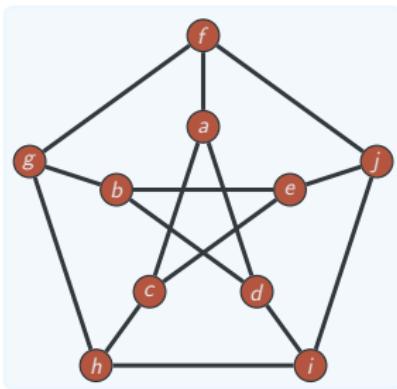
Petersen Graph



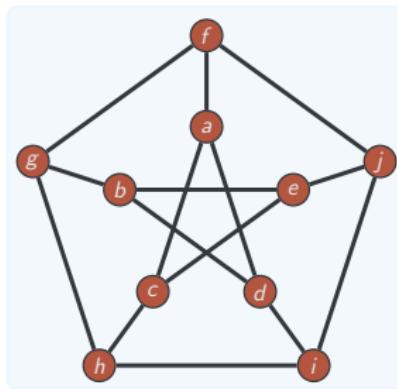
$K_{3,3}$

Wagner's theorem

The Wagner's theorem says that a graph has planar embedding, if, and only if, it contains no minor isomorphic to K_5 or $K_{3,3}$.



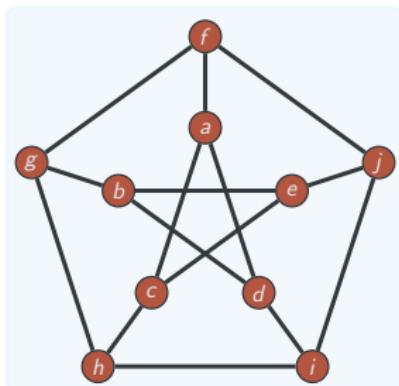
Petersen Graph



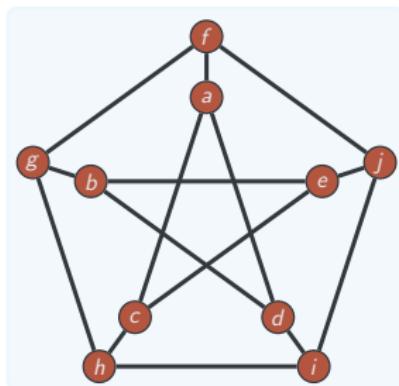
Petersen Graph

Wagner's theorem

The Wagner's theorem says that a graph has planar embedding, if, and only if, it contains no minor isomorphic to K_5 or $K_{3,3}$.



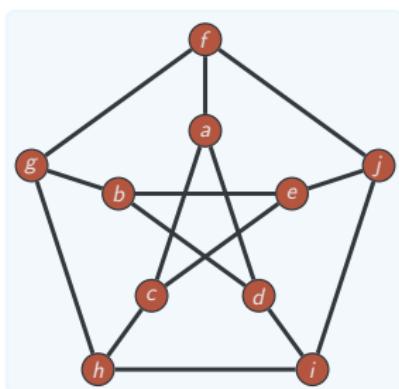
Petersen Graph



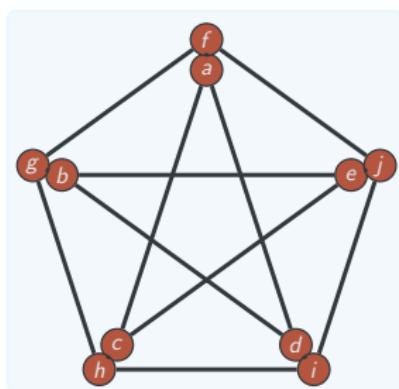
Petersen Graph

Wagner's theorem

The Wagner's theorem says that a graph has planar embedding, if, and only if, it contains no minor isomorphic to K_5 or $K_{3,3}$.



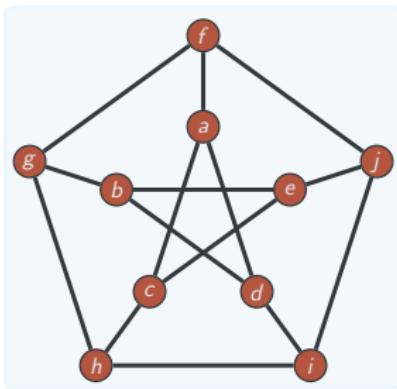
Petersen Graph



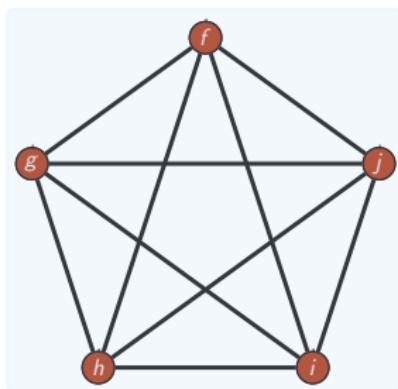
Petersen Graph

Wagner's theorem

The Wagner's theorem says that a graph has planar embedding, if, and only if, it contains no minor isomorphic to K_5 or $K_{3,3}$. A contraction of G is a graph obtained from G by repeated edge contractions. A minor of G is any subgraph of a contraction of G .



Petersen Graph

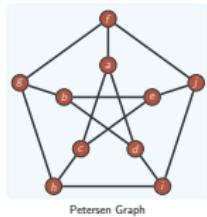


Petersen Graph – K_5

Wagner's theorem

Let $G = (V, E)$ be a graph and let $\{x, y\} \in E$. The graph G/xy , called the **edge xy -contraction of G** , consists of

- ▶ the vertex set $V' = V \setminus \{y\}$ and the edge set E' consisting of pairs $\{w, v\} \in E$ such that $y \notin \{w, v\}$

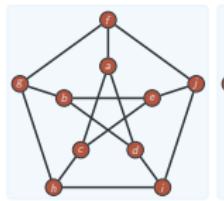


Petersen Graph

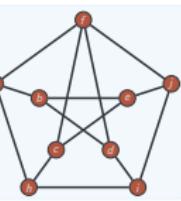
Wagner's theorem

Let $G = (V, E)$ be a graph and let $\{x, y\} \in E$. The graph G/xy , called the edge xy -contraction of G , consists of

- ▶ the vertex set $V' = V \setminus \{y\}$ and the edge set E' consisting of pairs $\{w, v\} \in E$ such that $y \notin \{w, v\}$
- ▶ plus all pairs $\{w, x\}$, $w \neq x$, with $\{w, y\} \in E$. No loops or multiple edges are allowed.



Petersen Graph

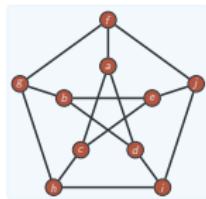


Petersen Graph – contraction

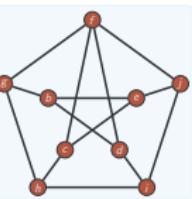
Wagner's theorem

Let $G = (V, E)$ be a graph and let $\{x, y\} \in E$. The graph G/xy , called the edge xy -contraction of G , consists of

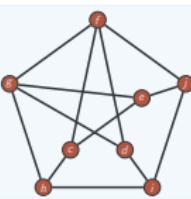
- ▶ the vertex set $V' = V \setminus \{y\}$ and the edge set E' consisting of pairs $\{w, v\} \in E$ such that $y \notin \{w, v\}$
- ▶ plus all pairs $\{w, x\}$, $w \neq x$, with $\{w, y\} \in E$. No loops or multiple edges are allowed.



Petersen Graph



Petersen Graph – contraction

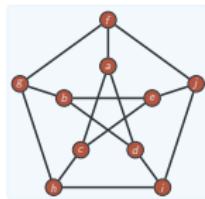


Petersen Graph – contraction

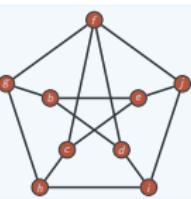
Wagner's theorem

Let $G = (V, E)$ be a graph and let $\{x, y\} \in E$. The graph G/xy , called the **edge xy -contraction of G** , consists of

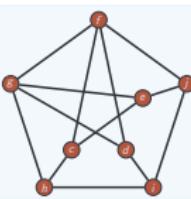
- ▶ the vertex set $V' = V \setminus \{y\}$ and the edge set E' consisting of pairs $\{w, v\} \in E$ such that $y \notin \{w, v\}$
- ▶ plus all pairs $\{w, x\}$, $w \neq x$, with $\{w, y\} \in E$. No loops or multiple edges are allowed.



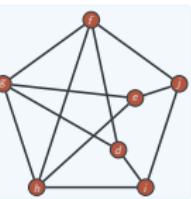
Petersen Graph



Petersen Graph – contraction



Petersen Graph – contraction

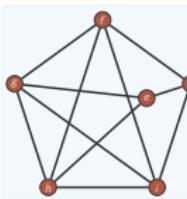
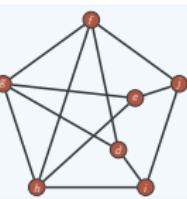
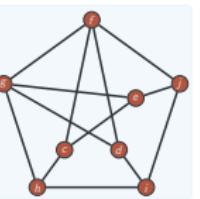
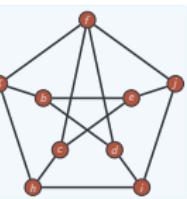
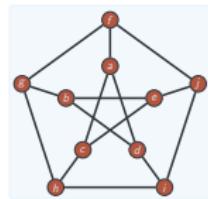


Petersen Graph – contraction

Wagner's theorem

Let $G = (V, E)$ be a graph and let $\{x, y\} \in E$. The graph G/xy , called the **edge xy -contraction** of G , consists of

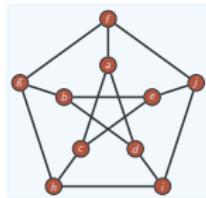
- ▶ the vertex set $V' = V \setminus \{y\}$ and the edge set E' consisting of pairs $\{w, v\} \in E$ such that $y \notin \{w, v\}$
- ▶ plus all pairs $\{w, x\}$, $w \neq x$, with $\{w, y\} \in E$. No loops or multiple edges are allowed.



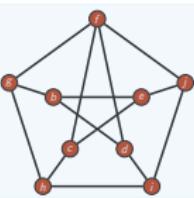
Wagner's theorem

Let $G = (V, E)$ be a graph and let $\{x, y\} \in E$. The graph G/xy , called the **edge xy -contraction** of G , consists of

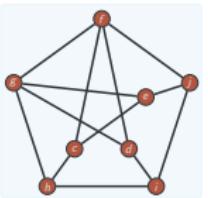
- ▶ the vertex set $V' = V \setminus \{y\}$ and the edge set E' consisting of pairs $\{w, v\} \in E$ such that $y \notin \{w, v\}$
- ▶ plus all pairs $\{w, x\}$, $w \neq x$, with $\{w, y\} \in E$. No loops or multiple edges are allowed.



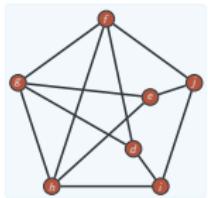
Petersen Graph



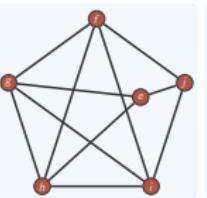
Petersen Graph – contraction



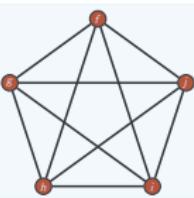
Petersen Graph – contraction



Petersen Graph – contraction



Petersen Graph – contraction



Petersen Graph – K_5

Questions?

Planar graphs

– Non-planar graphs –



Teoria dos Grafos e Computabilidade

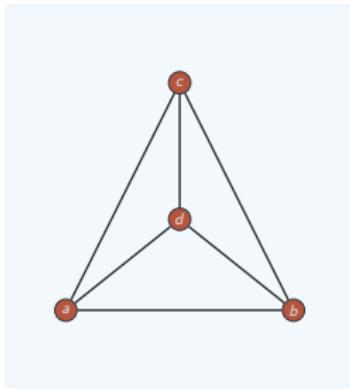
— Geometric duality —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Image and Multimedia Data Science Laboratory – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Geometric duality

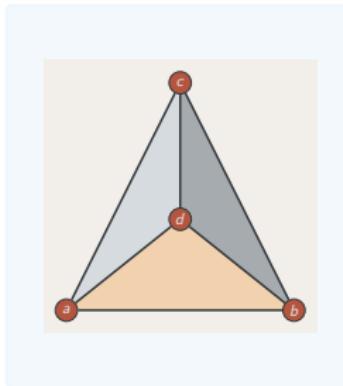
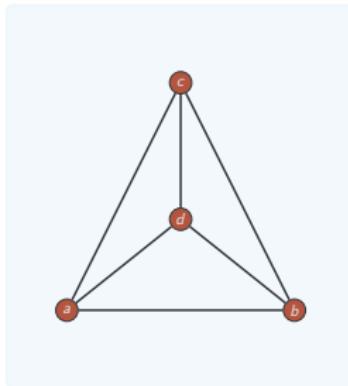
Let $G = (V, E)$. A geometric dual $G^* = (V^*, E^*)$ of a planar representation of G – no crossing edges – is computed as follows:



Geometric duality

Let $G = (V, E)$. A geometric dual $G^* = (V^*, E^*)$ of a planar representation of G – no crossing edges – is computed as follows:

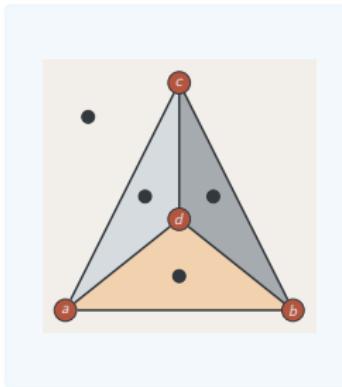
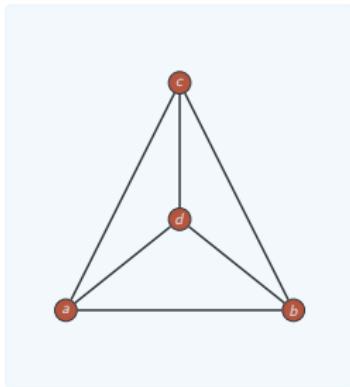
- ▶ For each face of G , pick one point v^* inside the face. These are the set of vertices V^* of G^* .



Geometric duality

Let $G = (V, E)$. A geometric dual $G^* = (V^*, E^*)$ of a planar representation of G – no crossing edges – is computed as follows:

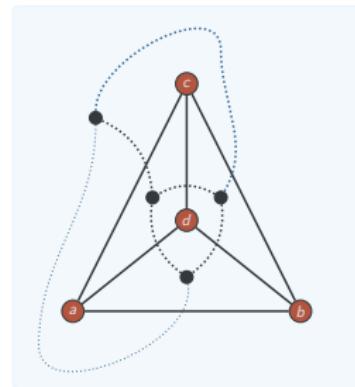
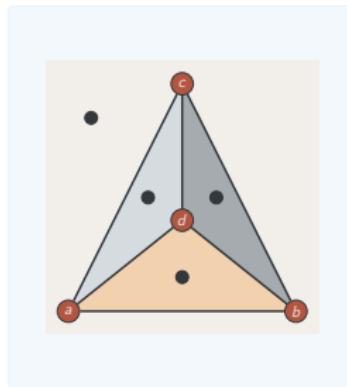
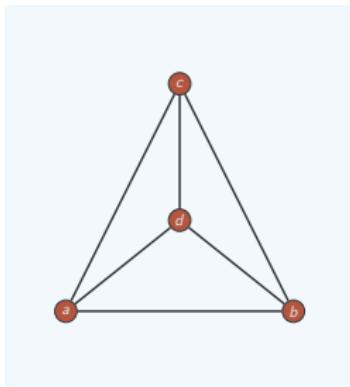
- ▶ For each face of G , pick one point v^* inside the face. These are the set of vertices V^* of G^* .



Geometric duality

Let $G = (V, E)$. A geometric dual $G^* = (V^*, E^*)$ of a planar representation of G – no crossing edges – is computed as follows:

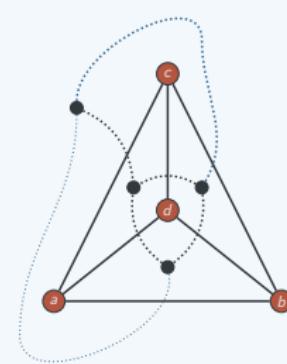
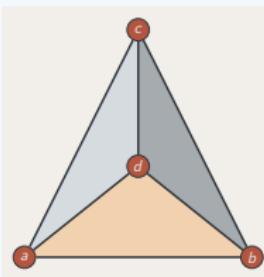
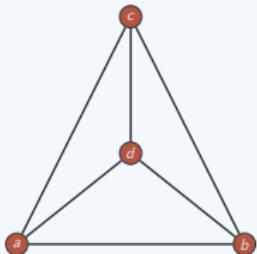
- ▶ For each face of G , pick one point v^* inside the face. These are the the set of vertices V^* of G^* .
- ▶ Any edge $e \in E$ of G that divides two faces of G and hence two vertices of G^* , so let e^* be the edge of G^* .



Geometric duality

Let $G = (V, E)$ be a planar connected graph.

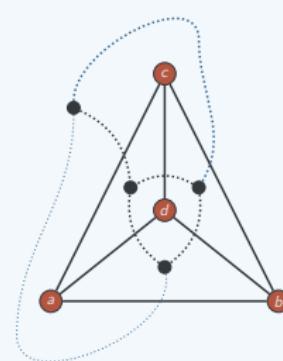
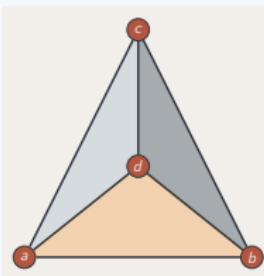
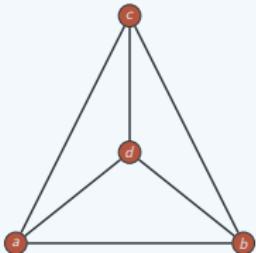
1. Is the number of edges which encloses the region a equal to the degree of the vertex correspondent to the region a ?



Geometric duality

Let $G = (V, E)$ be a planar connected graph.

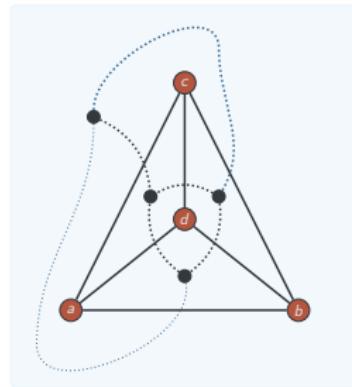
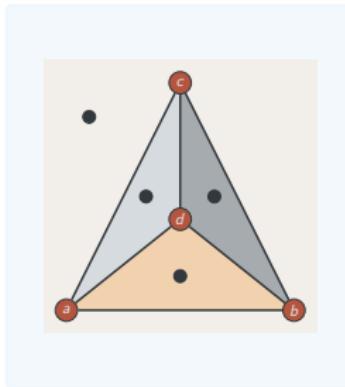
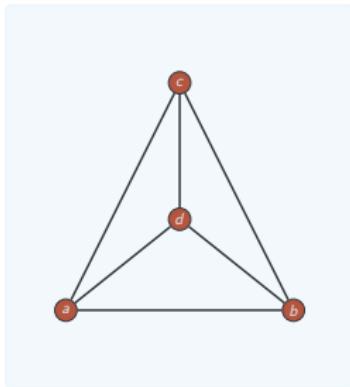
1. Is the number of edges which encloses the region a equal to the degree of the vertex correspondent to the region a ? Yes



Geometric duality

Let $G = (V, E)$ be a planar connected graph.

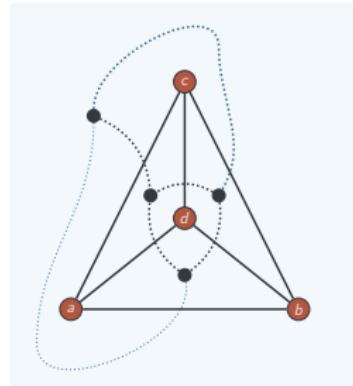
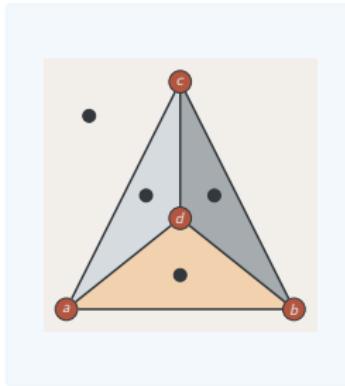
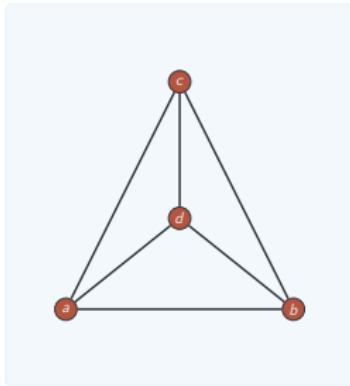
1. Is the number of edges which encloses the region a equal to the degree of the vertex correspondent to the region a ? Yes
2. Is the dual graph a planar one?



Geometric duality

Let $G = (V, E)$ be a planar connected graph.

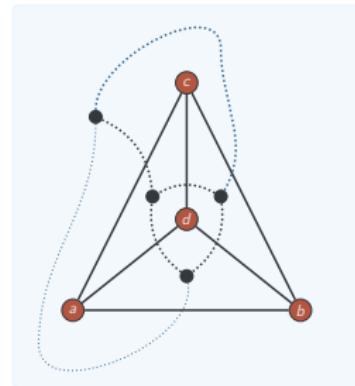
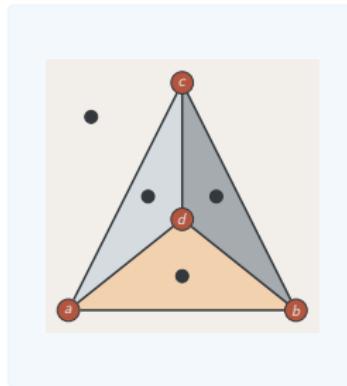
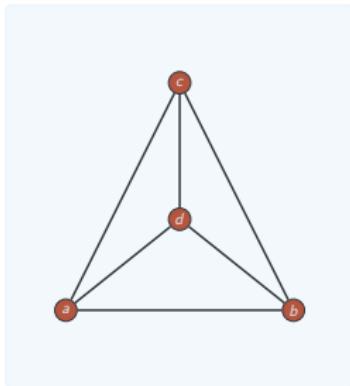
1. Is the number of edges which encloses the region a equal to the degree of the vertex correspondent to the region a ? Yes
2. Is the dual graph a planar one? Yes



Geometric duality

Let $G = (V, E)$ be a planar connected graph.

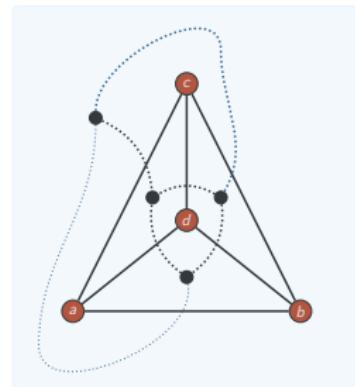
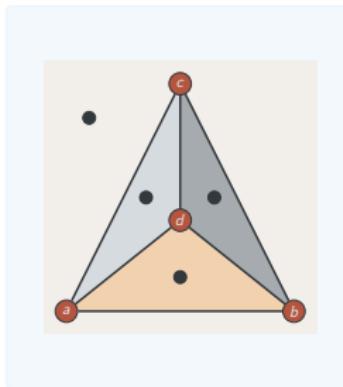
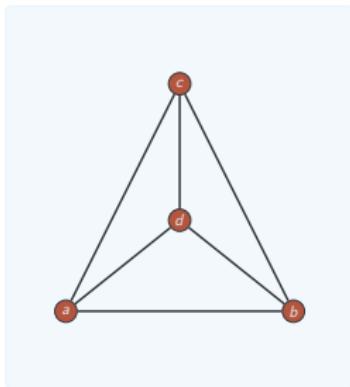
1. Is the number of edges which encloses the region a equal to the degree of the vertex correspondent to the region a ? Yes
2. Is the dual graph a planar one? Yes
3. Is the dual graph of the dual graph G equal to G ?



Geometric duality

Let $G = (V, E)$ be a planar connected graph.

1. Is the number of edges which encloses the region a equal to the degree of the vertex correspondent to the region a ? Yes
2. Is the dual graph a planar one? Yes
3. Is the dual graph of the dual graph G equal to G ?
No. They are isomorphic



Questions?

Planar graphs
– Geometric duality –

Teoria dos Grafos e Computabilidade

— Graph coloring —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

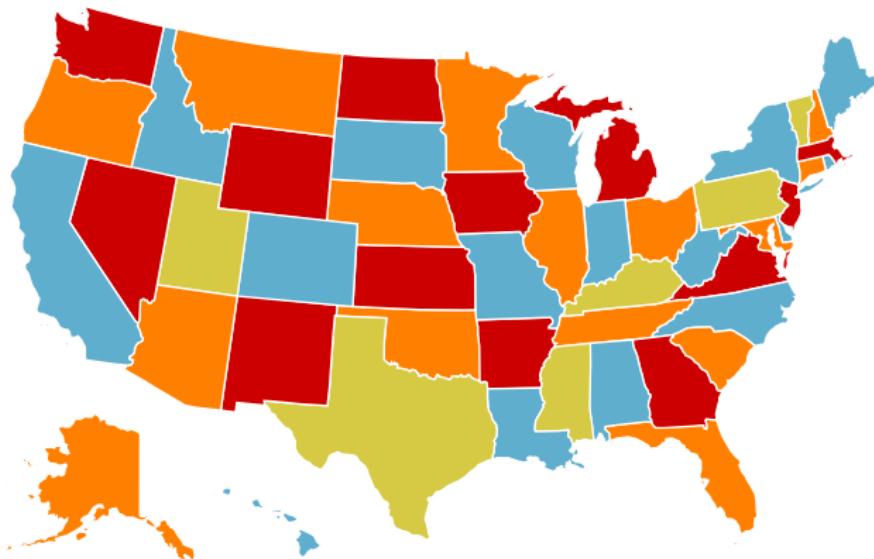
Pontifical Catholic University of Minas Gerais – PUC Minas

Graph coloring

Here is a map of the USA country. Color it so that adjacent regions are colored differently. What is the fewest colors required?

Graph coloring

Here is a map of the USA country. Color it so that adjacent regions are colored differently. What is the fewest colors required?



Graph coloring

Here is a map of the USA country. Color it so that adjacent regions are colored differently. What is the fewest colors required?

There are maps can be colored with: (i) one color; (ii) two colors; (iii) three colors; (iv) four colors.

It turns out that the is no map that needs more than 4 colors.

This is the famous Four Colour Theorem, which was originally conjectured by the British/South African mathematician and botanist, Francis Guthrie who at the time was a student at University College London

Graph coloring

Thanks to the geometric duality, a map can be seen as a graph in which:

Graph coloring

Thanks to the geometric duality, a map can be seen as a graph in which:



Graph coloring

Thanks to the geometric duality, a map can be seen as a graph in which:

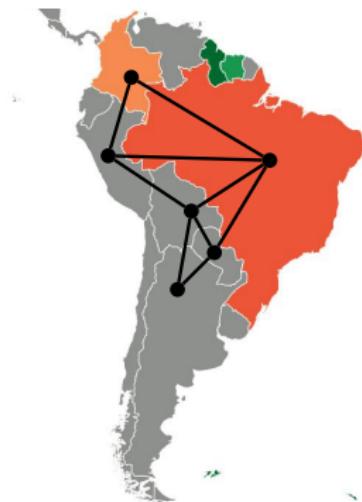
- A vertex in the graph corresponds to a region (face) in the map;



Graph coloring

Thanks to the geometric duality, a map can be seen as a graph in which:

- ▶ A vertex in the graph corresponds to a region (face) in the map;
- ▶ There is an edge between two vertices in the graph if the corresponding regions share a border.



Graph coloring

Thanks to the geometric duality, coloring regions of a map corresponds to coloring vertices of the graph.

Graph coloring

Thanks to the geometric duality, coloring regions of a map corresponds to coloring vertices of the graph.

- ▶ Vertex Coloring An assignment of colors to the vertices of a graph;

Graph coloring

Thanks to the geometric duality, coloring regions of a map corresponds to coloring vertices of the graph.

- ▶ **Vertex Coloring** An assignment of colors to the vertices of a graph;
- ▶ **Proper Coloring** If the vertex coloring has the property that adjacent vertices are colored differently.

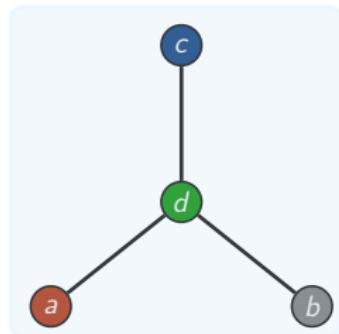
Graph coloring

Thanks to the geometric duality, coloring regions of a map corresponds to coloring vertices of the graph.

- ▶ **Vertex Coloring** An assignment of colors to the vertices of a graph;
- ▶ **Proper Coloring** If the vertex coloring has the property that adjacent vertices are colored differently.

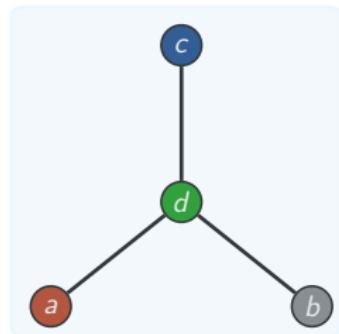
If the graph has v vertices, the clearly at most v colours are needed. However, usually, we need far fewer .

Graph coloring

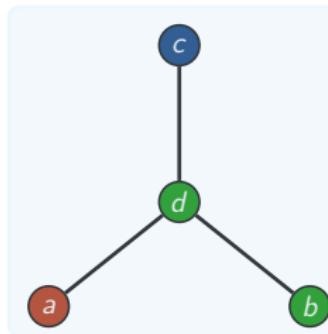


4 colors

Graph coloring

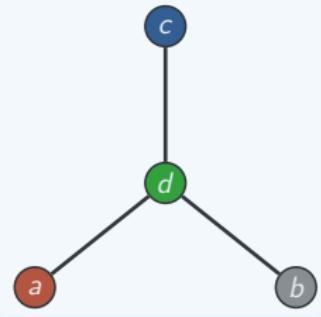


4 colors

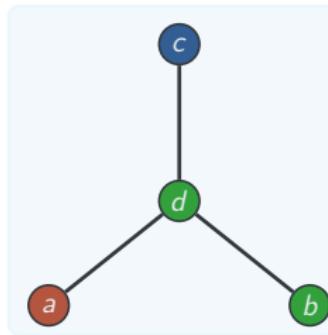


3 colors – no proper

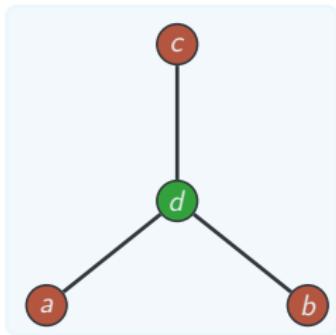
Graph coloring



4 colors

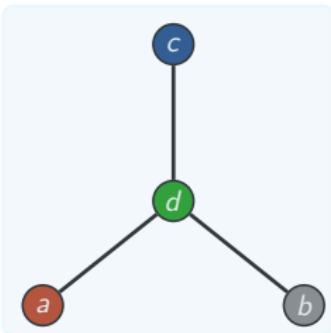


3 colors – no proper

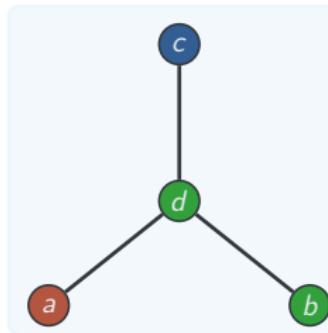


2 colors – proper and
minimal

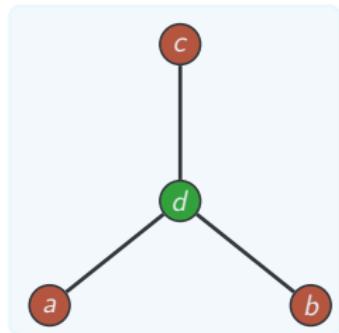
Graph coloring



4 colors



3 colors – no proper



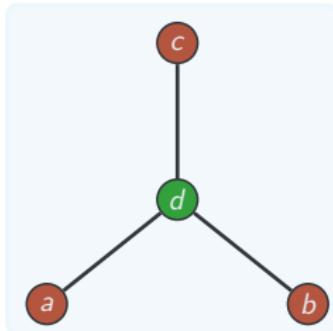
2 colors – proper and
minimal

From now, the vertex coloring will be also proper coloring .

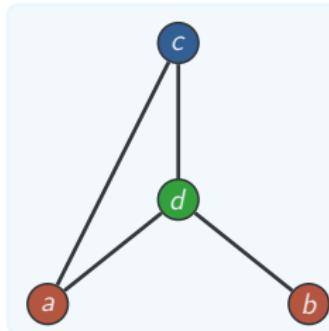
Graph coloring

The **smallest** number of colors needed to get a proper vertex coloring of a graph $G=(V,E)$ is called the **chromatic number** of the graph, written $\chi(G)$ in which $1 \leq \chi(G) \leq |V|$.

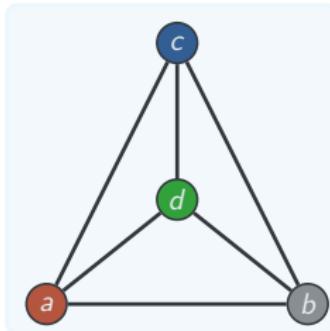
We said that a graph is **K -colorable** if K colors are sufficient to compute a vertex coloring.



$$\chi(G) = 2$$



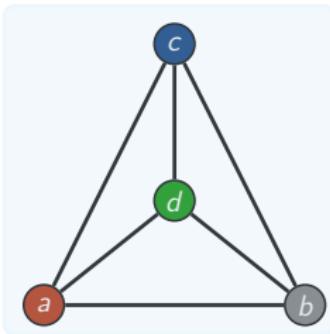
$$\chi(G) = 3$$



$$\chi(G) = 4$$

Graph coloring

If the graph $G = (V, E)$ is a complete one, then $\chi(G) = |V|$. If it is not complete we can look at cliques in the graph.

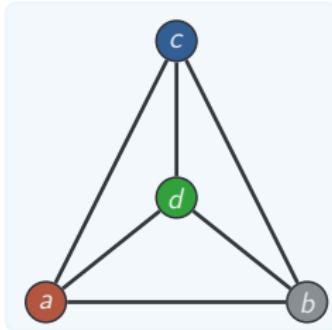


$$\chi(G) = 4$$

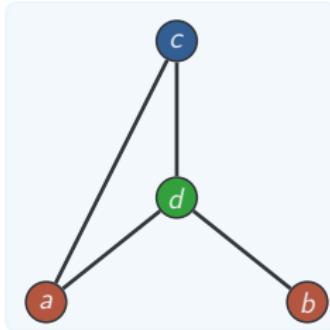
Graph coloring

If the graph $G = (V, E)$ is a complete one, then $\chi(G) = |V|$. If it is not complete we can look at cliques in the graph.

A clique is a subgraph of a graph all of whose vertices are connected to each other.



$$\chi(G) = 4$$



$$\chi(G) = 3$$

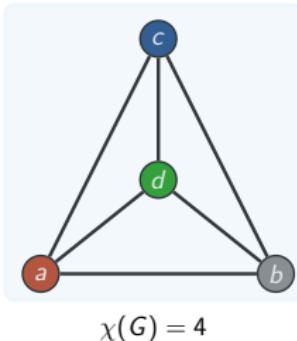
Graph coloring

The clique number of a graph, $G = (V, E)$, is the number of vertices in the largest clique in G.

Graph coloring

The clique number of a graph, $G = (V, E)$, is the number of vertices in the largest clique in G .

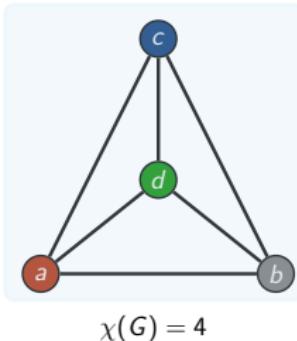
- ▶ The chromatic number of a graph G , called $\chi(G)$, is at least its clique number



Graph coloring

The clique number of a graph, $G = (V, E)$, is the number of vertices in the largest clique in G .

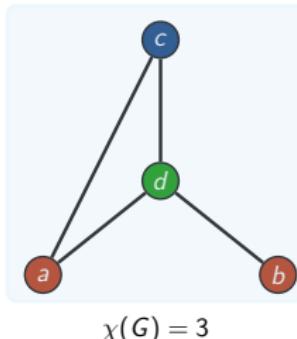
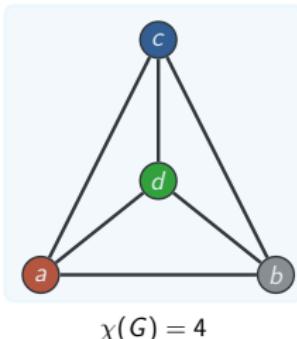
- ▶ The chromatic number of a graph G , called $\chi(G)$, is at least its clique number Lower bound



Graph coloring

The clique number of a graph, $G = (V, E)$, is the number of vertices in the largest clique in G .

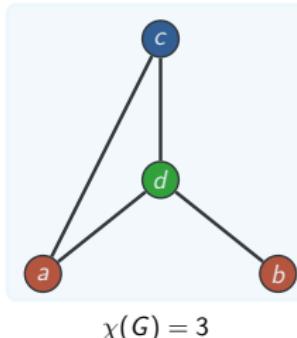
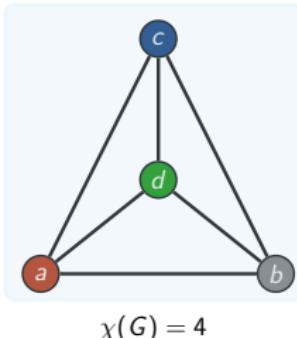
- ▶ The chromatic number of a graph G , called $\chi(G)$, is at least its clique number Lower bound
- ▶ Let $\Delta(G)$ be the largest degree of any vertex in the graph, G . Thus $\chi(G) \leq \Delta(G) + 1$



Graph coloring

The clique number of a graph, $G = (V, E)$, is the number of vertices in the largest clique in G .

- ▶ The chromatic number of a graph G , called $\chi(G)$, is at least its clique number Lower bound
- ▶ Let $\Delta(G)$ be the largest degree of any vertex in the graph, G . Thus $\chi(G) \leq \Delta(G) + 1$ Upper bound



Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient

1. Number all the vertices and number your colors;
2. Give a color to the first vertex;
3. Take the remaining vertices in order. Assign each one the lowest numbered color, that is different from the colours of its neighbours.

Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .

1. Number all the vertices and number your colors;
2. Give a color to the first vertex;
3. Take the remaining vertices in order . Assign each one the lowest numbered color, that is different from the colours of its neighbours.

Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

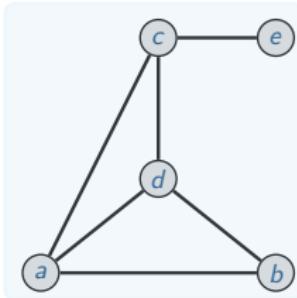
2. The Welsh-Powell algorithm: slightly more complicated, but can give better colorings.

1. Sort the vertices in non-increasing order of their degree;
2. Colour to the first vertex;
3. Take the next sorted vertice, giving that new or old color to the vertice depending if it is connected to one previously colored or not .

Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

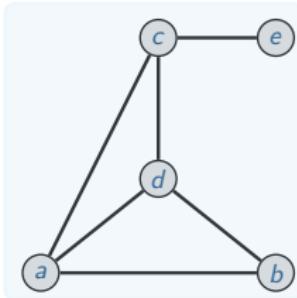
1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .



Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .

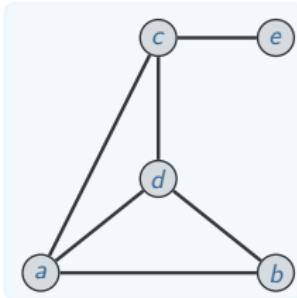


b-e-c-d-a

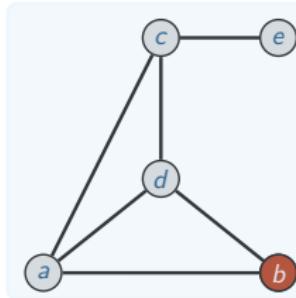
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .



b-e-c-d-a

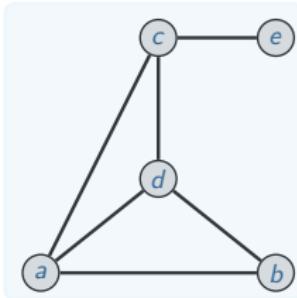


b-e-c-d-a

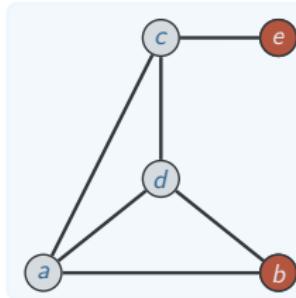
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .



b-e-c-d-a

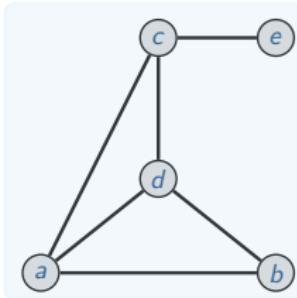


b-e-c-d-a

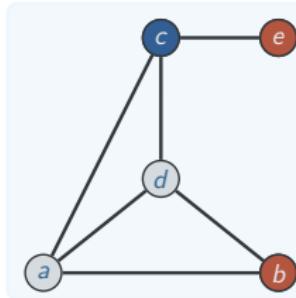
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .



b-e-c-d-a

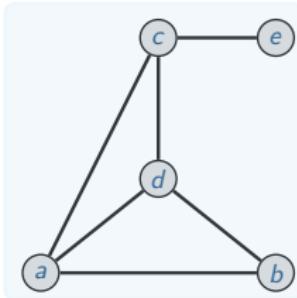


b-e-c-d-a

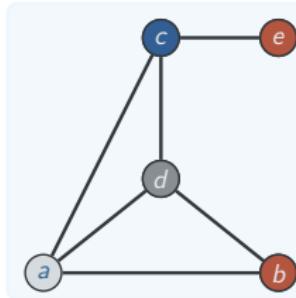
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .



b-e-c-d-a

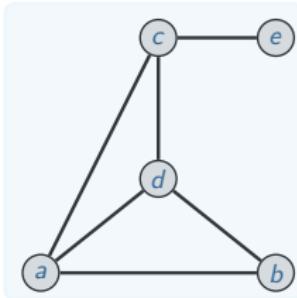


b-e-c-d-a

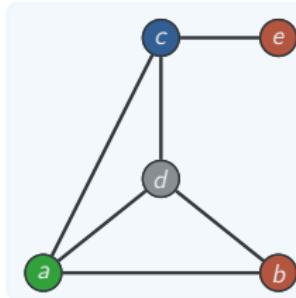
Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

1. The Greedy algorithm: simple and efficient,
but the result can depend on the ordering of the vertices .



b-e-c-d-a

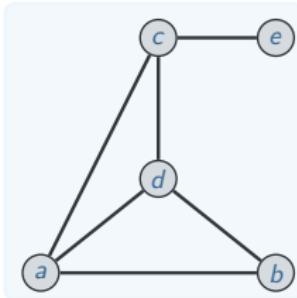


b-e-c-d-a

Graph coloring

There are some algorithms that are efficient, but not optimal to compute a vertex coloring (that is proper too as defined).

2. The Welsh-Powell algorithm: slightly more complicated, but can give better colorings.

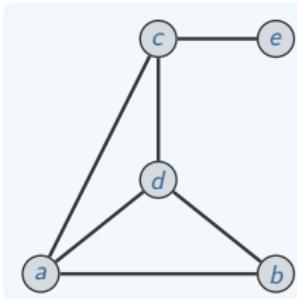


a-d-c-b-e

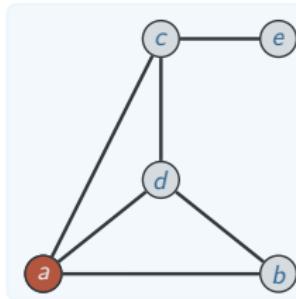
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

2. The Welsh-Powell algorithm: slightly more complicated, but can give better colorings.



a-d-c-b-e

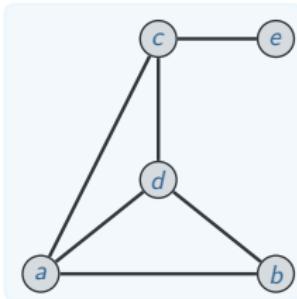


a-d-c-b-e

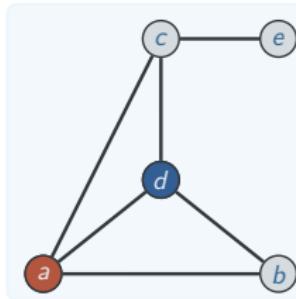
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

2. The Welsh-Powell algorithm: slightly more complicated, but can give better colorings.



a-d-c-b-e

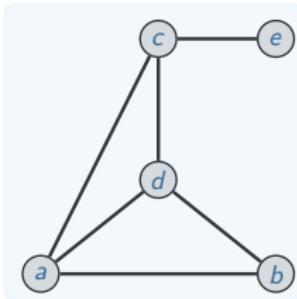


a-d-c-b-e

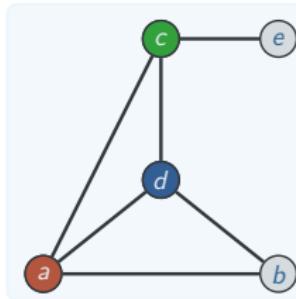
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

2. The Welsh-Powell algorithm: slightly more complicated, but can give better colorings.



a-d-c-b-e

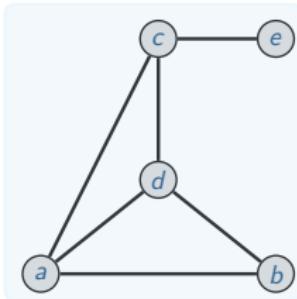


a-d-c-b-e

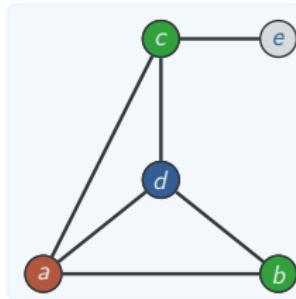
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

2. The Welsh-Powell algorithm: slightly more complicated, but can give better colorings.



a-d-c-b-e

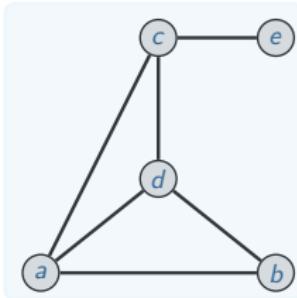


a-d-c-b-e

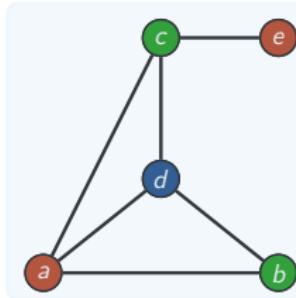
Graph coloring

There are some algorithms that are efficient, but **not optimal** to compute a vertex coloring (that is proper too as defined).

2. The Welsh-Powell algorithm: slightly more complicated, but can give better colorings.



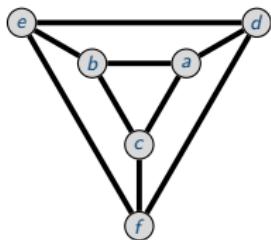
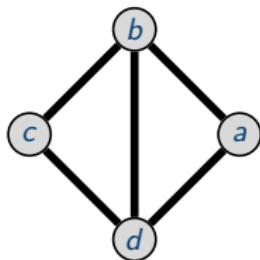
a-d-c-b-e



a-d-c-b-e

Edge coloring

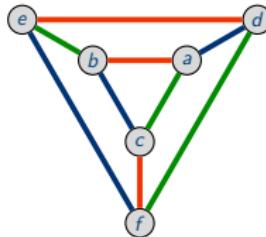
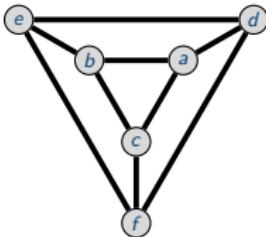
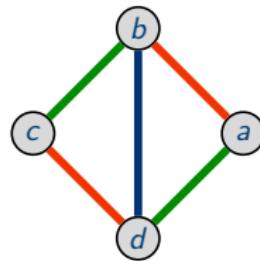
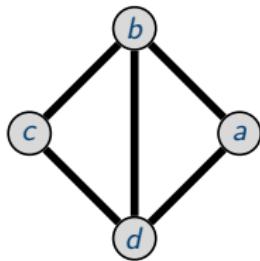
Let $G = (V, E)$ be a undirected connected graph.



Edge coloring

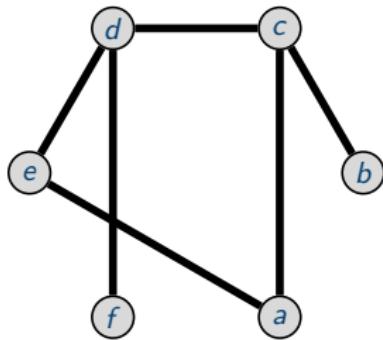
Let $G = (V, E)$ be a undirected connected graph.

- Edge Coloring is an assignment of colors to the edges of G in which adjacent edges are colored differently.



Edge coloring

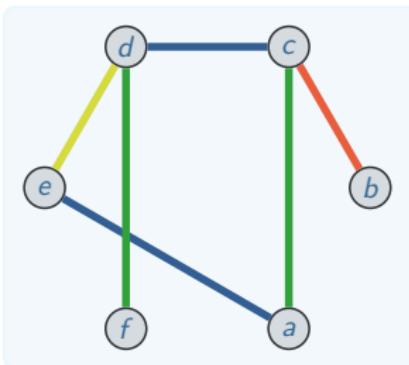
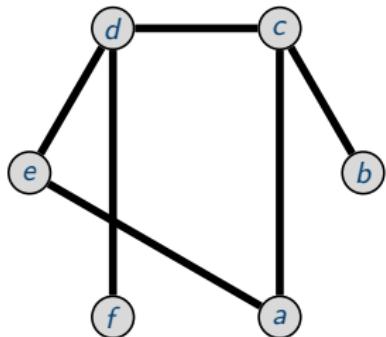
Let $G = (V, E)$ be a undirected connected graph.



Edge coloring

Let $G = (V, E)$ be a undirected connected graph.

- The graph G is **K -edge-colorable** if the edges can be colored by using K colors;

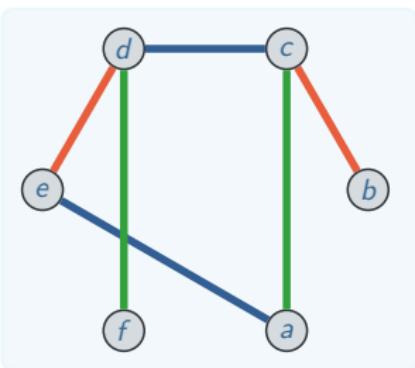
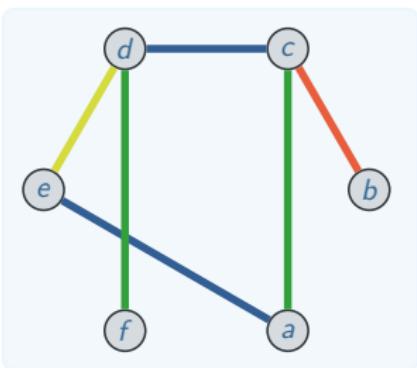
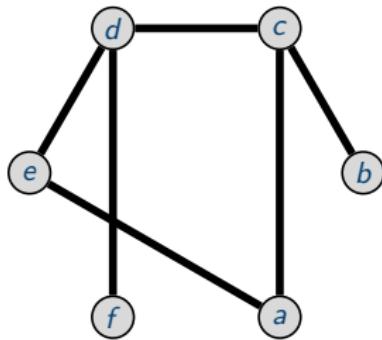


$$K = 4$$

Edge coloring

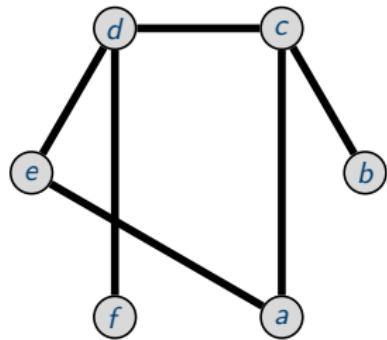
Let $G = (V, E)$ be a undirected connected graph.

- ▶ The graph G is **K -edge-colorable** if the edges can be colored by using K colors;
- ▶ The **chromatic number** $\chi'(G)$ is equal to the smallest number of K for coloring the edges of G .



Line graph

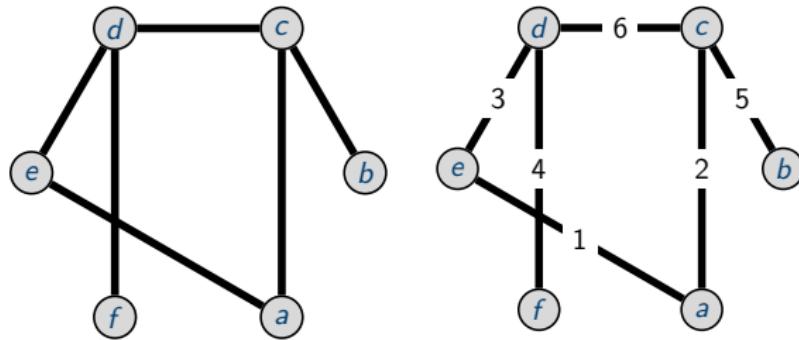
Let $G = (V, E)$ be a undirected connected graph.



Line graph

Let $G = (V, E)$ be a undirected connected graph. A **line graph** $L(G)$ is defined as follows:

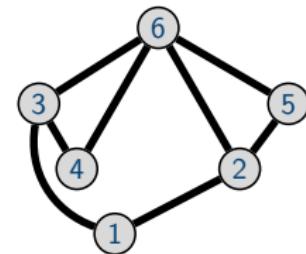
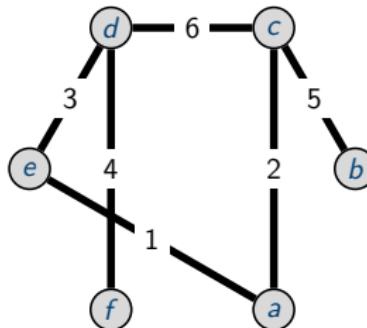
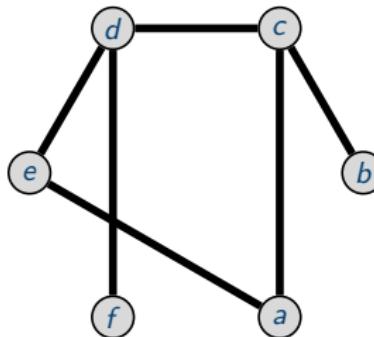
- The vertices of $L(G)$ are the edges of G ;



Line graph

Let $G = (V, E)$ be a undirected connected graph. A **line graph** $L(G)$ is defined as follows:

- ▶ The vertices of $L(G)$ are the edges of G ;
- ▶ Two vertices are adjacent in $L(G)$ if their corresponding edges in G are adjacent.

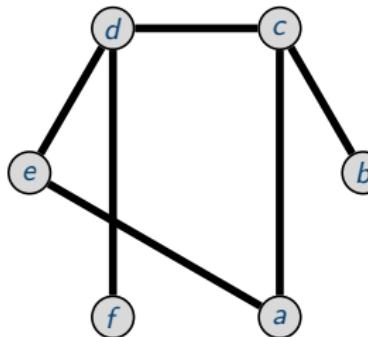


$$\chi'(G) = \chi(L(G))$$

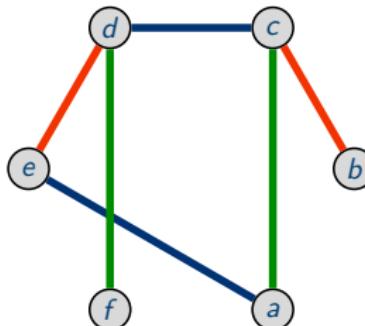
Line graph

Let $G = (V, E)$ be a undirected connected graph. A **line graph** $L(G)$ is defined as follows:

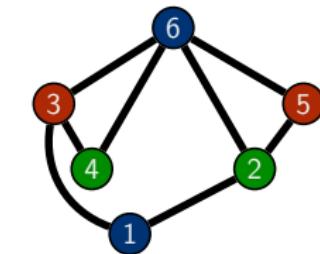
- ▶ The vertices of $L(G)$ are the edges of G ;
- ▶ Two vertices are adjacent in $L(G)$ if their corresponding edges in G are adjacent.



$$\chi'(G) = 3$$



$$\chi(L(G)) = 3$$



Questions?

Planar graphs
– Graph coloring –

Slots of time for exams – an example

A university is preparing a selection process for its n courses. How to organize the exams in order to minimize the number of days for the process in which each candidate can make just one exam per day. It's known that for the candidates will be applied specific exams depending on the course.

1. Computer Science – Math, Physics
2. Nutrition – Chemical, Biology, History
3. Architecture – Physics, Math, History
4. Biological Science - Chemical, Biology, Math

Slots of time for exams – an example

A university is preparing a selection process for its n courses. How to organize the exams in order to minimize the number of days for the process in which each candidate can make just one exam per day. It's known that for the candidates will be applied specific exams depending on the course.

1. Computer Science – Math, Physics
2. Nutrition – Chemical, Biology, History
3. Architecture – Physics, Math, History
4. Biological Science - Chemical, Biology, Math

How to model this selection process as a graph problem?

Task completion – an example

An industry has N tasks to be done and M employees. Each employee was assigned to a set of tasks, and the length of each task is by one day. Thus, how many days are needed to finish all tasks?

Task completion – an example

An industry has N tasks to be done and M employees. Each employee was assigned to a set of tasks, and the length of each task is by one day. Thus, how many days are needed to finish all tasks?

How to model this process as a graph problem?

Selection process – an example

A software house is hiring. For the positions, there are N software developers with different skills that will participate for the selection process. From a list of projects, each candidate must indicate just one project in which it wish to work. The interview for a specific candidate will be conducted by manager of the chosen project. How many slots of time are needed to the whole selection process?

Selection process – an example

A software house is hiring. For the positions, there are N software developers with different skills that will participate for the selection process. From a list of projects, each candidate must indicate just one project in which it wish to work. The interview for a specific candidate will be conducted by manager of the chosen project. How many slots of time are needed to the whole selection process?

How to model this process as a graph problem?



Teoria dos Grafos e Computabilidade

— Sets on graphs —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Teoria dos Grafos e Computabilidade

— Independent sets —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

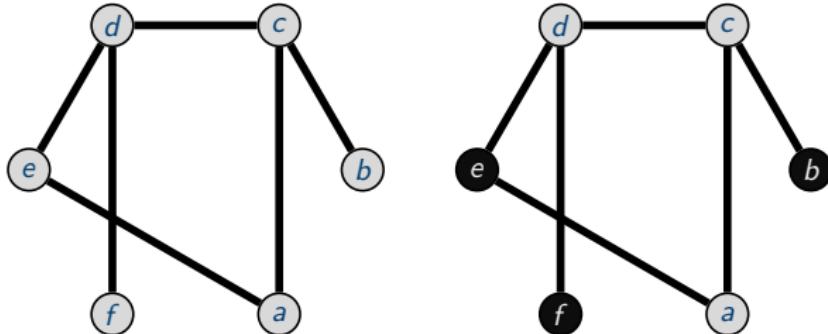
Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Independent sets

Let $G = (V, E)$ be an undirected connected graph.

- ▶ A subset $S \subseteq V$ is an **independent set** if $\forall u, v \in S$ there is no edge $(u, v) \in E$.
- ▶ Independent sets have also been called internally stable sets.



Independent sets

Let $G = (V, E)$ be an undirected connected graph, and S an independent set of G

- ▶ We say that the subset $S \subseteq V$ is a maximal independent set if there is no other independent set A in which $S \subset A$;
- ▶ The number of internal stability $\beta(G)$ is equal to the cardinality of the largest maximal independent set.

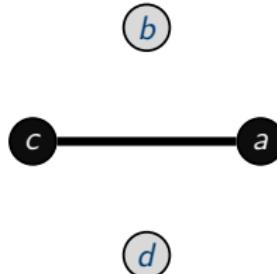
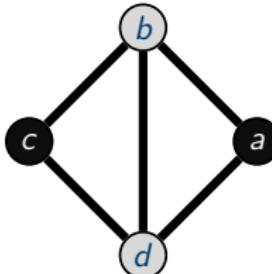
As S is an independent set of G , then S is a clique in the complement graph.

Independent sets

Let $G = (V, E)$ be an undirected connected graph, and S an independent set of G

- ▶ We say that the subset $S \subseteq V$ is a maximal independent set if there is no other independent set A in which $S \subset A$;
- ▶ The number of internal stability $\beta(G)$ is equal to the cardinality of the largest maximal independent set.

As S is an independent set of G , then S is a clique in the complement graph.



Independent sets

Let $G = (V, E)$ be an undirected connected graph. Design a method for computing an independent set of G

Algorithm: A method for computing an independent set

input : A graph $G = (V, E)$.

output: A independent set S

```
1  $S = \emptyset;$ 
2 while  $V \neq \emptyset$  do
3    $u =$  vertex with the smallest degree in  $G$ ;
4    $V = V - \{u\} - \Gamma(u);$ 
5    $S = S \cup \{u\};$ 
6 end
7 return  $S;$ 
```

Questions?

Sets on graphs
– Independent sets –

Teoria dos Grafos e Computabilidade

— Dominating sets —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

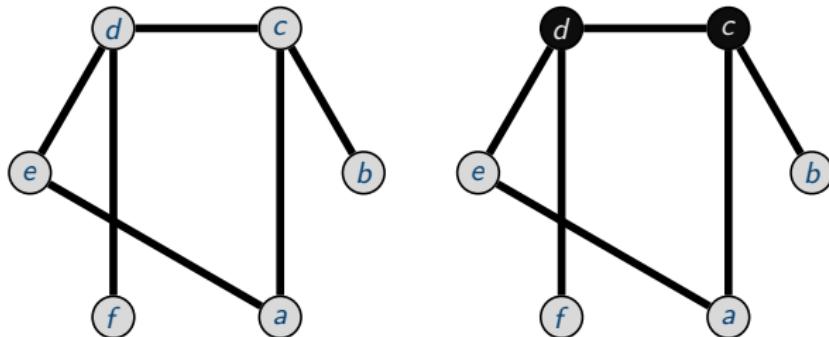
Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Dominating sets

Let $G = (V, E)$ be an undirected connected graph.

- ▶ A subset $S \subseteq V$ is an **dominating set** if $\forall u \in S$ there exist a $v \in V - S$ such that $(u, v) \in E$.
- ▶ Dominating sets have also been called externally stable sets.



Independent sets

Let $G = (V, E)$ be an undirected connected graph, and S a dominating set of G

- ▶ We say that the subset $S \subseteq V$ is a **minimal dominating set** if there is no other dominating set A in which $A \subset S$;
- ▶ The number of **external stability** $\beta(G)$ is equal to the cardinality of the smallest minimal dominating set.

Independent sets

Let $G = (V, E)$ be an undirected connected graph. Design a method for computing a dominance set of G

Algorithm: A method for computing a dominating set

input : A graph $G = (V, E)$.

output: A dominating set D

```
1  $D = \emptyset;$ 
2 while  $V \neq \emptyset$  do
3    $u =$  vertex with the highest degree in  $G$ ;
4    $V = V - \{u\} - \Gamma(u);$ 
5    $D = D \cup \{u\};$ 
6 end
7 return  $D;$ 
```

Questions?

Sets on graphs
– Dominating sets –

Teoria dos Grafos e Computabilidade

— Vertex cover —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF

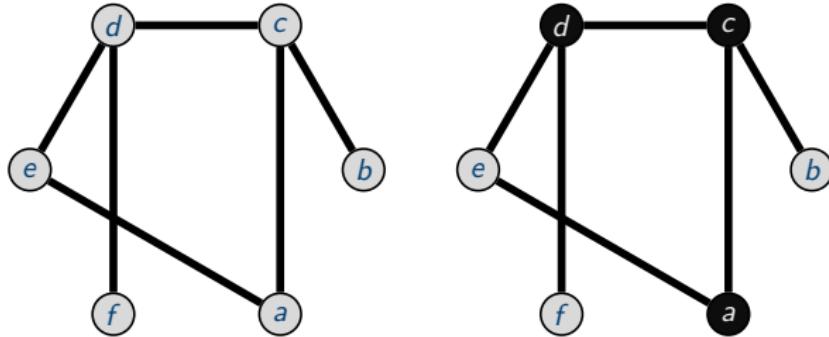
Image and Multimedia Data Science Laboratory – IMScience

Pontifical Catholic University of Minas Gerais – PUC Minas

Vertex cover

Let $G = (V, E)$ be an undirected connected graph.

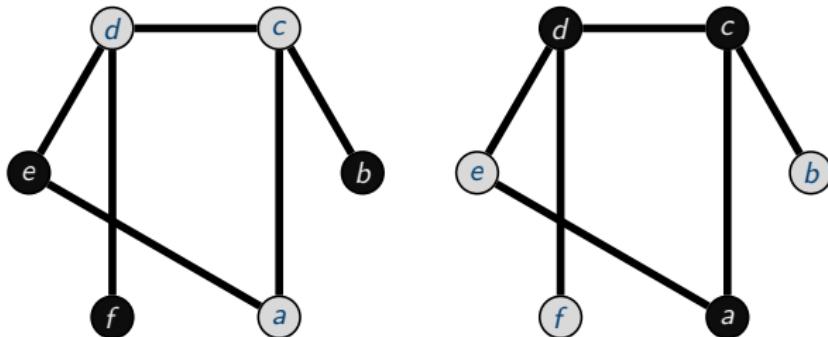
- ▶ A subset $S \subseteq V$ is an **vertex cover** if $\forall(u, v) \in E$, either $u \in S$ or $v \in S$.



Vertex cover

Let $G = (V, E)$ be an undirected connected graph, and S a vertex cover of G

As S is a vertex cover of G , then $V - S$ is an independent set.



Vertex cover

Let $G = (V, E)$ be an undirected connected graph. Design a method for computing a vertex cover in G

Algorithm: A method for computing a minimum vertex cover

input : A graph $G = (V, E)$.

output: A independent set S

```
1  $S = \emptyset;$ 
2 while  $E \neq \emptyset$  do
3   Let  $(u, v)$  an arbitrary edge of  $E$ ;
4   Choose either  $u$  or  $v$  to be included to  $C$ ;
5    $S = S \cup \{u\}$  for instance;
6    $V = V - u;$ 
7 end
8 return  $S;$ 
```

Questions?

Sets on graphs
– Vertex cover –