Problemas NP-Completos

Fábio Botler

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro

Aula de hoje

- Algoritmos eficientes
- Problemas tratáveis e intratáveis
- Problemas algorítmicos
- Codificações
- Tipos de problemas
- A Classe P
- Problemas aparentemente difíceis

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

↳ TAMANHO DE I

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$
- $O(n^{1000^{32938}})$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$
- $O(n^{1000^{32938}})$
- $O(n^4 \log n)$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(1)$
- $O(n^2)$
- $O(n^{1000})$
- $O(n^{1000^{32938}})$
- $O(n^4 \log n)$
- $O(n \log \log n)$

LINEAR = $O(n)$ $\Rightarrow \exists C$ T.q. $\leq |C| \cdot n$ $\leq |C| n^2$ $n \geq 1$

$\rightsquigarrow$ EFICIENTE

$= O(n^5)$

$= O(n^2)$ $\log n < n$ $n \to \infty$

Definição

Um problema é dito **tratável** se existe algoritmo eficiente que o resolva, e **intratável** caso contrário.

Definição

Um problema é dito **tratável** se existe algoritmo eficiente que o resolva, e **intratável** caso contrário.

Como mostramos que um problema Π é tratável?

Definição

Um problema é dito **tratável** se existe algoritmo eficiente que o resolva, e **intratável** caso contrário.

Como mostramos que um problema Π é tratável?

- $\triangleright$ Basta mostrar um algoritmo eficiente que resolva Π!

Definição

Um problema é dito **tratável** se existe algoritmo eficiente que o resolva, e **intratável** caso contrário.

Como mostramos que um problema Π é tratável?

$\triangleright$ Basta mostrar um algoritmo eficiente que resolva Π!

Se não conhecemos um algoritmo eficiente para resolver um problema Π, então Π é intratável?

Definição

Um problema é dito **tratável** se existe algoritmo eficiente que o resolva, e **intratável** caso contrário.

Como mostramos que um problema Π é tratável?

- $\triangleright$ Basta mostrar um algoritmo eficiente que resolva Π!

Se não conhecemos um algoritmo eficiente para resolver um problema Π, então Π é intratável?

- $\triangleright$ Não.

Definição

Um problema é dito **tratável** se existe algoritmo eficiente que o resolva, e **intratável** caso contrário.

Como mostramos que um problema Π é tratável?

- ▷ Basta mostrar um algoritmo eficiente que resolva Π!

Se não conhecemos um algoritmo eficiente para resolver um problema Π, então Π é intratável?

- ▷ Não.
- ▷ Precisamos mostrar que TODO algoritmo que resolve Π não é eficiente.

Alguns problemas problemas tratáveis:

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo

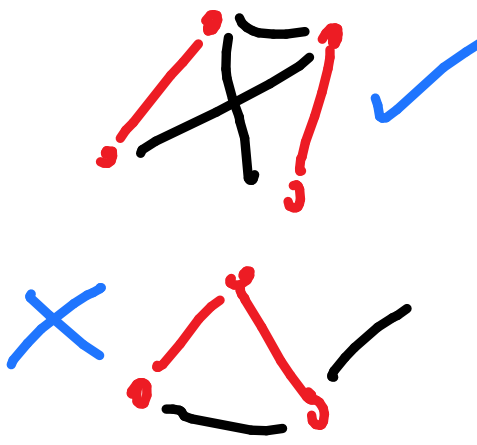
Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo

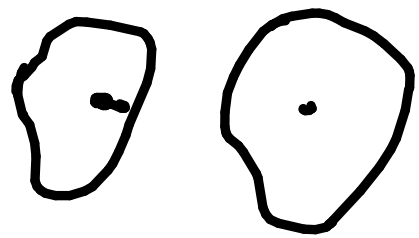
CONJ. ARESTAS $M \subseteq E(G)$ SEM VÉRTICES EM COMUM



Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo

↳ conj. $C \subseteq E(G)$ t.q. $G \setminus C$ é desconexo. (conexo)



Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo $O(|V| + |E|)$

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo
- Decidir se um grafo é planar

Alguns problemas problemas tratáveis:

- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo
- Decidir se um grafo é planar
- Decidir se um grafo é bipartido

Alguns problemas problemas tratáveis:

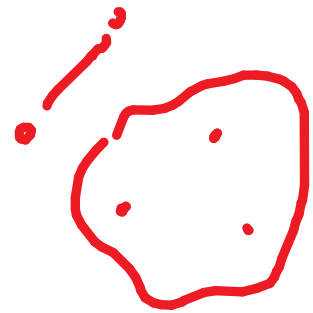
- Encontrar um caminho mínimo entre dois vértices de um grafo
- Multiplicar matrizes
- Encontrar um emparelhamento máximo em um grafo
- Encontrar um corte mínimo em um grafo
- Decidir se um grafo é conexo
- Decidir se um grafo é planar LINEAR?
- Decidir se um grafo é bipartido
- Decidir se um número é primo AKS

Alguns problemas que **não sabemos** se são tratáveis:

Alguns problemas que **não sabemos** se são tratáveis:

- Encontrar um caminho máximo em um grafo

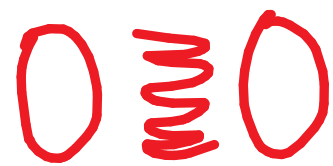
Alguns problemas que **não sabemos** se são tratáveis:



- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo 3-UNIFORME

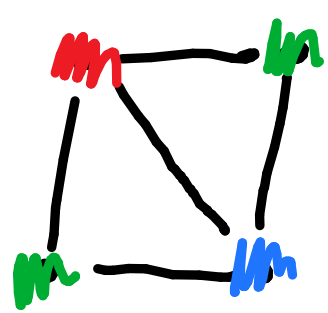
Alguns problemas que **não sabemos** se são tratáveis:

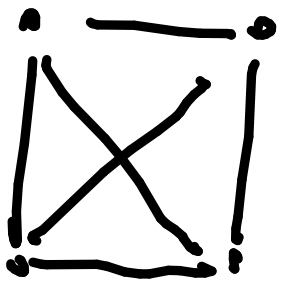
- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo
- Encontrar um corte máximo em um grafo



Alguns problemas que **não sabemos** se são tratáveis:

- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo
- Encontrar um corte máximo em um grafo
- Decidir se um grafo pode ser colorido com três cores





$c : V(G) \to \{1, 2, 3\}$

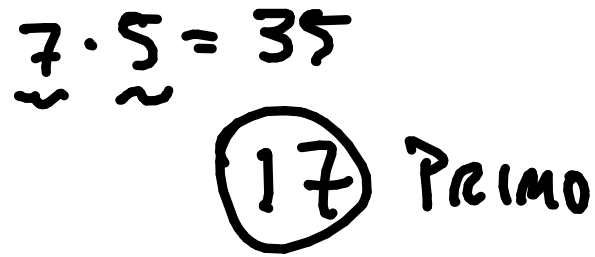
$\forall\, xy \in E(G)$ temos

$c(x) \neq c(y)$

Col. com 3 cores $= 3^n$

Alguns problemas que **não sabemos** se são tratáveis:

- Encontrar um caminho máximo em um grafo
- Encontrar um emparelhamento máximo em um hipergrafo
- Encontrar um corte máximo em um grafo
- Decidir se um grafo pode ser colorido com três cores
- Encontrar uma fatorização de um número composto



Tratáveis	**Tratabilidade desconhecida**
Caminho mínimo	Caminho máximo
Multiplicação de matrizes	
Emparelhamento em grafos	Emparelhamento em hipergrafos
Corte Mínimo	Corte máximo
Conexidade	
Planaridade	
2-coloração BIPARTIDO	3-coloração
Primalidade AKS	Fatorização de inteiros RSA

Definição

$\Pi \subseteq \Sigma^*$ $\Sigma = \{0,1\}$

Um **problema algorítmico** Π é um par $(\mathcal{C}_\Pi, Q_\Pi)$, onde $\mathcal{C}_\Pi$ é o **conjunto de dados**, também chamados de **instâncias**, do problema e uma questão solicitada Q_Π, chamada **objetivo do problema**.

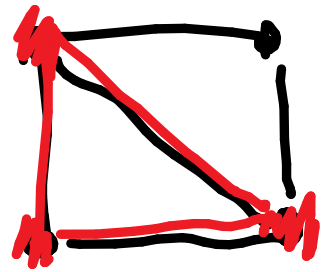
Definição

Um **problema algorítmico** Π é um par $(\mathcal{C}_\Pi, Q_\Pi)$, onde $\mathcal{C}_\Pi$ é o **conjunto de dados**, também chamados de **instâncias**, do problema e uma questão solicitada Q_Π, chamada **objetivo do problema**.

Resolver um problema Π é desenvolver um algoritmo cuja entrada sejam os elementos de $\mathcal{C}_\Pi$, e cuja saída, chamada **solução**, responda ao objetivo do problema.

Exemplo

`Clique`**(G,k)**



DADOS: Um grafo G e um inteiro $k > 0$
OBJETIVO: Encontrar uma clique
de tamanho pelo menos k em G.

Exemplo

`Clique`**(G,k)**

DADOS: Um grafo G e um inteiro $k > 0$
OBJETIVO: Encontrar uma clique
de tamanho pelo menos k em G.

Conjunto de dados: o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.

Exemplo

`Clique`**(G,k)**

DADOS: Um grafo G e um inteiro $k > 0$
OBJETIVO: Encontrar uma clique
de tamanho pelo menos k em G.

Conjunto de dados: o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.

Instância: um par (G, k), onde G é um grafo, e k é um inteiro positivo.

Exemplo

`Clique`**(G,k)**

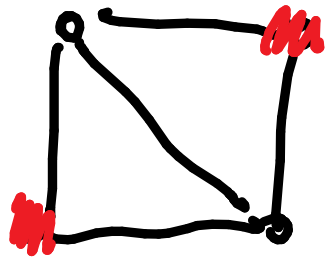
DADOS: Um grafo G e um inteiro $k > 0$
OBJETIVO: Encontrar uma clique
de tamanho pelo menos k em G.

Conjunto de dados: o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.

Instância: um par (G, k), onde G é um grafo, e k é um inteiro positivo.

Solução: uma clique em G com pelo menos k vértices, se existir.

Exemplo



`Conjunto Independente`**(G,k)**

DADOS: Um grafo G e um inteiro $k > 0$

OBJETIVO: Decidir se G possui conjunto independente de tamanho pelo menos k.

Exemplo

`Conjunto Independente`**(G,k)**

DADOS:	Um grafo G e um inteiro $k > 0$
OBJETIVO:	Decidir se G possui conjunto independente de tamanho pelo menos k.

Conjunto de dados:	o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.

Exemplo

`Conjunto Independente`**(G,k)**

DADOS:	Um grafo G e um inteiro $k > 0$
OBJETIVO:	Decidir se G possui conjunto independente de tamanho pelo menos k.

Conjunto de dados:	o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.
Instância:	um par (G, k), onde G é um grafo, e k é um inteiro positivo.

Exemplo

`Conjunto Independente`**(G,k)**

DADOS: Um grafo G e um inteiro $k > 0$
OBJETIVO: Decidir se G possui conjunto independente de tamanho pelo menos k.

Conjunto de dados: o conjunto de todos os pares (G, k), onde G é um grafo, e k é um inteiro positivo.

Instância: um par (G, k), onde G é um grafo, e k é um inteiro positivo.

Solução: SIM, caso G possua um conjunto independente de tamanho pelo menos k,
ou NÃO, caso todo conjunto independente de G tenha tamanho menor que k.

Codificações

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

- ▷ Porque é a medida pela qual calculamos a complexidade do algoritmo.

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

- ▷ Porque é a medida pela qual calculamos a complexidade do algoritmo.

Para um grafo usaremos o número de vértices ou de arestas como parâmetro.

Codificações

Cada instância deve ser apresentada em uma codificação "razoável", isso é, que não seja desnecessariamente longa.

Porque o tamanho da instância é importante?

- ▷ Porque é a medida pela qual calculamos a complexidade do algoritmo.

Para um grafo usaremos o número de vértices ou de arestas como parâmetro.

Em geral uma codificação é desnecessariamente longa quando

- ▷ contém partes irrelevantes para o problema;
- ▷ algum inteiro da instância está codificado no sistema unário.

É natural que a codificação de uma instância aumente quando a instância aumenta.

É natural que a codificação de uma instância aumente quando a instância aumenta.

Porém codificar um número na base unária traz alguns problemas:

É natural que a codificação de uma instância aumente quando a instância aumenta.

Porém codificar um número na base unária traz alguns problemas:

Exemplo

Base	*Representação*	*Tamanho*
10	*130*	*1*
1	*111111111111111111111111111111* *111111111111111111111111111111* *111111111111111111111111111111* *111111111111111111111111111111* *1111111111*	*130*
2	*10000010*	*8*
3	*11211*	*5*
4	*2002*	*4*

FATORIZAÇÃO (K)

FOR $i = 1 \ldots K$:

IF $\frac{K}{i}$ CONST. É INTEIRO :

RETURN $i, K/i$

RETURN PRIMO

$= O(K)$

LINEAR?

$= O(2^{\log_2(K)})$

FATORIZAÇÃO (K) $\rightsquigarrow K$

FATORIZAÇÃO (2K) $\rightsquigarrow 2K$

$K = 5$	101
$K = 10$	1010

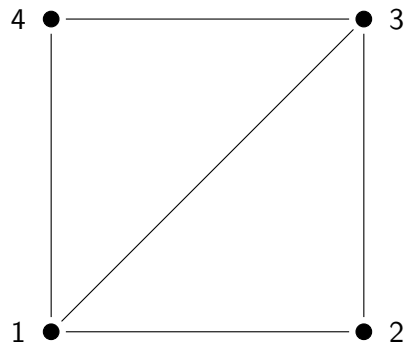
CHAVE DE 128 BITS : _ _ _ _ _ _ _ _ ... _ _ _

2^{128}

2^{50}

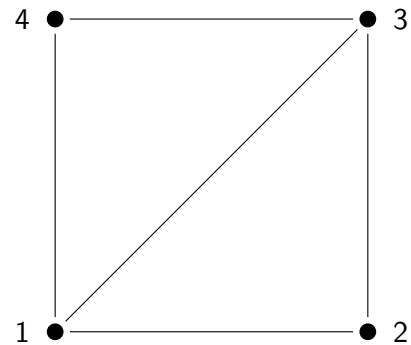
Exemplo

Vamos codificar o grafo.



Exemplo

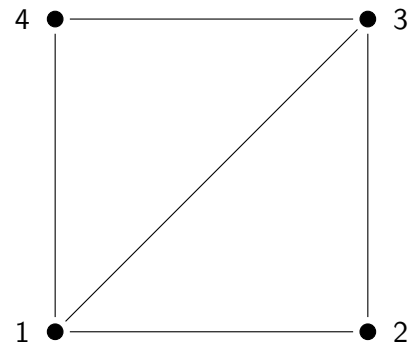
Vamos codificar o grafo.



G pode ser descrito por $\{12, 23, 34, 41, 13\}$

Exemplo

Vamos codificar o grafo.

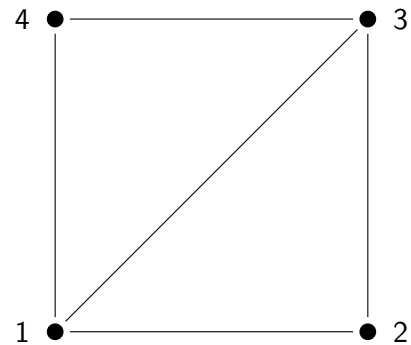


G pode ser descrito por $\{12, 23, 34, 41, 13\}$

Se a aresta xy é codificada por x_b, y_b, onde x_b é a representação de x na base b.

Exemplo

Vamos codificar o grafo.



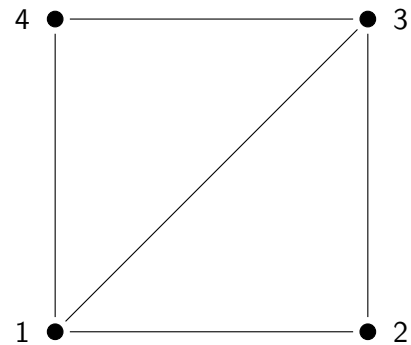
G pode ser descrito por $\{12, 23, 34, 41, 13\}$

Se a aresta xy é codificada por $/x_b, y_b/$, onde x_b é a representação de x na base b.

- ▷ Binária: $/1, 10/10, 11/11, 100/100, 1/1, 11/$

Exemplo

Vamos codificar o grafo.



G pode ser descrito por $\{12, 23, 34, 41, 13\}$

Se a aresta xy é codificada por $/x_b, y_b/$, onde x_b é a representação de x na base b.

- ▷ Binária: $/1, 10/10, 11/11, 100/100, 1/1, 11/$
- ▷ Unária: $/1, 11/11, 111/111, 1111/1111, 1/1, 111/$

Tipos de problemas

Há três tipos de problemas algorítmicos.

Tipos de problemas

Há três tipos de problemas algorítmicos.

1) problemas de **decisão**

2) problemas de **localização**

3) problemas de **otimização**

Exemplo

`Problema do caixeiro-viajante`

Exemplo

`Problema do caixeiro-viajante`

Versão de decisão

DADOS: Um grafo G com pesos nas arestas e um inteiro $k > 0$

OBJETIVO: **Decidir** se G possui um ciclo Hamiltoniano com peso menor ou igual a k.

Exemplo

`Problema do caixeiro-viajante`

Versão de decisão

DADOS: Um grafo G com pesos nas arestas e um inteiro $k > 0$

OBJETIVO: **Decidir** se G possui um ciclo Hamiltoniano com peso menor ou igual a k.

Versão de localização

DADOS: Um grafo G com pesos nas arestas e um inteiro $k > 0$

OBJETIVO: **Encontrar** em G um ciclo Hamiltoniano com peso menor ou igual a k.

Exemplo

$\Pi_{SIM} \subseteq \{0,1\}^*$

Problema do caixeiro-viajante

Versão de decisão

DADOS: Um grafo G com pesos nas arestas e um inteiro $k > 0$

OBJETIVO: **Decidir** se G possui um ciclo Hamiltoniano com peso menor ou igual a k.

Versão de localização

DADOS: Um grafo G com pesos nas arestas e um inteiro $k > 0$

OBJETIVO: **Encontrar** em G um ciclo Hamiltoniano com peso menor ou igual a k.

Versão de otimização

DADOS: Um grafo G com pesos nas arestas

OBJETIVO: Encontrar em G um ciclo Hamiltoniano de custo **ótimo** (mínimo/máximo).

Qual a relação entre as versões de um mesmo problema?

Qual a relação entre as versões de um mesmo problema?

Podemos usar um para resolver outro?

- ▷ É possível usar a versão de otimização para resolver a versão de localização,

Qual a relação entre as versões de um mesmo problema?

Podemos usar um para resolver outro?

- ▷ É possível usar a versão de otimização
  para resolver a versão de localização,

- ▷ e usar a versão de localização para resolver a versão de decisão

Qual a relação entre as versões de um mesmo problema?

Podemos usar um para resolver outro?

- ▷ É possível usar a versão de otimização
  para resolver a versão de localização,

- ▷ e usar a versão de localização para resolver a versão de decisão

Frequentemente, é possível usar a versão de decisão para resolver a versão de otimização!

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(n^2)$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(n^2)$
- $O(n^{1000})$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(n^2)$
- $O(n^{1000})$
- $O(n \log \log n)$

Definição

Um algoritmo A é dito **eficiente** se sua complexidade for um polinômio no tamanho da sua entrada.

A é eficiente se existe inteiro k tal que para toda instância I, o número de operações que A realiza para resolver I é $O(|I|^k)$.

Exemplo

- $O(n^2)$
- $O(n^{1000})$
- $O(n \log \log n)$

Também usaremos

▷ **polinomial** para algoritmos eficientes

▷ **exponencial** para algoritmos de complexidade $O(2^n)$

A eficiência do algoritmo representa apenas o **caso assintótico**.

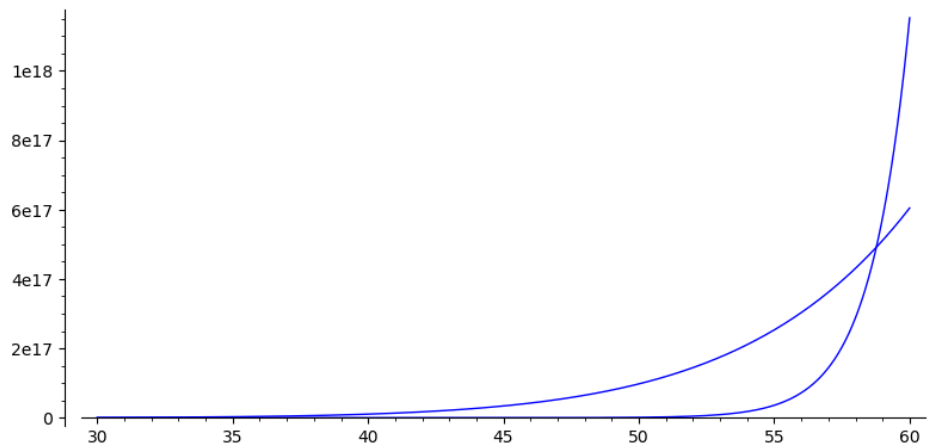
A eficiência do algoritmo representa apenas o **caso assintótico**.

Exemplo

Dois algoritmos A_1 *e* A_2 *para um mesmo problema, tal que*

$$A_1 = \theta(n^{10}) \text{ e } A_2 = \theta(2^n)$$

Para uma instância de tamanho $n = 2$, A_1 *realiza algo da ordem de* 1024 *operações, enquanto* A_2 *realiza algo da ordem de* 4 *operações.*



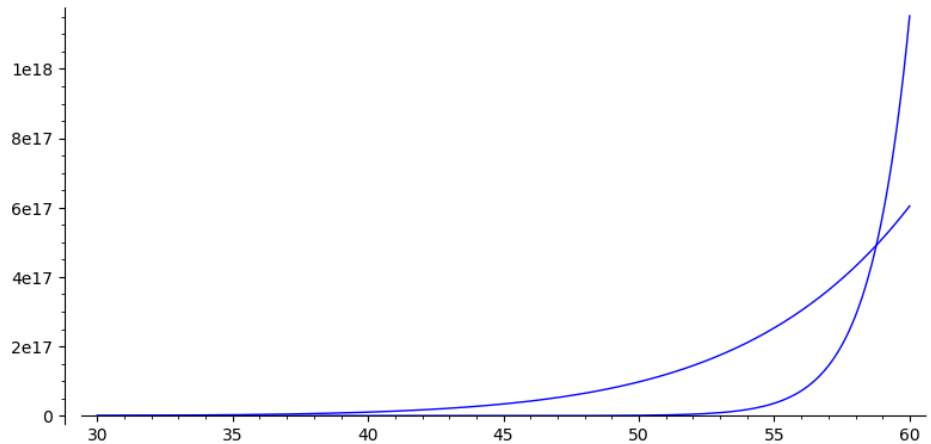
A eficiência do algoritmo representa apenas o **caso assintótico**.

Exemplo

Dois algoritmos A_1 *e* A_2 *para um mesmo problema, tal que*

$$A_1 = \theta(n^{10}) \text{ e } A_2 = \theta(2^n)$$

Para uma instância de tamanho $n = 2$, A_1 *realiza algo da ordem de* 1024 *operações, enquanto* A_2 *realiza algo da ordem de* 4 *operações.*



Um algoritmo de complexidade $\theta(n^{1000})$ é eficiente só em teoria.

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

Exemplo

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

Exemplo

- *programação linear;*

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

Exemplo

- *programação linear;*
- *máximo divisor comum;*

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

Exemplo

- *programação linear;*
- *máximo divisor comum;*
- *emparelhamento máximo;*

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

Exemplo

- *programação linear;*
- *máximo divisor comum;*
- *emparelhamento máximo;*
- *decidir se um número é primo;*

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

Exemplo

- *programação linear;*
- *máximo divisor comum;*
- *emparelhamento máximo;*
- *decidir se um número é primo;*
- *decidir se um grafo é bipartido, ou se admite uma 2-coloração própria;*

A Classe P

$$P = \left\{ \begin{array}{l} \text{problemas de decisão que} \\ \text{admitem algoritmo polinomial} \end{array} \right\}$$

Exemplo

- *programação linear;*
- *máximo divisor comum;*
- *emparelhamento máximo;*
- *decidir se um número é primo;*
- *decidir se um grafo é bipartido, ou se admite uma 2-coloração própria;*
- *decidir se um grafo é planar.* *linear*

NÃO CONHECEMOS alg. EFICIENTE

O problema de decidir se um grafo possui um circuito Hamiltoniano está em P?

EFICIENTE

- ▷ Como não é conhecido nenhum algoritmo polinomial para o Problema de Circuito Hamiltoniano, não podemos concluir que ele está em P.

- ▷ Por outro lado, não há prova de que esse problema não está em P.

Alguns problemas "aparentemente" difíceis

Satisfatibilidade

DADOS: Uma expressão booleana E
na Forma Normal Conjuntiva (FNC)

OBJETIVO: E é satisfatível?

Satisfatibilidade

DADOS: Uma expressão booleana E
na Forma Normal Conjuntiva (FNC)

OBJETIVO: E é satisfatível?

Conjunto de **variáveis booleanas**

$$x_1, x_2, \ldots$$

e suas negações

$$\overline{x_1}, \overline{x_2}, \ldots$$

Satisfatibilidade

DADOS: Uma expressão booleana E
na Forma Normal Conjuntiva (FNC)
OBJETIVO: E é satisfatível?

Conjunto de **variáveis booleanas**

$$x_1, x_2, \ldots$$

e suas negações

$$\overline{x_1}, \overline{x_2}, \ldots$$

Também chamados de **literais**.

Satisfatibilidade

DADOS: Uma expressão booleana E
na Forma Normal Conjuntiva (FNC)
OBJETIVO: E é satisfatível?

Conjunto de **variáveis booleanas**

$$x_1, x_2, \ldots$$

e suas negações

$$\overline{x_1}, \overline{x_2}, \ldots$$

$$\neg x_1 = \begin{cases} F \text{ se } x_1 = T \\ T \text{ se } x_1 = F \end{cases}$$

Também chamados de **literais**.

x_i pode ser **verdadeiro** ou **falso**
x_i é verdadeiro se e somente se $\bar{x}_i$ é falso

$\wedge$ conjunção
$\vee$ disjunção

Operadores lógicos

$\wedge$ conjunção

Operadores lógicos

$\wedge$ conjunção - operador "e": isso E aquilo

Operadores lógicos

$\wedge$ conjunção - operador "e": isso E aquilo

$\vee$ disjunção

Operadores lógicos

$\wedge$ conjunção - operador "e": isso E aquilo

$\vee$ disjunção - operador "ou": isso OU aquilo (ou ambos)

Operadores lógicos

$x \wedge y$

$x \vee y$

$\wedge$ conjunção - operador "e": isso E aquilo

$\vee$ disjunção - operador "ou": isso OU aquilo (ou ambos)

$\wedge$	0	1
0	0	0
1	0	1

$\vee$	0	1
0	0	1
1	1	1

Forma normal conjuntiva

Uma **cláusula** é uma disjunção ($\vee$) de literais.

Forma normal conjuntiva

Uma **cláusula** é uma disjunção ($\vee$) de literais.

Exemplo

$$\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4$$

Forma normal conjuntiva

Uma **cláusula** é uma disjunção ($\vee$) de literais.

Exemplo

$$\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4$$

Uma expressão booleana é dita na **forma normal conjuntiva (FNC)** quando for uma conjunção ($\wedge$) de cláusulas.

Forma normal conjuntiva

Uma **cláusula** é uma disjunção ($\vee$) de literais.

Exemplo

$$\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4$$

Uma expressão booleana é dita na **forma normal conjuntiva (FNC)** quando for uma conjunção ($\wedge$) de cláusulas.

Exemplo

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_4})$$

Definição

Uma expressão booleana é dita **satisfatível** se existe uma atribuição de valores às suas variáveis de tal modo que o valor da expressão seja verdadeiro.

Exemplo

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_4})$$

Definição

Uma expressão booleana é dita **satisfatível** se existe uma atribuição de valores às suas variáveis de tal modo que o valor da expressão seja verdadeiro.

Exemplo

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3} \vee x_4) \wedge (\overline{x_4})$$

com $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$ *temos*

$$(1 \vee 0) \wedge (0 \vee 1 \vee 1 \vee 1) \wedge (1) = 1$$

Exemplo

$$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$$

Exemplo

$x_1 = 1$

$x_2 = 0$

$$(x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2})$$

0

Não é satisfatível.

Satisfatibilidade

n VARS

2^n

DADOS: Uma expressão booleana E
na Forma Normal Conjuntiva (FNC)

OBJETIVO: E é satisfatível?

Clique

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui uma clique de tamanho pelo menos k?

Clique

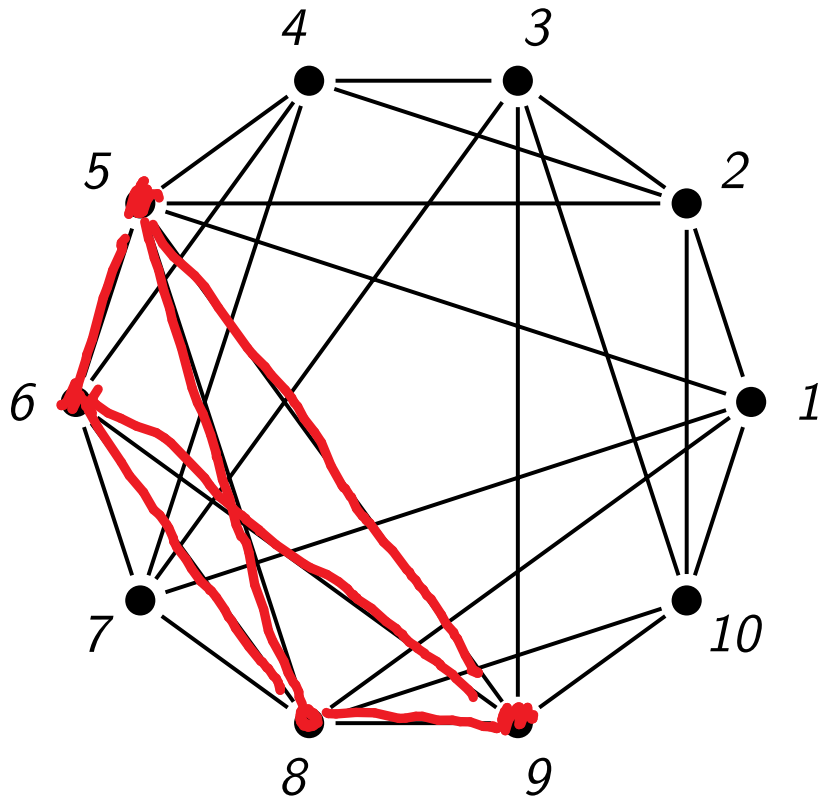
DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui uma clique de tamanho pelo menos k?

Exemplo

(G,3) SIM

(G,4) SIM

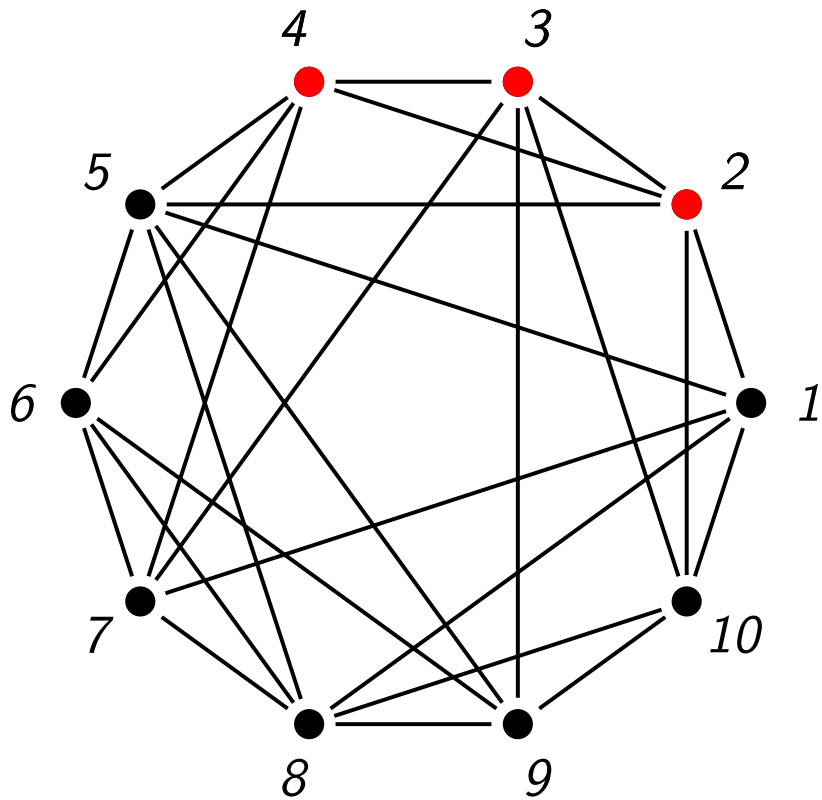
G =



Clique

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui uma clique de tamanho pelo menos k?

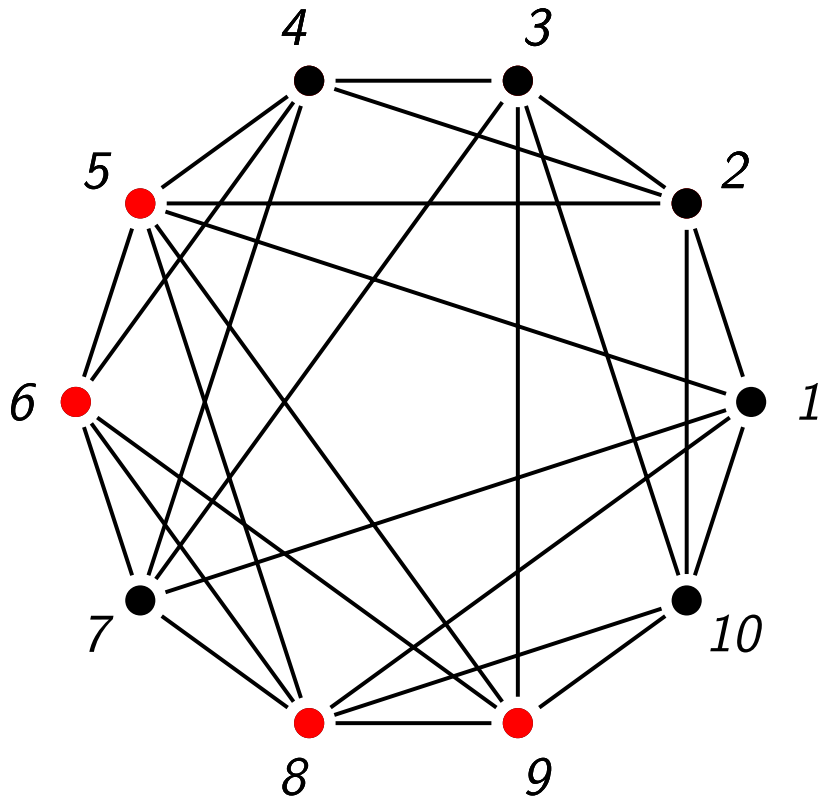
Exemplo



Clique

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui uma clique de tamanho pelo menos k?

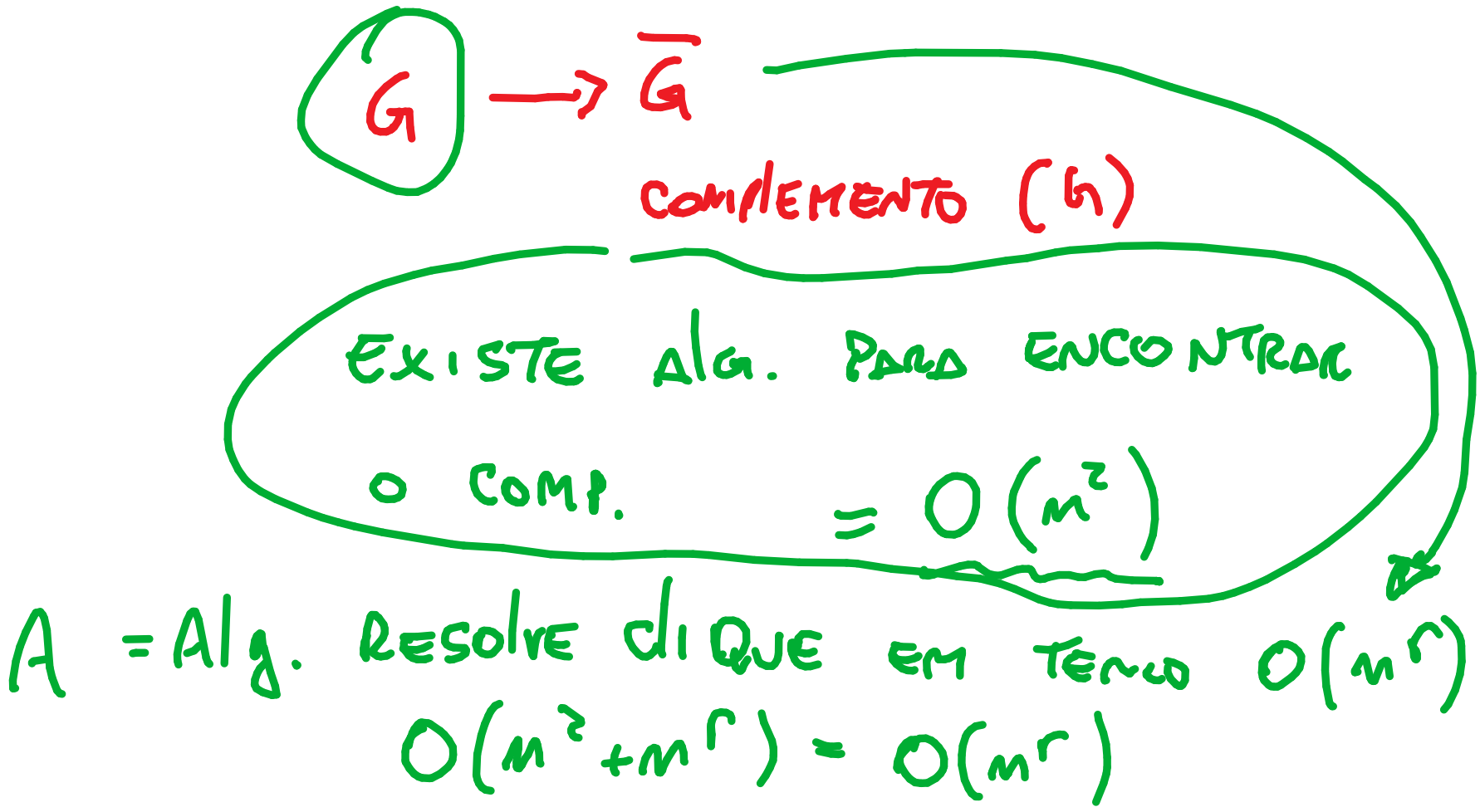
Exemplo



Conjunto Independente de Vértices

DADOS: Um grafo G e um inteiro k

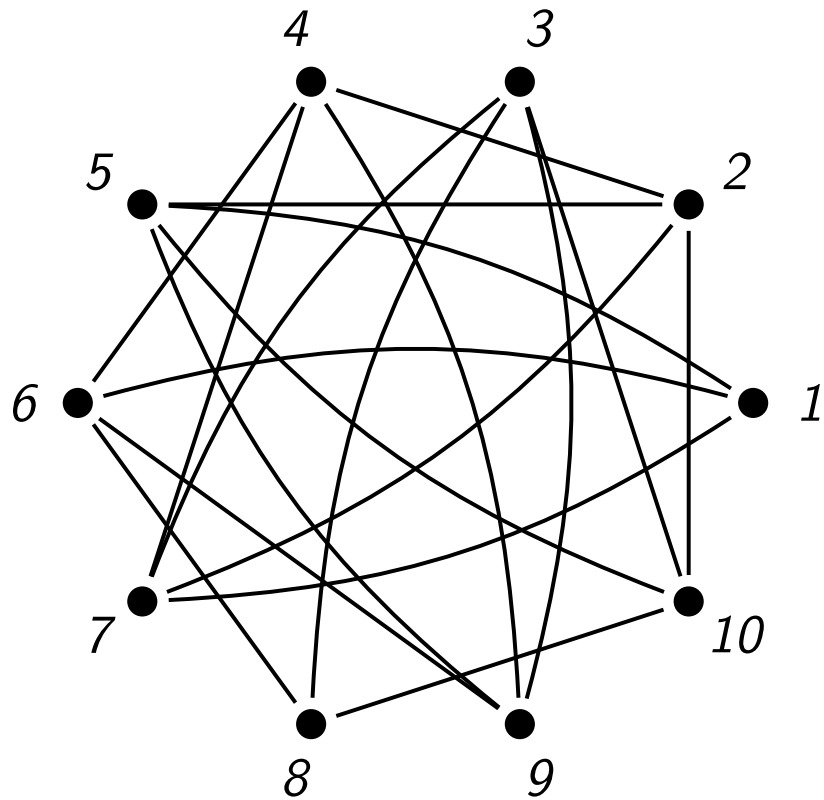
OBJETIVO: G possui um conjunto independente de vértices de tamanho pelo menos k?



Conjunto Independente de Vértices

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui um conjunto independente de vértices de tamanho pelo menos k?

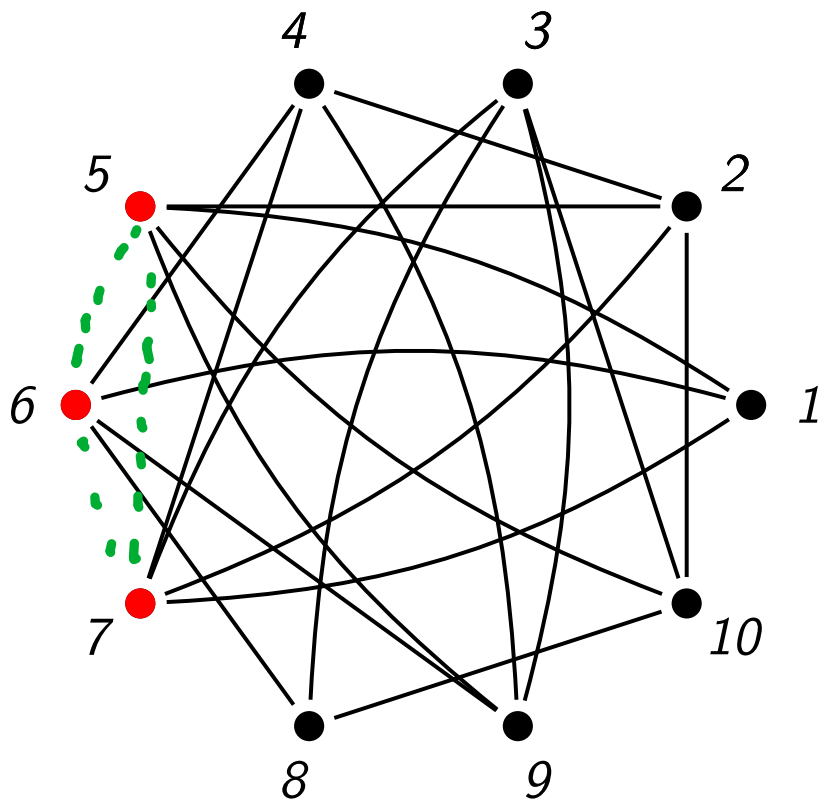
Exemplo



Conjunto Independente de Vértices

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui um conjunto independente de vértices de tamanho pelo menos k?

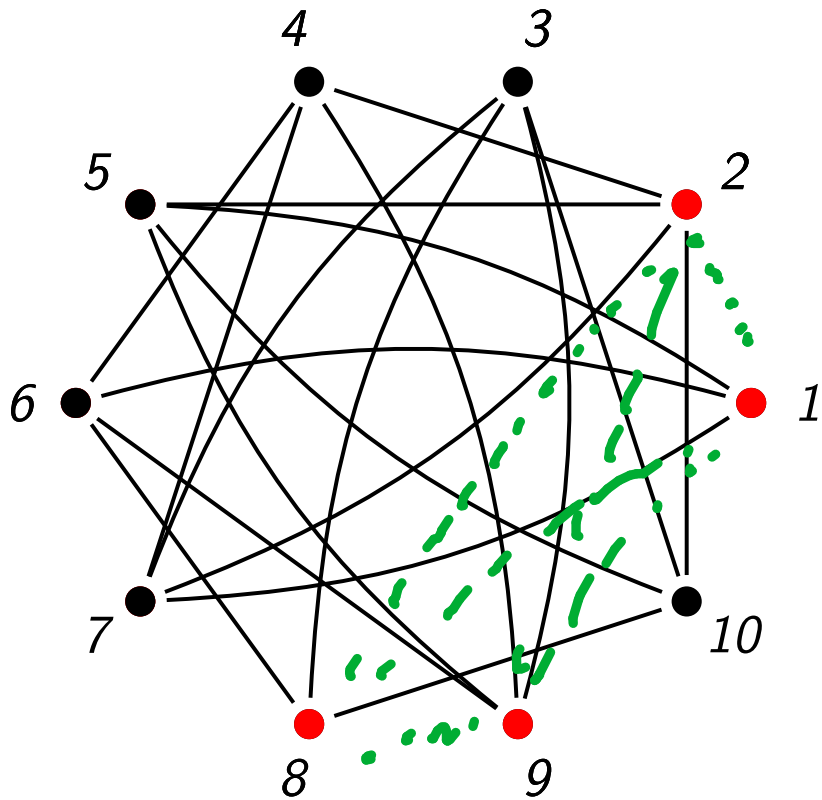
Exemplo



Conjunto Independente de Vértices

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui um conjunto independente de vértices de tamanho pelo menos k?

Exemplo



Cobertura por vértices

DADOS: Um grafo G e um inteiro k

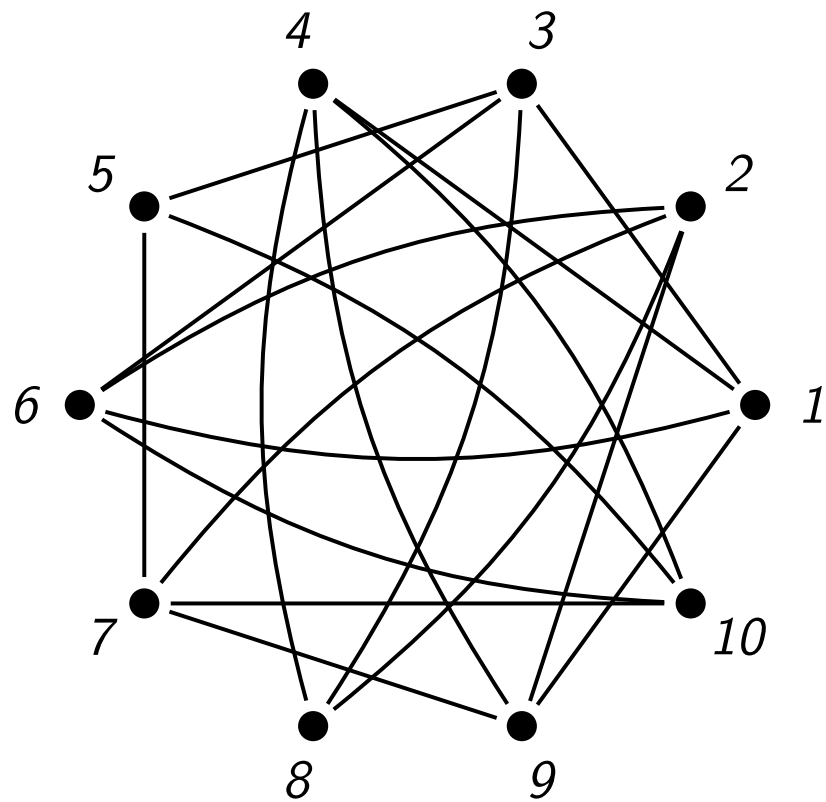
OBJETIVO: G possui uma cobertura por vértices de tamanho no máximo k?

→ CONJ. $S \subseteq V(G)$ T.Q. TODA ARESTA DE G POSSUI VTX EM S

Cobertura por vértices

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui uma cobertura por vértices de tamanho no máximo k?

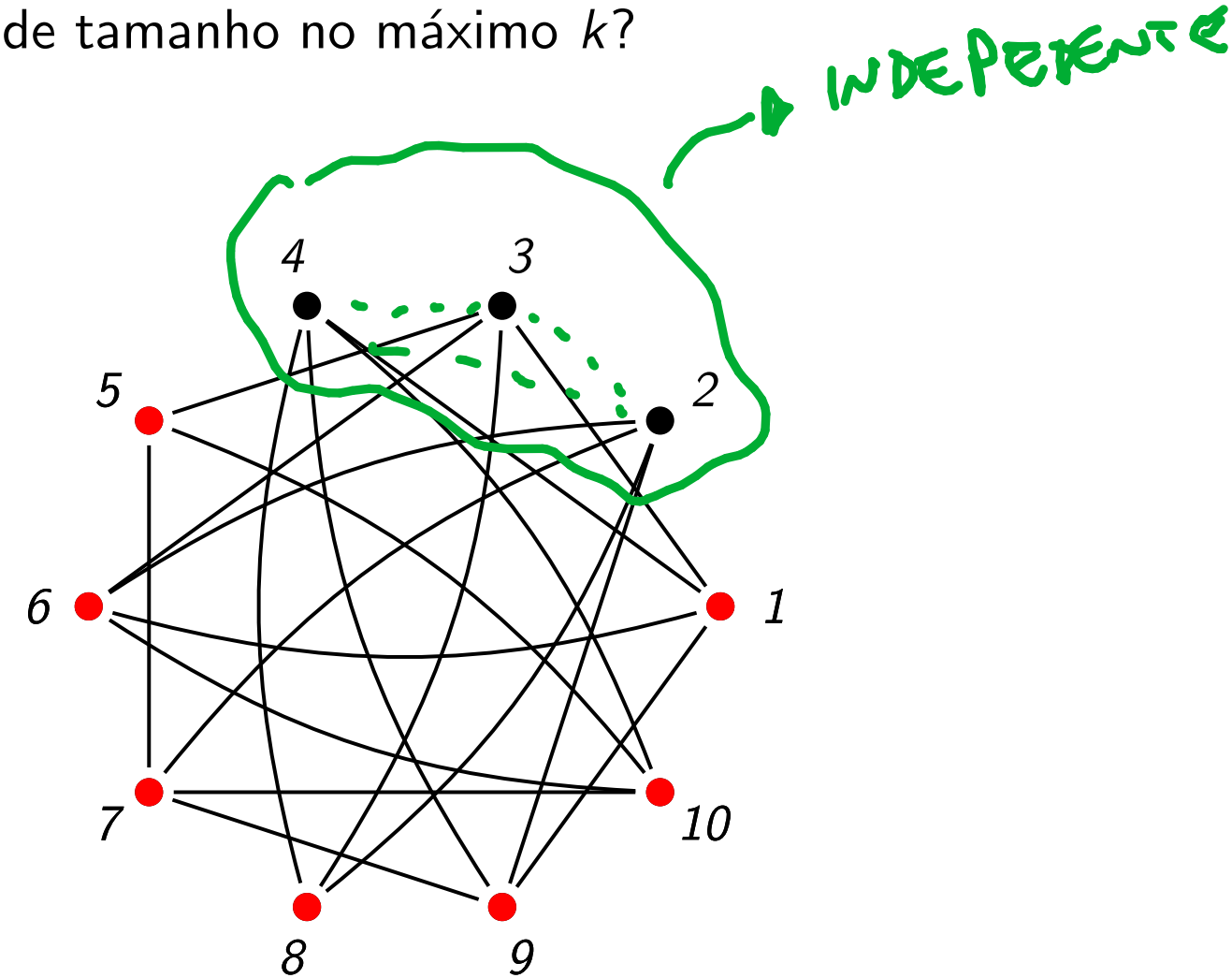
Exemplo



Cobertura por vértices

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui uma cobertura por vértices de tamanho no máximo k?

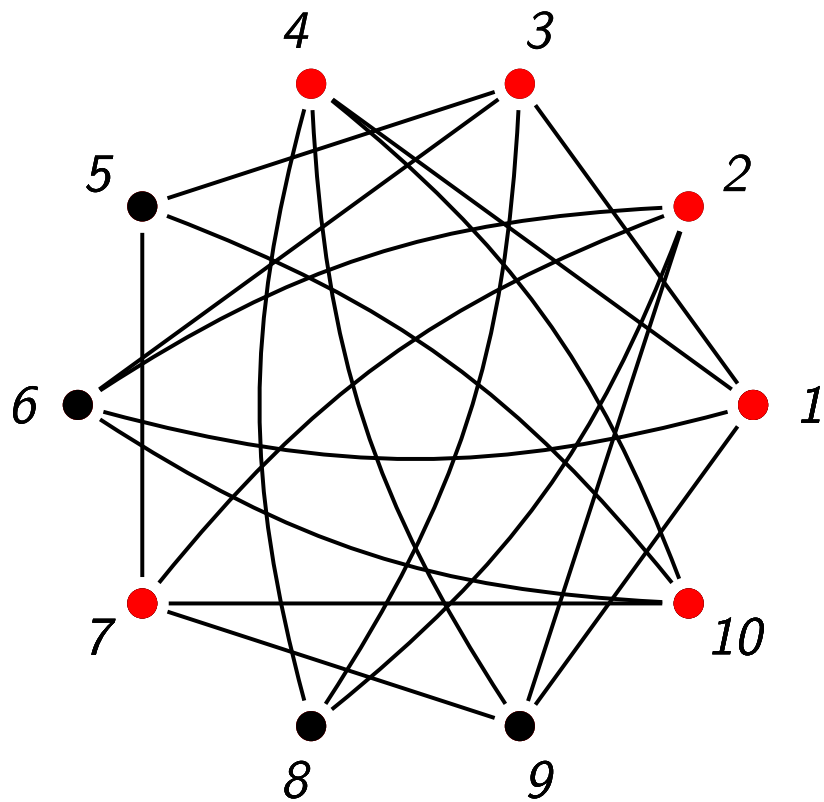
Exemplo



Cobertura por vértices

DADOS: Um grafo G e um inteiro k
OBJETIVO: G possui uma cobertura por vértices de tamanho no máximo k?

Exemplo



Circuito Hamiltoniano

DADOS: Um grafo G

OBJETIVO: G possui um Circuito Hamiltoniano?

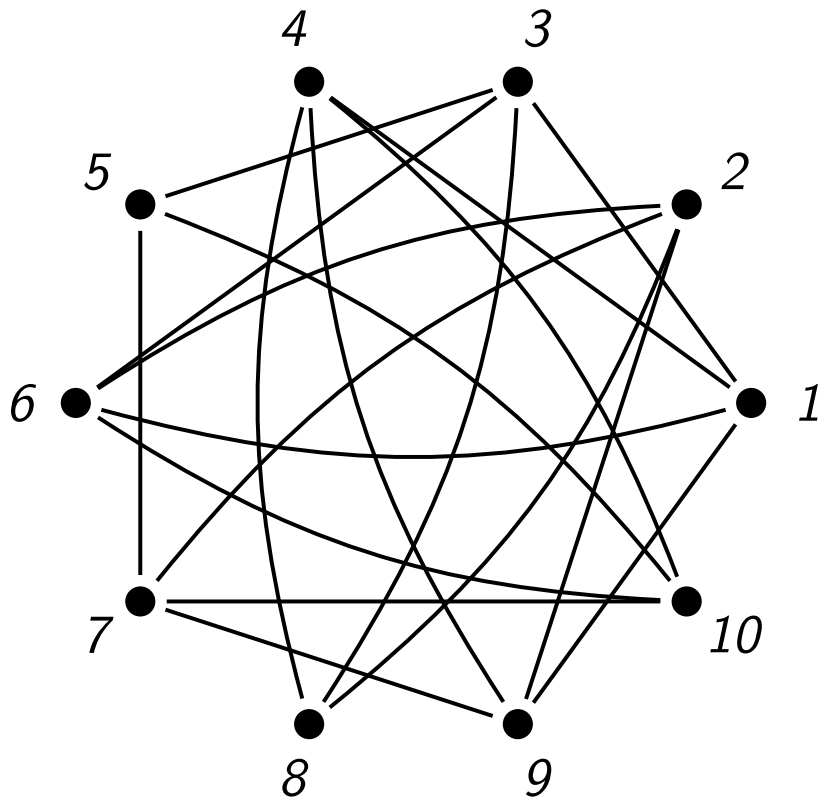
Circuito Hamiltoniano

DADOS: Um grafo G

OBJETIVO: G possui um Circuito Hamiltoniano?

Ciclo

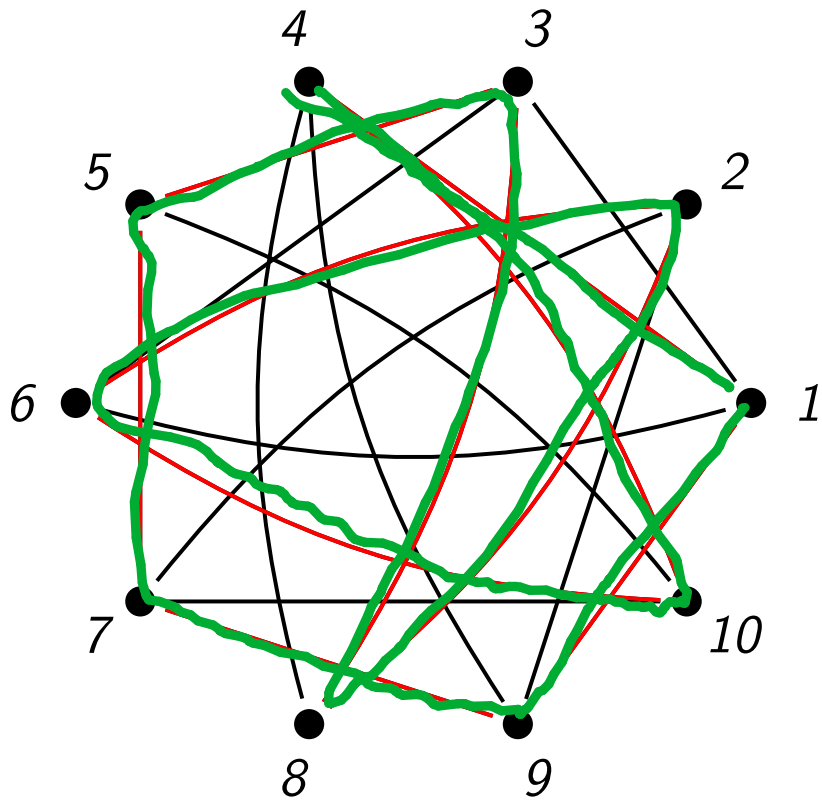
Exemplo



Circuito Hamiltoniano

DADOS: Um grafo G
OBJETIVO: G possui um Circuito Hamiltoniano?

Exemplo



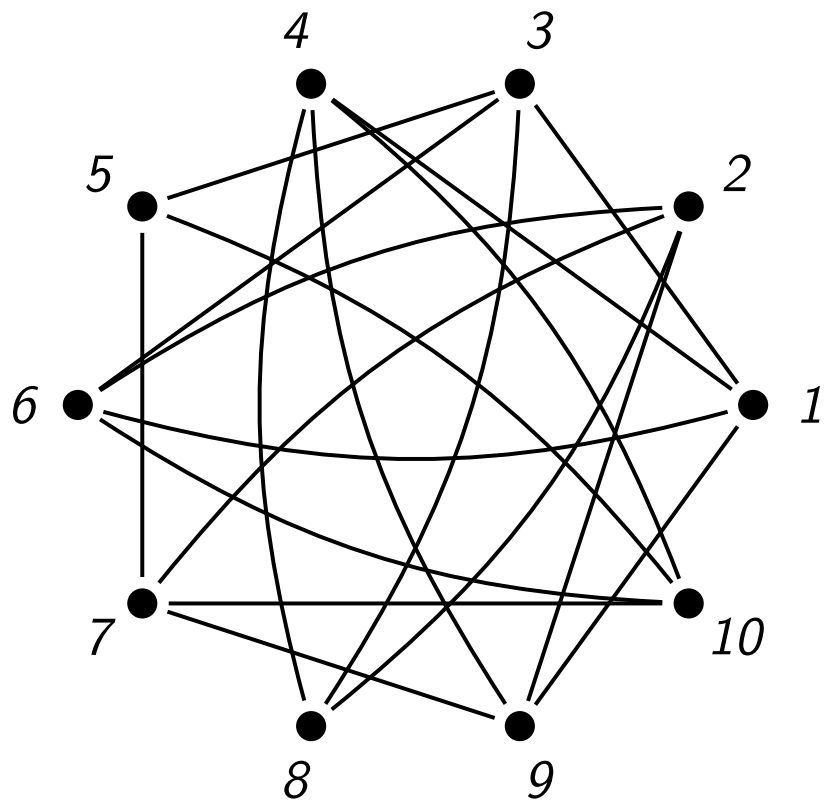
Coloração

DADOS: Um grafo G e um inteiro k
OBJETIVO: G admite uma coloração própria com k cores?

Coloração

DADOS: Um grafo G e um inteiro k
OBJETIVO: G admite uma coloração própria com k cores?

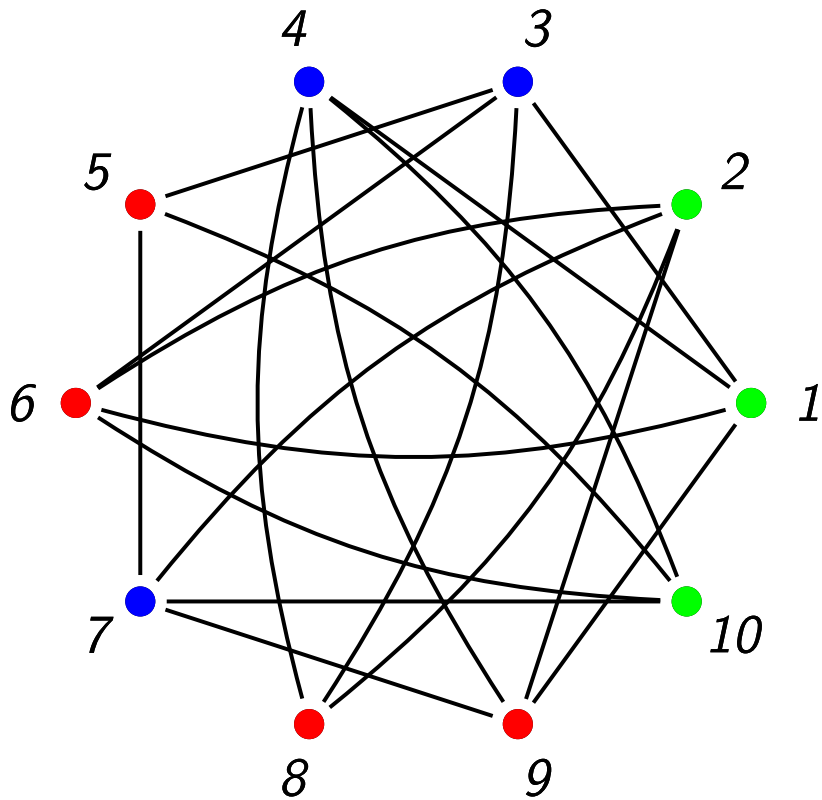
Exemplo



Coloração

DADOS: Um grafo G e um inteiro k
OBJETIVO: G admite uma coloração própria com k cores?

Exemplo



Aresta coloração

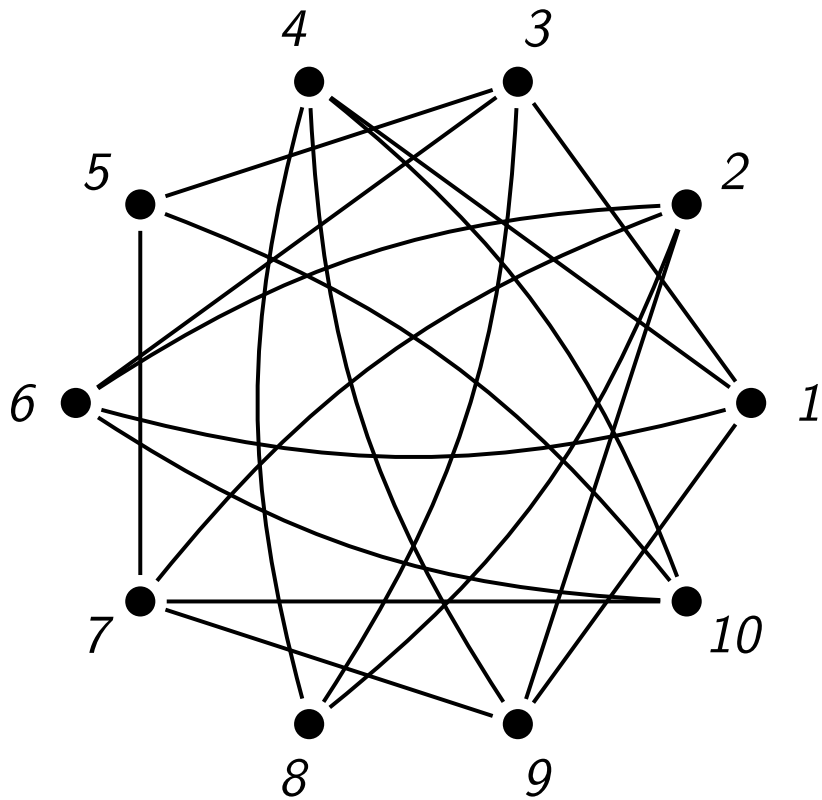
DADOS: Um grafo G

OBJETIVO: G admite uma aresta coloração própria com $\Delta(G)$ cores?

Aresta coloração

DADOS: Um grafo G
OBJETIVO: G admite uma aresta coloração própria com $\Delta(G)$ cores?

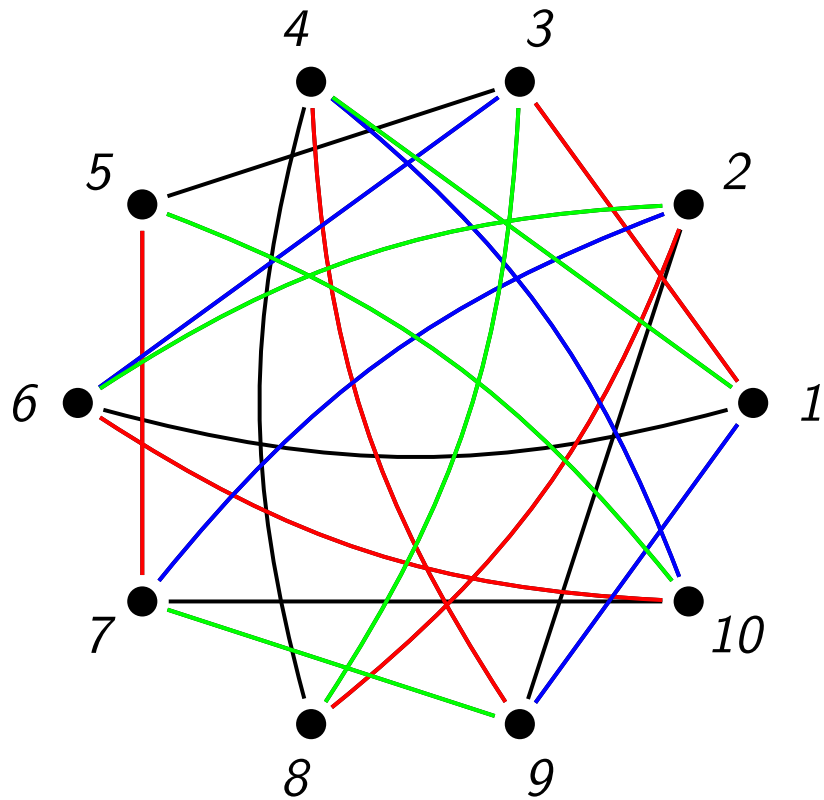
Exemplo



Aresta coloração

DADOS: Um grafo G
OBJETIVO: G admite uma aresta coloração própria com $\Delta(G)$ cores?

Exemplo



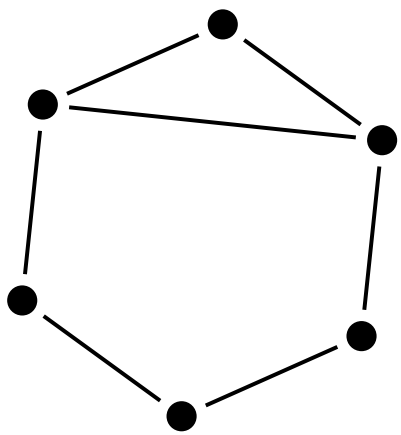
Isomorfismo de subgrafos

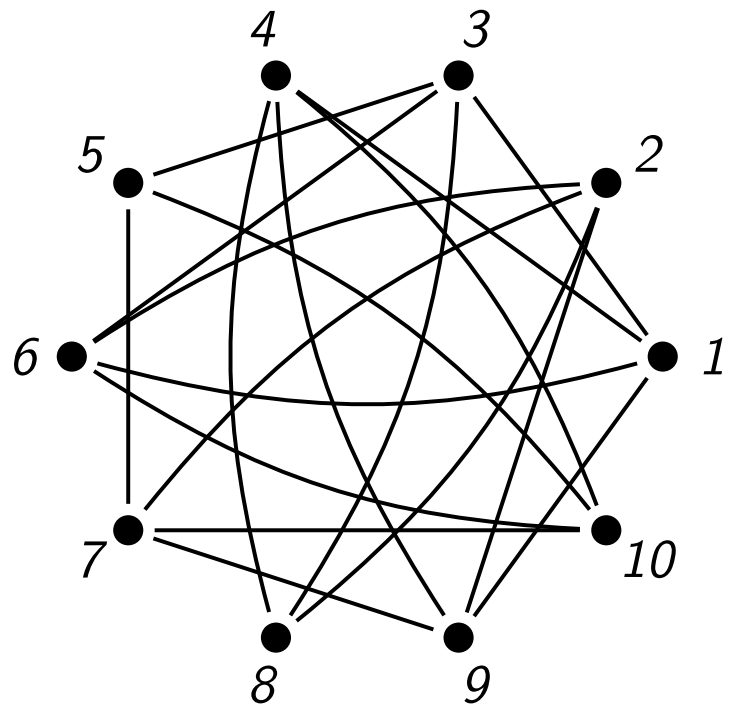
DADOS: Grafo G_1 e G_2
OBJETIVO: G_1 contém um subgrafo isomorfo a G_2?

Isomorfismo de subgrafos

DADOS: Grafo G_1 e G_2
OBJETIVO: G_1 contém um subgrafo isomorfo a G_2?

Exemplo

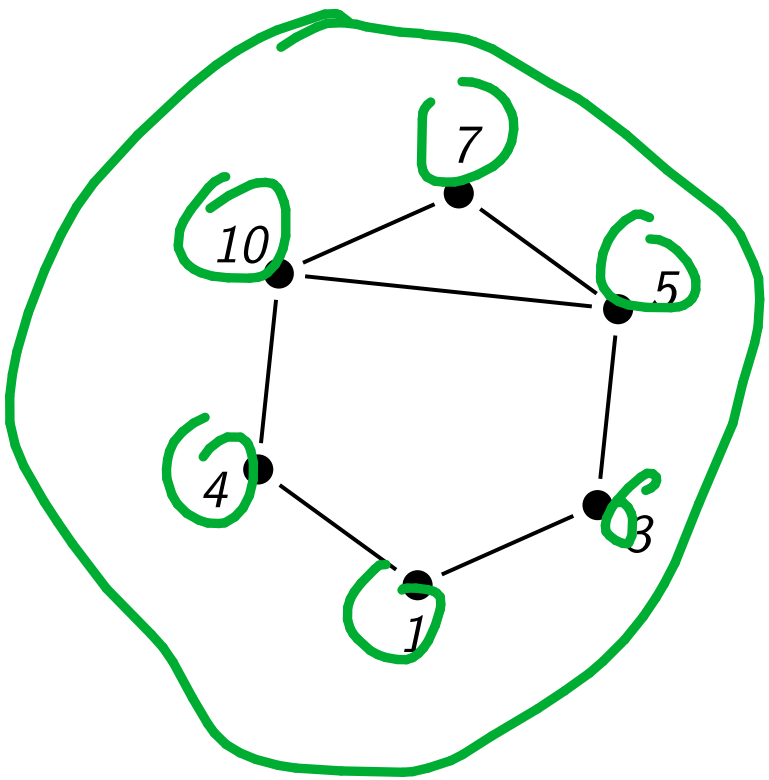


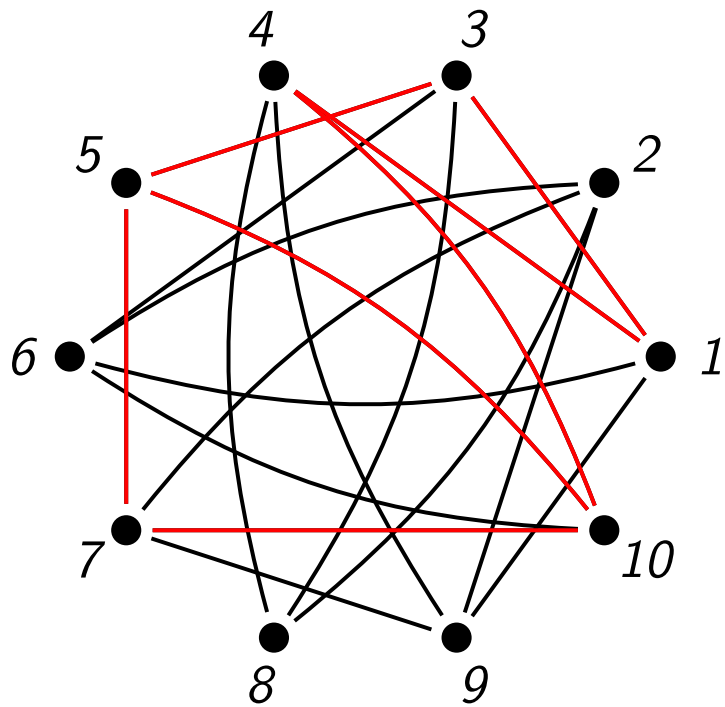


Isomorfismo de subgrafos

DADOS: Grafo G_1 e G_2
OBJETIVO: G_1 contém um subgrafo isomorfo a G_2?

Exemplo





Cobertura por conjuntos

DADOS: Um conjunto U, uma família $\mathcal{S}$ de subconjuntos de U, e um inteiro k

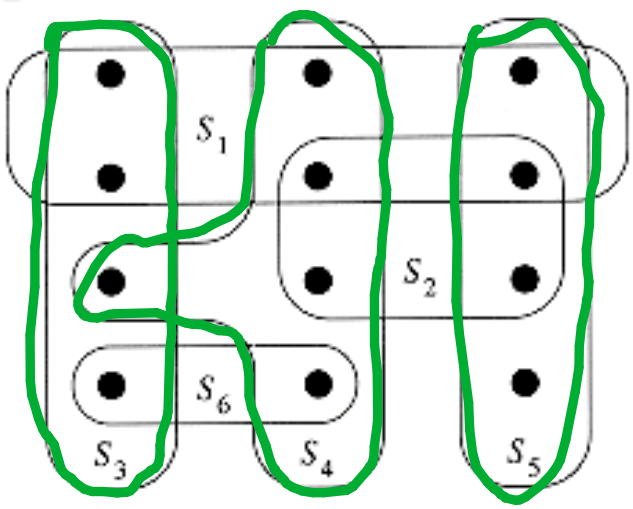
OBJETIVO: existe $\mathcal{S}' \subseteq \mathcal{S}$ tal que $|\mathcal{S}'| \leq k$ e $U \subseteq \bigcup_{S \in \mathcal{S}'} S$?

Cobertura por conjuntos

DADOS: Um conjunto U, uma família $\mathcal{S}$ de subconjuntos de U, e um inteiro k

OBJETIVO: existe $\mathcal{S}' \subseteq \mathcal{S}$ tal que $|\mathcal{S}'| \leq k$ e $U \subseteq \bigcup_{S \in \mathcal{S}'} S$?

Exemplo



Certificados

Um **certificado** para um problema Π é uma justificativa para a resposta SIM.

Um **co-certificado** para um problema Π é uma justificativa para a resposta NÃO.

Fase 1: **exibição** do certificado

Fase 2: **reconhecimento** do certificado

A Classe NP

$$NP = \left\{ \begin{array}{l} \text{problemas de decisão para os quais existe} \\ \text{certificado que pode ser reconhecido por} \\ \text{um algoritmo polinomial} \end{array} \right\}$$

Problemas NP-Completos

Fábio Botler

Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro