

# Complexidade computacional

Classifica os problemas em relação à dificuldade de resolvê-los algoritmicamente.

CLRS 34

# Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma **sequência de símbolos** retirados de algum **alfabeto**.

# Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma **sequência de símbolos** retirados de algum **alfabeto**.

Este alfabeto pode ser, por exemplo, o conjunto de símbolos **ASCII** ou o conjunto  $\{0,1\}$ .

# Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma **sequência de símbolos** retirados de algum **alfabeto**.

Este alfabeto pode ser, por exemplo, o conjunto de símbolos **ASCII** ou o conjunto  $\{0,1\}$ .

Qualquer sequência de elementos de um alfabeto é chamada de uma **palavra**.

# Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma **sequência de símbolos** retirados de algum **alfabeto**.

Este alfabeto pode ser, por exemplo, o conjunto de símbolos **ASCII** ou o conjunto  $\{0,1\}$ .

Qualquer sequência de elementos de um alfabeto é chamada de uma **palavra**.

Não é difícil codificar objetos tais como **racionais**, **vetores**, **matrizes**, **grafos** e **funções** como palavras.

# Palavras

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma **sequência de símbolos** retirados de algum **alfabeto**.

Este alfabeto pode ser, por exemplo, o conjunto de símbolos **ASCII** ou o conjunto  $\{0,1\}$ .

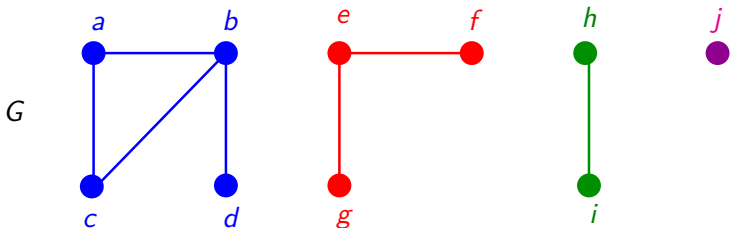
Qualquer sequência de elementos de um alfabeto é chamada de uma **palavra**.

Não é difícil codificar objetos tais como **racionais**, **vetores**, **matrizes**, **grafos** e **funções** como palavras.

O **tamanho** de uma palavra  $w$ , denotado por  $\langle w \rangle$ , é o número de símbolos usados em  $w$ , contando multiplicidades. O tamanho do racional '123/567' é **7**.

# Exemplo 1

Grafo



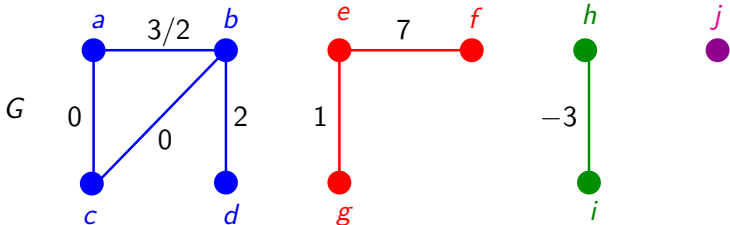
Palavra:

$((\{a, b, c, d, e, f, g, h, i, j\}, \{\{bd\}, \{eg\}, \{ac\}, \{hi\}, \{ab\}, \{ef\}, \{bc\}\}))$

Tamanho da palavra: 59

## Exemplo 2

Grafo com função de peso nas arestas:



Palavra:

$$(((\{bd\}, 2), (\{eg\}, 1), (\{ac\}, 0), (\{hi\}, -3), \\ (\{ab\}, 3/2), (\{ef\}, 7), (\{bc\}, 0)))$$

Tamanho da palavra: 67

# Tamanho de uma palavra

Para os nossos propósitos,  
não há mal em subestimar o tamanho de um objeto.

# Tamanho de uma palavra

Para os nossos propósitos,  
não há mal em subestimar o tamanho de um objeto.

Não é necessário contar rigorosamente os caracteres  
'{', '}', '(', ')', e ',' dos exemplos anteriores.

# Tamanho de uma palavra

Para os nossos propósitos,  
não há mal em subestimar o tamanho de um objeto.

Não é necessário contar rigorosamente os caracteres  
'{', '}', '(', ')', e ',' dos exemplos anteriores.

Tamanho de um inteiro  $p$  é essencialmente  $\lg |p|$ .

Tamanho do racional  $p/q$  é essencialmente  $\lg |p| + \lg |q|$ .

# Tamanho de uma palavra

Para os nossos propósitos,  
não há mal em subestimar o tamanho de um objeto.

Não é necessário contar rigorosamente os caracteres  
'{', '}', '(', ')', e ',', dos exemplos anteriores.

Tamanho de um inteiro  $p$  é essencialmente  $\lg |p|$ .

Tamanho do racional  $p/q$  é essencialmente  $\lg |p| + \lg |q|$ .

Tamanho de um vetor  $A[1..n]$  é  
a soma dos tamanhos de seus componentes

$$\langle A \rangle = \langle A[1] \rangle + \langle A[2] \rangle + \cdots + \langle A[n] \rangle.$$

# Problemas e instâncias

Cada conjunto específico de dados de um problema define uma **instância**.

**Tamanho de uma instância** é o tamanho de uma palavra que representa a instância.

# Problemas e instâncias

Cada conjunto específico de dados de um problema define uma **instância**.

**Tamanho de uma instância** é o tamanho de uma palavra que representa a instância.

Problema que pede uma resposta do tipo **SIM** ou **NÃO** é chamado de **problema de decisão**.

Problema que procura um elemento em um conjunto é um **problema de busca**.

# Problemas e instâncias

Cada conjunto específico de dados de um problema define uma **instância**.

**Tamanho de uma instância** é o tamanho de uma palavra que representa a instância.

Problema que pede uma resposta do tipo **SIM** ou **NÃO** é chamado de **problema de decisão**.

Problema que procura um elemento em um conjunto é um **problema de busca**.

Problema que procura um elemento de um conjunto de soluções viáveis que seja **melhor possível** em relação a algum critério é um **problema de otimização**.

# Máximo divisor comum

**Problema:** Dados dois números inteiros não-negativos  $a$  e  $b$ , determinar  $\text{mdc}(a, b)$ .

**Exemplo:**

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

# Máximo divisor comum

**Problema:** Dados dois números inteiros não-negativos  $a$  e  $b$ , determinar  $\text{mdc}(a, b)$ .

**Exemplo:**

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

**Problema de busca**

**Instância:**  $a$  e  $b$

**Tamanho da instância:**  $\langle a \rangle + \langle b \rangle$ ,

$$\lg a + \lg b.$$

# Máximo divisor comum

**Problema:** Dados dois números inteiros não-negativos  $a$  e  $b$ , determinar  $\text{mdc}(a, b)$ .

**Exemplo:**

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

**Problema de busca**

**Instância:**  $a$  e  $b$

**Tamanho da instância:**  $\langle a \rangle + \langle b \rangle$ ,

$$\lg a + \lg b.$$

Consumo de tempo do algoritmo CAFÉ-COM-LEITE é  $O(b)$ .

Consumo de tempo do algoritmo EUCLIDES é  $O(\lg b)$ .

# Máximo divisor comum (decisão)

**Problema:** Dados números inteiros não-negativos  $a$ ,  $b$  e  $k$ ,  
 $\text{mdc}(a, b) = k$ ?

**Exemplo:**

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

# Máximo divisor comum (decisão)

**Problema:** Dados números inteiros não-negativos  $a$ ,  $b$  e  $k$ ,  
 $\text{mdc}(a, b) = k$ ?

**Exemplo:**

máximo divisor comum de 30 e 24 é 6

máximo divisor comum de 514229 e 317811 é 1

máximo divisor comum de 3267 e 2893 é 11

**Problema de decisão:** resposta SIM ou NÃO

**Instância:**  $a$ ,  $b$ ,  $k$

**Tamanho da instância:**  $\langle a \rangle + \langle b \rangle + \langle k \rangle$ , essencialmente

$$\lg a + \lg b + \lg k$$

# Subsequência comum máxima

**Problema:** Encontrar uma **ssco máxima** de  $X[1..m]$  e  $Y[1..n]$ .

**Exemplos:**  $X = A \text{ B C B D A B}$

$Y = B \text{ D C A B A}$

ssco **máxima** =  $B \text{ C A B}$

# Subsequência comum máxima

**Problema:** Encontrar uma **ssco máxima** de  $X[1..m]$  e  $Y[1..n]$ .

**Exemplos:**  $X = A \text{ B C B D A B}$

$Y = B \text{ D C A B A}$

**ssco máxima** =  $B \text{ C A B}$

**Problema de otimização**

**Instância:**  $X[1..m]$  e  $Y[1..n]$

**Tamanho da instância:**  $\langle X \rangle + \langle Y \rangle$ , essencialmente

$$m + n$$

Consumo de tempo **REC-LCS-LENGTH** é  $\Omega(2^{\min\{m,n\}})$ .

Consumo de tempo **LCS-LENGTH** é  $\Theta(mn)$ .

# Subsequência comum máxima (decisão)

**Problema:**  $X[1..m]$  e  $Y[1..n]$  possuem uma  $\text{ssco} \geq k$ ?

**Exemplo:**  $X = A \text{ B C B D A B}$

$Y = \text{B D C A B A}$

$\text{ssco máxima} = \text{B C A B}$

# Subsequência comum máxima (decisão)

**Problema:**  $X[1..m]$  e  $Y[1..n]$  possuem uma  $ssco \geq k$ ?

**Exemplo:**  $X = A \text{ B C B D A B}$

$Y = B \text{ D C A B A}$

$ssco \text{ máxima} = B \text{ C A B}$

**Problema de decisão:** resposta **SIM** ou **NÃO**

**Instância:**  $X[1..m]$ ,  $Y[1..n]$ ,  $k$

**Tamanho da instância:**  $\langle X \rangle + \langle Y \rangle + \langle k \rangle$ , essencialmente

$$n + m + \lg k$$

# Problema da mochila

**Problema (Knapsack Problem):** Dados  $n$ ,  $w[1..n]$   $v[1..n]$  e  $W$ , encontrar uma **mochila ótima**.

**Exemplo:**  $W = 50$ ,  $n = 4$

|     | 1   | 2   | 3   | 4   |
|-----|-----|-----|-----|-----|
| $w$ | 40  | 30  | 20  | 10  |
| $v$ | 840 | 600 | 400 | 100 |
| $x$ | 0   | 1   | 1   | 0   |

valor = 1000

# Problema da mochila

**Problema (Knapsack Problem):** Dados  $n$ ,  $w[1..n]$ ,  $v[1..n]$  e  $W$ , encontrar uma mochila ótima.

**Exemplo:**  $W = 50$ ,  $n = 4$

|     | 1   | 2   | 3   | 4   |
|-----|-----|-----|-----|-----|
| $w$ | 40  | 30  | 20  | 10  |
| $v$ | 840 | 600 | 400 | 100 |
| $x$ | 0   | 1   | 1   | 0   |

valor = 1000

**Problema de otimização**

**Instância:**  $n$ ,  $w[1..n]$ ,  $v[1..n]$  e  $W$

**Tamanho da instância:**  $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle$ , essencialmente  $\lg n + n \lg W + n \lg V + \lg W$

**Consumo de tempo** MOCHILA-BOOLEANA é  $\Theta(nW)$ .

# Problema da mochila (decisão)

**Problema (Knapsack Problem):** Dados  $n$ ,  $w[1..n]$ ,  $v[1..n]$ ,  $W$  e  $k$ , existe uma mochila de valor  $\geq k$ .

**Exemplo:**  $W = 50$ ,  $n = 4$ ,  $k = 1010$

|     | 1   | 2   | 3   | 4   |
|-----|-----|-----|-----|-----|
| $w$ | 40  | 30  | 20  | 10  |
| $v$ | 840 | 600 | 400 | 100 |
| $x$ | 0   | 1   | 1   | 0   |

valor = 1000

# Problema da mochila (decisão)

**Problema (Knapsack Problem):** Dados  $n$ ,  $w[1..n]$ ,  $v[1..n]$ ,  $W$  e  $k$ , existe uma mochila de valor  $\geq k$ .

**Exemplo:**  $W = 50$ ,  $n = 4$ ,  $k = 1010$

|     | 1   | 2   | 3   | 4   |
|-----|-----|-----|-----|-----|
| $w$ | 40  | 30  | 20  | 10  |
| $v$ | 840 | 600 | 400 | 100 |
| $x$ | 0   | 1   | 1   | 0   |

valor = 1000

**Problema de decisão:** resposta SIM ou NÃO

**Instância:**  $n$ ,  $w[1..n]$ ,  $v[1..n]$ ,  $W$  e  $k$

**Tamanho da instância:**  $\langle n \rangle + \langle w \rangle + \langle v \rangle + \langle W \rangle + \lg k$ ,  
essencialmente  $\lg n + n \lg W + n \lg V + \lg W + \lg k$ .

# Modelo de computação

É uma descrição abstrata e conceitual de um computador que será usado para executar um algoritmo.

# Modelo de computação

É uma descrição abstrata e conceitual de um computador que será usado para executar um algoritmo.

Um modelo de computação especifica as **operações elementares** que um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome.

# Modelo de computação

É uma descrição abstrata e conceitual de um computador que será usado para executar um algoritmo.

Um modelo de computação especifica as **operações elementares** que um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome.

**Operações elementares típicas** são operações aritméticas entre números e comparações.

# Modelo de computação

É uma descrição abstrata e conceitual de um computador que será usado para executar um algoritmo.

Um modelo de computação especifica as **operações elementares** que um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome.

**Operações elementares típicas** são operações aritméticas entre números e comparações.

No **critério uniforme** supõe-se que cada operação elementar consome uma **quantidade de tempo constante**.

# Problemas polinomiais

Análise de um algoritmo em determinado modelo de computação estima seu **consumo de tempo** e **quantidade de espaço** como função do **tamanho da instância do problema**.

# Problemas polinomiais

Análise de um algoritmo em determinado modelo de computação estima seu **consumo de tempo** e **quantidade de espaço** como função do **tamanho da instância do problema**.

**Exemplo:** o consumo de tempo do algoritmo **EUCLIDES** ( $a, b$ ) é expresso como uma função de  $\langle a \rangle + \langle b \rangle$ .

# Problemas polinomiais

Análise de um algoritmo em determinado modelo de computação estima seu **consumo de tempo** e **quantidade de espaço** como função do **tamanho da instância do problema**.

**Exemplo:** o consumo de tempo do algoritmo **EUCLIDES** ( $a, b$ ) é expresso como uma função de  $\langle a \rangle + \langle b \rangle$ .

Um problema é **solúvel em tempo polinomial** se existe um algoritmo que consome tempo  $O(\langle I \rangle^\alpha)$  para resolver o problema, onde  $\alpha$  é uma constante e  $I$  é uma instância do problema.

# Exemplos

- Máximo divisor comum

Tamanho da instância:  $\lg a + \lg b$

Consumo de tempo:

CAFÉ-COM-LEITE é  $O(b)$  (não-polinomial)

EUCLIDES é  $O(\lg b)$  (polinomial)

# Exemplos

- ▶ Máximo divisor comum

Tamanho da instância:  $\lg a + \lg b$

Consumo de tempo:

CAFÉ-COM-LEITE é  $O(b)$  (não-polinomial)

EUCLIDES é  $O(\lg b)$  (polinomial)

- ▶ Subsequência comum máxima

Tamanho da instância:  $n + m$

Consumo de tempo:

REC-LCS-LENGTH é  $\Omega(2^{\min\{m,n\}})$  (exponencial)

LCS-LENGTH é  $\Theta(mn)$  (polinomial)

# Mais exemplos

► Problema da mochila

Tamanho da instância:  $\lg n + n \lg W + n \lg V + \lg W$

Consumo de tempo:

MOCHILA-BOOLEANA é  $\Theta(nW)$  (não-polinomial).

# Mais exemplos

- Problema da mochila

Tamanho da instância:  $\lg n + n \lg W + n \lg V + \lg W$

Consumo de tempo:

MOCHILA-BOOLEANA é  $\Theta(nW)$  (não-polinomial).

- Ordenação de inteiros  $A[1..n]$

Tamanho da instância:  $n \lg M$ , onde

$$M := \max\{|A[1]|, |A[2]|, \dots, |A[n]|\} + 1$$

Consumo de tempo:

MERGE Sort é  $\Theta(n \lg n)$  (polinomial).

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

Dava para ter outra?

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

Dava para ter outra?

Seria interessante que a classe de *algoritmos eficientes* satisfizesse:

- Suponha que um algoritmo  $\mathcal{A}$  usa um algoritmo  $\mathcal{B}$  como subrotina e, contando  $\mathcal{B}$  como passo elementar,  $\mathcal{A}$  é relativamente eficiente. Então, se  $\mathcal{B}$  é eficiente,  $\mathcal{A}$  também é.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

Dava para ter outra?

Seria interessante que a classe de *algoritmos eficientes* satisfizesse:

- ▶ Suponha que um algoritmo  $\mathcal{A}$  usa um algoritmo  $\mathcal{B}$  como subrotina e, contando  $\mathcal{B}$  como passo elementar,  $\mathcal{A}$  é relativamente eficiente. Então, se  $\mathcal{B}$  é eficiente,  $\mathcal{A}$  também é.
- ▶ Algoritmos lineares são eficientes.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

Dava para ter outra?

Seria interessante que a classe de *algoritmos eficientes* satisfizesse:

- ▶ Suponha que um algoritmo  $\mathcal{A}$  usa um algoritmo  $\mathcal{B}$  como subrotina e, contando  $\mathcal{B}$  como passo elementar,  $\mathcal{A}$  é relativamente eficiente. Então, se  $\mathcal{B}$  é eficiente,  $\mathcal{A}$  também é.
- ▶ Algoritmos lineares são eficientes.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

Dava para ter outra?

Seria interessante que a classe de *algoritmos eficientes* satisfizesse:

- ▶ Suponha que um algoritmo  $\mathcal{A}$  usa um algoritmo  $\mathcal{B}$  como subrotina e, contando  $\mathcal{B}$  como passo elementar,  $\mathcal{A}$  é relativamente eficiente. Então, se  $\mathcal{B}$  é eficiente,  $\mathcal{A}$  também é.
- ▶ Algoritmos lineares são eficientes.

Isso leva à classe dos polinomiais como a menor que é *eficiente*.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

Dava para ter outra?

Seria interessante que a classe de *algoritmos eficientes* satisfizesse:

- ▶ Suponha que um algoritmo  $\mathcal{A}$  usa um algoritmo  $\mathcal{B}$  como subrotina e, contando  $\mathcal{B}$  como passo elementar,  $\mathcal{A}$  é relativamente eficiente. Então, se  $\mathcal{B}$  é eficiente,  $\mathcal{A}$  também é.
- ▶ Algoritmos lineares são eficientes.

Isso leva à classe dos polinomiais como a menor que é *eficiente*.

Lucro: fazer transformações polinomiais nos dados “é permitido”.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

A classe de todos os problemas de decisão que podem ser resolvidos por algoritmos polinomiais é denotada por P (classe de complexidade).

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

A classe de todos os problemas de decisão que podem ser resolvidos por algoritmos polinomiais é denotada por P (classe de complexidade).

**Exemplo:** As versões de decisão dos problemas:  
*máximo divisor comum e subsequência comum máxima*  
estão em P.

# Classe P

Algoritmos polinomiais são uma abstração de algoritmos eficientes.

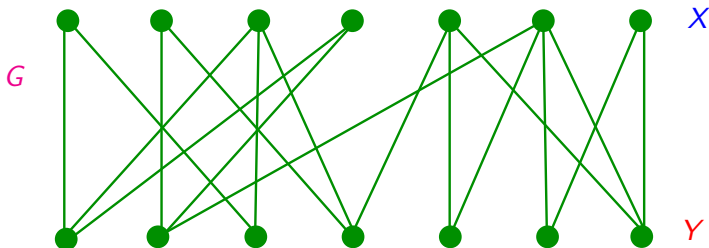
A classe de todos os problemas de decisão que podem ser resolvidos por algoritmos polinomiais é denotada por  $P$  (classe de complexidade).

**Exemplo:** As versões de decisão dos problemas:  
*máximo divisor comum e subsequência comum máxima*  
estão em  $P$ .

Para muitos problemas, não se conhece algoritmo essencialmente melhor que “testar todas as possibilidades”. Em geral, isso não está em  $P$ .

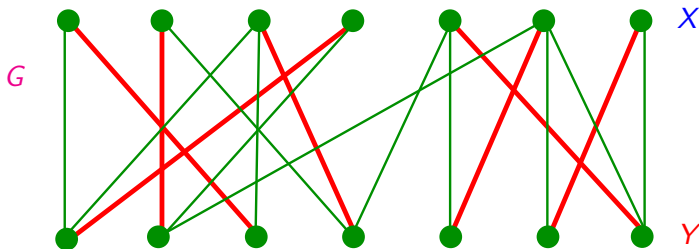
# Emparelhamentos

**Problema:** Dado um grafo bipartido, encontrar um emparelhamento perfeito.



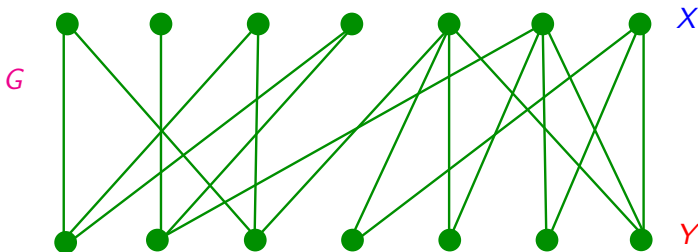
# Emparelhamentos

**Problema:** Dado um grafo bipartido, encontrar um emparelhamento perfeito.



# Emparelhamentos

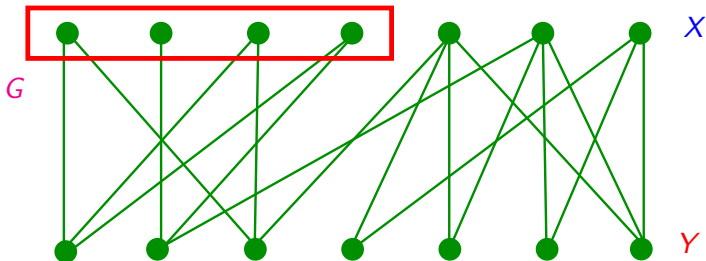
**Problema:** Dado um grafo bipartido, encontrar um emparelhamento perfeito.



**NÃO** existe! Certificado?

# Emparelhamentos

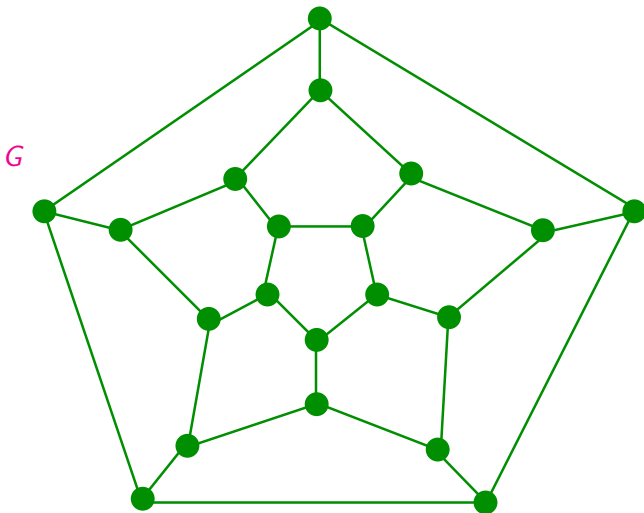
**Problema:** Dado um grafo bipartido, encontrar um emparelhamento bipartido.



**NÃO** existe! Certificado:  $S \subseteq X$  tal que  $|S| > |\text{vizinhos}(S)|$ .

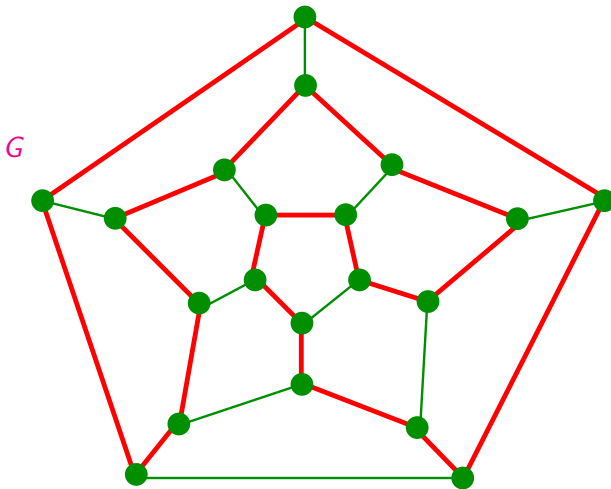
# Grafos hamiltonianos

**Problema:** Dado um grafo, encontrar um ciclo hamiltoniano.



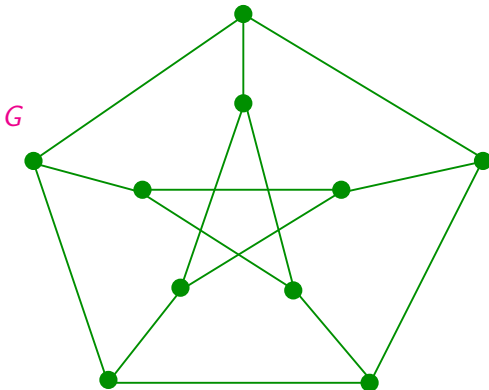
# Grafos hamiltonianos

**Problema:** Dado um grafo, encontrar um ciclo hamiltoniano.



# Grafos hamiltonianos

**Problema:** Dado um grafo, encontrar um ciclo hamiltoniano.



**NÃO** existe! Certificado? Hmmm ...

# Verificador polinomial para sim

Um **verificador polinomial para a resposta SIM** a um problema  $\Pi$  é um algoritmo polinomial **ALG** que **recebe**

*uma instância  $I$  de  $\Pi$  e um objeto  $C$ ,  
tal que  $\langle C \rangle$  é  $O(\langle I \rangle^\alpha)$  para alguma constante  $\alpha$*

# Verificador polinomial para sim

Um **verificador polinomial para a resposta SIM** a um problema  $\Pi$  é um algoritmo polinomial **ALG** que **recebe**

*uma instância  $I$  de  $\Pi$  e um objeto  $C$ ,  
tal que  $\langle C \rangle$  é  $O(\langle I \rangle^\alpha)$  para alguma constante  $\alpha$*

e **devolve**

**SIM** para algum  $C$  se a resposta a  $\Pi(I)$  é **SIM**;  
**NÃO** para todo  $C$  se a resposta a  $\Pi(I)$  é **NÃO**.

# Verificador polinomial para sim

Um **verificador polinomial para a resposta SIM** a um problema  $\Pi$  é um algoritmo polinomial **ALG** que **recebe**

*uma instância  $I$  de  $\Pi$  e um objeto  $C$ ,  
tal que  $\langle C \rangle$  é  $O(\langle I \rangle^\alpha)$  para alguma constante  $\alpha$*

e **devolve**

**SIM** para algum  $C$  se a resposta a  $\Pi(I)$  é **SIM**;  
**NÃO** para todo  $C$  se a resposta a  $\Pi(I)$  é **NÃO**.

No caso de resposta **SIM**, o objeto  $C$  é dito um **certificado polinomial** ou **certificado curto** da resposta **SIM** a  $\Pi(I)$ .

# Exemplos

- ▶ Se  $G$  é hamiltoniano, então um ciclo hamiltoniano de  $G$  é um certificado polinomial:

*dados um grafo  $G$  e  $C$ , pode-se verificar em tempo  $O(\langle G \rangle)$  se  $C$  é um ciclo hamiltoniano.*

# Exemplos

- ▶ Se  $G$  é hamiltoniano, então um ciclo hamiltoniano de  $G$  é um certificado polinomial:

*dados um grafo  $G$  e  $C$ , pode-se verificar em tempo  $O(\langle G \rangle)$  se  $C$  é um ciclo hamiltoniano.*

- ▶ se  $X[1..m]$  e  $Y[1..n]$  possuem uma ssc  $\geq k$ , então uma subsequência comum  $Z[1..k]$  é um certificado polinomial:

*dados  $X[1..m]$ ,  $Y[1..n]$  e  $Z[1..k]$ , pode-se verificar em tempo  $O(m+n)$  se  $Z$  é ssc de  $X$  e  $Y$ .*

# Exemplos

- ▶ Se  $G$  é hamiltoniano, então um ciclo hamiltoniano de  $G$  é um certificado polinomial:

*dados um grafo  $G$  e  $C$ , pode-se verificar em tempo  $O(\langle G \rangle)$  se  $C$  é um ciclo hamiltoniano.*

- ▶ se  $X[1..m]$  e  $Y[1..n]$  possuem uma ssco  $\geq k$ , então uma subsequência comum  $Z[1..k]$  é um certificado polinomial:

*dados  $X[1..m]$ ,  $Y[1..n]$  e  $Z[1..k]$ , pode-se verificar em tempo  $O(m+n)$  se  $Z$  é ssco de  $X$  e  $Y$ .*

- ▶ se  $n$  é um número composto, então um divisor próprio  $d > 1$  de  $n$  é um certificado polinomial.

# Verificador polinomial para não

Um verificador polinomial para a resposta NÃO de um problema  $\Pi$  é um algoritmo polinomial ALG que recebe

*uma instância  $I$  de  $\Pi$  e um objeto  $C$ ,  
tal que  $\langle C \rangle$  é  $O(\langle I \rangle^\alpha)$  para alguma constante  $\alpha$*

e devolve

*SIM para algum  $C$  se a resposta a  $\Pi(I)$  é NÃO;  
NÃO para todo  $C$  se a resposta a  $\Pi(I)$  é SIM.*

No caso de resposta SIM, o objeto  $C$  é dito um certificado polinomial ou certificado curto da resposta NÃO a  $\Pi(I)$ .

# Classe NP

Formada pelos problemas de decisão que possuem um verificador polinomial para a resposta SIM.

# Classe NP

Formada pelos problemas de decisão que possuem um verificador polinomial para a resposta SIM.

Em outras palavras, a classe NP é formada pelos problemas de decisão  $\Pi$  para os quais existe um problema  $\Pi'$  em P e uma função polinomial  $p(n)$  tais que, para cada instância  $I$  do problema  $\Pi$ , existe um objeto  $C$  com  $\langle C \rangle \leq p(\langle I \rangle)$  tal que

- a resposta a  $\Pi(I)$  é SIM se e somente se*
- a resposta a  $\Pi'(I, C)$  é SIM.*

# Classe NP

Formada pelos problemas de decisão que possuem um verificador polinomial para a resposta SIM.

Em outras palavras, a classe NP é formada pelos problemas de decisão  $\Pi$  para os quais existe um problema  $\Pi'$  em P e uma função polinomial  $p(n)$  tais que, para cada instância  $I$  do problema  $\Pi$ , existe um objeto  $C$  com  $\langle C \rangle \leq p(\langle I \rangle)$  tal que

- a resposta a  $\Pi(I)$  é SIM se e somente se
- a resposta a  $\Pi'(I, C)$  é SIM.

O objeto  $C$  é dito um certificado polinomial ou certificado curto da resposta SIM a  $\Pi(I)$ .

# Exemplos

Problemas de decisão com certificado polinomial para SIM:

- ▶ existe subsequência crescente  $\geq k$ ?
- ▶ existe subcoleção disjunta  $\geq k$  de intervalos?
- ▶ existe mochila de valor  $\geq k$ ?
- ▶ existe subsequência comum  $\geq k$ ?
- ▶ grafo tem ciclo de comprimento  $\geq k$ ?
- ▶ grafo tem ciclo hamiltoniano?
- ▶ grafo tem emparelhamento (casamento) perfeito?

Todos esses problemas estão em NP.

$$\mathbf{P} \subseteq \mathbf{NP}$$

Prova:

Se  $\Pi$  é um problema em  $\mathbf{P}$ , então pode-se tomar a sequência de instruções realizadas por um algoritmo polinomial para resolver  $\Pi(I)$  como certificado polinomial da resposta  $\mathbf{SIM}$  a  $\Pi(I)$ .

$$\mathbf{P} \subseteq \mathbf{NP}$$

### Prova:

Se  $\Pi$  é um problema em  $\mathbf{P}$ , então pode-se tomar a sequência de instruções realizadas por um algoritmo polinomial para resolver  $\Pi(I)$  como certificado polinomial da resposta **SIM** a  $\Pi(I)$ .

### Outra prova:

Pode-se construir um verificador polinomial para a resposta **SIM** a  $\Pi$  **utilizando-se** um algoritmo polinomial para  $\Pi$  como subrotina e **ignorando-se** o certificado **C**.

# $P \neq NP?$

É crença de muitos que a classe **NP** é maior que a classe **P**, ainda que isso **não tenha sido provado** até agora.

# $P \neq NP?$

É crença de muitos que a classe  $NP$  é maior que a classe  $P$ , ainda que isso **não tenha sido provado** até agora.

Este é o intrigante problema matemático conhecido pelo rótulo  
“ $P \neq NP?$ ”

# $P \neq NP?$

É crença de muitos que a classe **NP** é maior que a classe **P**, ainda que isso **não tenha sido provado** até agora.

Este é o intrigante problema matemático conhecido pelo rótulo  
“**P**  $\neq$  **NP**?”

**Não confunda NP com “não-polinomial”.**

# Classe co-NP

A classe co-NP é definida trocando-se SIM por NÃO na definição de NP.

# Classe co-NP

A classe co-NP é definida trocando-se SIM por NÃO na definição de NP.

Um problema de decisão  $\Pi$  está em co-NP se admite um certificado polinomial para a resposta NÃO.

# Classe co-NP

A classe co-NP é definida trocando-se SIM por NÃO na definição de NP.

Um problema de decisão  $\Pi$  está em co-NP se admite um certificado polinomial para a resposta NÃO.

Os problemas em  $NP \cap co-NP$  admitem certificados polinomiais para as respostas SIM e NÃO.

# Classe co-NP

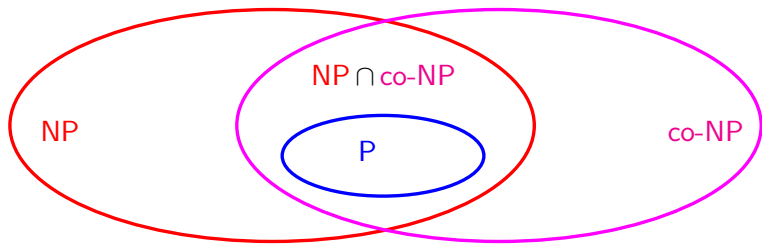
A classe co-NP é definida trocando-se SIM por NÃO na definição de NP.

Um problema de decisão  $\Pi$  está em co-NP se admite um certificado polinomial para a resposta NÃO.

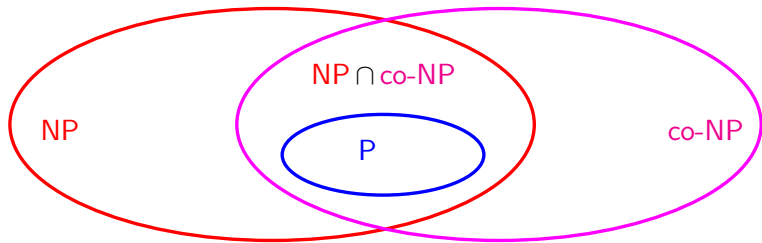
Os problemas em  $NP \cap co-NP$  admitem certificados polinomiais para as respostas SIM e NÃO.

Em particular,  $P \subseteq NP \cap co-NP$ .

# P, NP e co-NP



# P, NP e co-NP



$P \neq NP?$

$NP \cap co-NP \neq P?$

$NP \neq co-NP?$