NP-complexidade

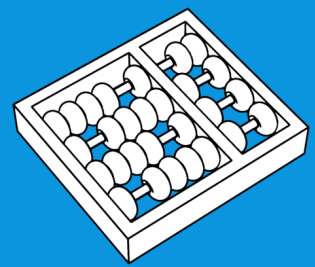
MO417 - Complexidade de Algoritmos I

06/24

25

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP

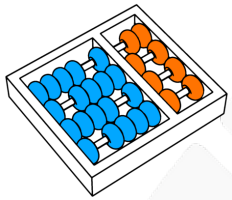


"NP captures vast domains of computational, scientific, and mathematical endeavors, and seems to roughly delimit what mathematicians and scientists have been aspiring to compute feasibly."[1]

Christos Harilaos Papadimitriou.

[1] Christos H. Papadimitriou, *"NP-completeness: A retrospective"*, International Colloquium on Automata, Languages, and Programming, 1997, p. 2.

Introdução

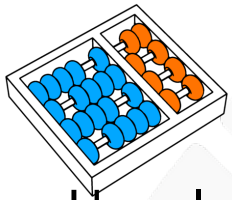


Algoritmos eficientes e ineficientes

- Para vários problemas, vimos algoritmos rápidos:
  - Ordenação: $O(n \log n)$.
  - Multiplicação de matrizes: $O(n^{2,72})$.
  - Caminho mais curto*: $O(mE)$.
  - Circuito euleriano: $O(V + E)$.
- Para outros, só são conhecidos algoritmos lentos:
  - Problema da mochila: $O(2^n)$.
  - Caminho mais longo*: $O(m!2^m E)$.
  - Circuito hamiltoniano: $O(2^V)$.

Como identificar algoritmos **RÁPIDOS**?

*m é o tamanho do caminho



Algoritmos de tempo polinomial

Um algoritmo é **POLINOMIAL** se o tempo de execução for limitado por $O(n^k)$ para alguma constante k.

- Nesse caso, dizemos que ele é um algoritmo **EFICIENTE**.
- Não necessariamente é rápido na prática.
- Mas exclui muitos dos algoritmos considerados lentos.

indebted to many people, at the Symposium and at the National Bureau of Standards, who have taken an interest in the matching problem. There has been much animated discussion on possible versions of an algorithm.

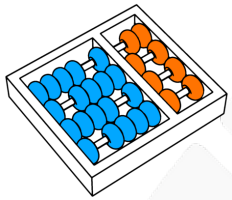
2. Digression. An explanation is due on the use of the words "efficient algorithm." First, what I present is a conceptual description of an algorithm and not a particular formalized algorithm or "code."

For practical purposes computational details are vital. However, my is an efficient quate in opera- the sense that

PATHS, TREES, AND FLOWERS

JACK EDMONDS

1. Introduction. A *graph* G for purposes here is a finite set of elem called *vertices* and a finite set of elements called *edges* such that each e *meets* exactly two vertices, called the *end-points* of the edge. An edge is s to *join* its end-points.



Organizando os problemas

Por que se preocupar com isso?

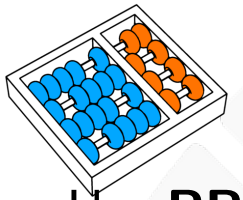
- Desconhecemos algoritmos rápidos para vários problemas.
- Acreditamos que não há algoritmos eficientes para eles.
- Queremos saber quais deles têm **ALGORITMOS POLINOMIAIS**!

Antes vamos discutir:

1. Como representar problemas?
2. Como comparar problemas?

Respondemos essas perguntas da seguinte forma:

1. Representamos problemas como **LINGUAGENS FORMAIS**.
2. Comparamos problemas com um tipo de **REDUÇÃO**.



Problemas de decisão

Um **PROBLEMA DE DECISÃO** é um problema em que a resposta de cada elemento da entrada é SIM ou NAO.

São problemas de decisão:

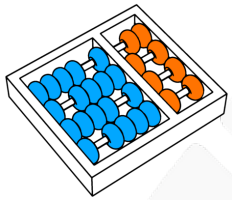
- Dado um número m, m é primo?
- Dadas as posições das peças em um tabuleiro de xadrez, o rei está em xeque?

Não são problemas de decisão:

- Soma, ordenação, caminho mínimo etc.

Por que estudar problemas de decisão?

- É mais simples estudá-los do que problemas em geral.
- Várias situações são postas como problemas de decisão.
- Às vezes, decidir se existe alguma solução para um problema em geral é tão difícil quanto encontrá-la.



Versão de decisão de problema de busca

Problema (Busca do ciclo hamiltoniano)

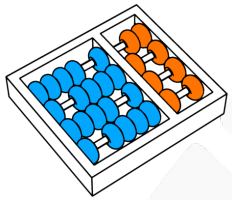
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Um ciclo em G que percorre todos vértices.*

Problema (Decisão do ciclo hamiltoniano)

- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe ciclo em G que percorre todos vértices.*

Suponha que sabemos **DECIDIR** em tempo polinomial

- Como **ENCONTRAR** em tempo polinomial?
- Descubra uma aresta por vez (exercício).



Versão de decisão de problema de otimização

Problema (Caixeiro Viajante)

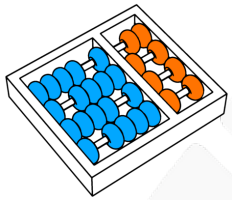
- ***Entrada:*** *Um grafo ponderado G.*
- ***Saída:*** *Um ciclo hamiltoniano com peso mínimo.*

Problema (Versão de decisão)

- ***Entrada:*** *Um grafo ponderado G, um parâmetro k.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G com peso no máximo k.*

Suponha que sabemos **DECIDIR** em tempo polinomial

- Como determinar o **VALOR ÓTIMO** em tempo polinomial?
- Faça uma busca binária (exercício).



Olhando para frente

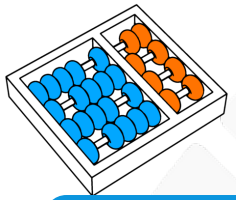
Vamos classificar **PROBLEMAS DE DECISÃO** nas classes:

P: podem ser resolvidos em tempo polinomial

NP: têm soluções curtas que podem ser verificadas em tempo polinomial

NP-difícil: pelo menos tão difíceis quanto quaisquer outros problemas em NP

NP-completo: são NP e NP-difícil



Exemplos de problemas em P

Problema (Soma de elemento)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $n \in S$ tal que $n = \sum_{m \in S \setminus \{n\}} m$.*

Problema (Conexidade)

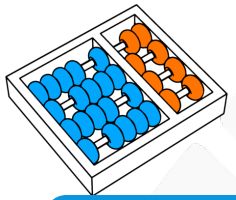
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se G é conexo.*

Problema (Caminho mínimo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no máximo k.*

Problema (4-Coloração)

- ***Entrada:*** *Um mapa de regiões.*
- ***Saída:*** *Decidir se é possível colorir as regiões com até 4 cores de forma que regiões adjacentes tenham cores distintas.*



Exemplos de problemas em NP-completo

Problema (Bipartição)

- ***Entrada:*** *Um conjunto de números S.*
- ***Saída:*** *Decidir se existe $N \subseteq S$ tal que $\sum_{m \in N} m = \sum_{m \in S \setminus N} m$.*

Problema (Ciclo hamiltoniano)

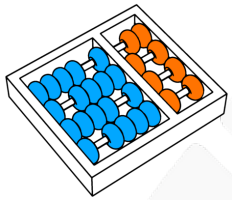
- ***Entrada:*** *Um grafo G.*
- ***Saída:*** *Decidir se existe um ciclo hamiltoniano em G.*

Problema (Caminho mais longo)

- ***Entrada:*** *Um grafo ponderado G, dois vértices s e t e um inteiro k.*
- ***Saída:*** *Decidir se existe um caminho de s até t de peso no mínimo k.*

Problema (3-Coloração)

- ***Entrada:*** *Um mapa de regiões.*
- ***Saída:*** *Decidir se é possível colorir as regiões com até 3 cores de forma que regiões adjacentes tenham cores distintas.*

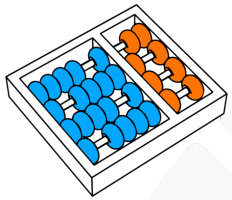


Uma motivação prática

Vários problemas importantes são NP-difíceis:

- Roteamento de veículos de cadeias de distribuição.
- Atribuição de frequências em telefonia celular.
- Empacotamento de objetos em contêineres.
- Escalonamento de funcionários em turnos de trabalho.
- Escalonamento de tarefas em computadores.
- Classificação de objetos.
- Coloração de mapas.
- Projetos de redes de computadores.
- Otimização de código.
- Muitos outros...

É **IMPRESCINDÍVEL** saber se nosso problema é NP-difícil!



Algumas sutilezas

Se sabe:

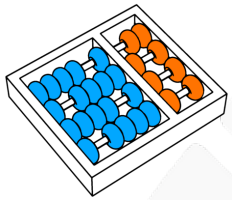
- Que o problema do caminho mais curto está em P.
- Que o problema da 4-coloração está em P.

Ainda não se sabe:

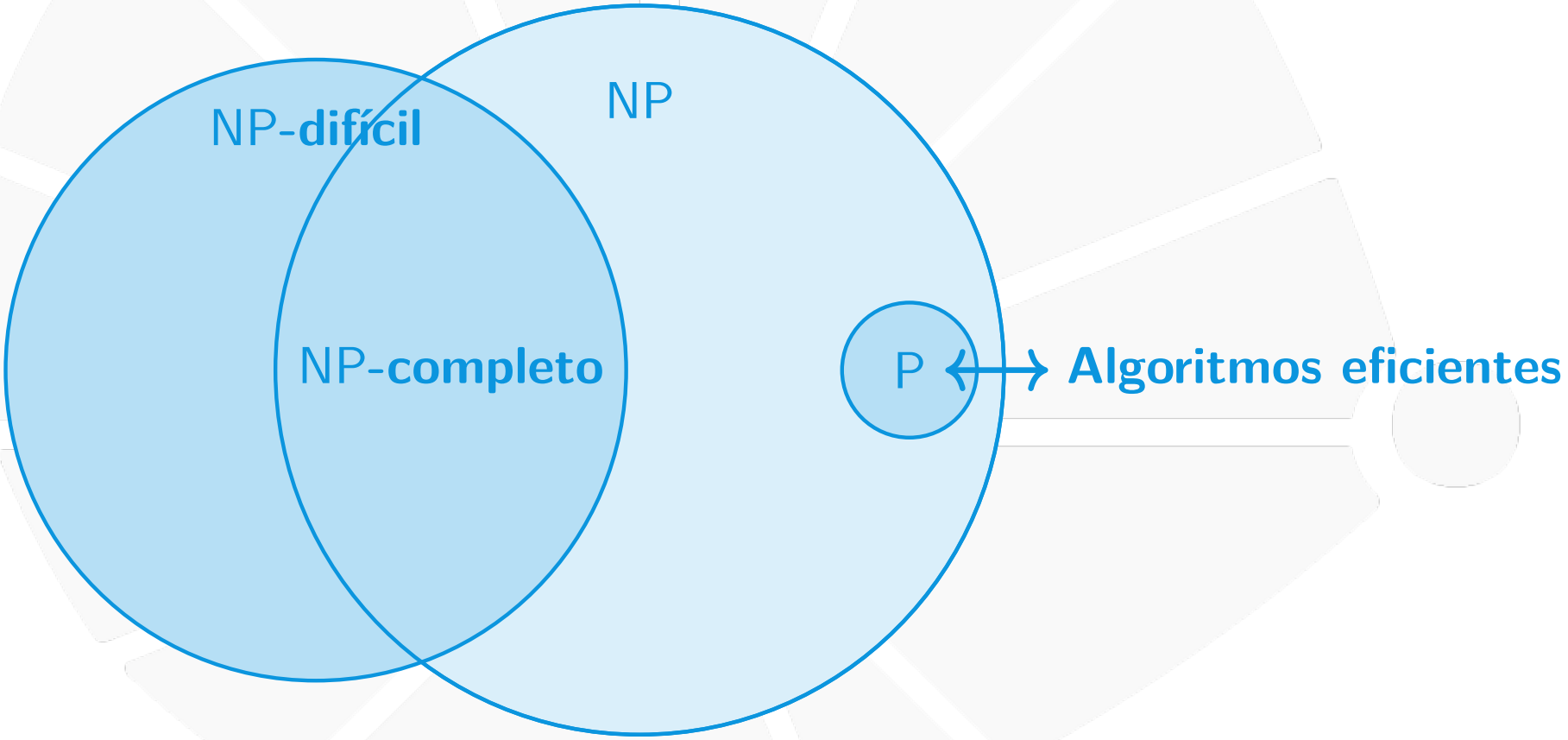
- O problema do caminho mais **LONGO** está em P?
- O problema da **3**-coloração está em P?

Parecem perguntas bem diferentes:

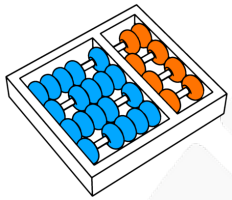
- Mas as duas têm respostas idênticas.
- Responder SIM é o mesmo que dizer P $=$ NP.
- Contudo, conjecturamos que a resposta seja NAO!



Possível configuração of classes



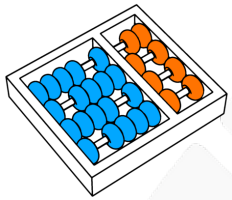
No restante do curso, procuraremos entender essa figura!



Evidências para essa configuração

Por que acreditamos que **NÃO HÁ** algoritmo rápido para problemas NP-difíceis?

- Demonstrou-se que um problema é NP-completo pela primeira vez na década de 1970 (Cook-Levin).
- Anteriormente, vários desses problemas já eram estudados.
- Desde então, mostrou-se que inúmeros problemas importantes também são NP-completos ou NP-difíceis.
- Vários pesquisadores estudaram seus problemas NP-completos preferidos, mas ninguém descobriu qualquer algoritmo polinomial.
- Basta que um problema NP-difícil tenha algoritmo de tempo polinomial para que **TODOS** problemas em NP tenham algoritmos de tempo polinomial.



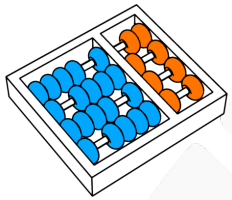
O que fazer se um problema for NP-difícil?

Se sabemos que um problema é NP-difícil, podemos concentrar os recursos em busca de:

- Algoritmos para **INSTÂNCIAS PEQUENAS**.
- Algoritmos que obtêm **SOLUÇÕES APROXIMADAS**.
- Algoritmos eficientes exatos, mas para **CASOS PARTICULARES**.
- Algoritmos e métodos **HEURÍSTICOS**.
- etc.

Antes, queremos identificar que problemas são NP-difíceis.

Classes de problemas



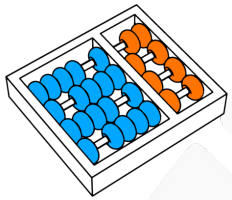
Linguagens formais

Alguns termos:

- Um alfabeto Σ é um conjunto finito de símbolos.
- Uma palavra é uma sequência finita de símbolos de Σ.
- O conjunto de todas as palavras é denotado por Σ^*, que inclui a sequência vazia, ε.

Uma **LINGUAGEM FORMAL** sobre Σ é um subconjunto $L \subseteq \Sigma^*$.

- A linguagem vazia é $L = \emptyset$.
- Uma finita é $F = \{banana, uva, pera\}$ sobre $\Sigma = \{a, b, \ldots, z\}$.
- Uma infinita é $P = \{2, 3, 7, 11, \ldots\}$ sobre $\Sigma = \{0, 1, \ldots, 9\}$.



Representando problemas de decisão

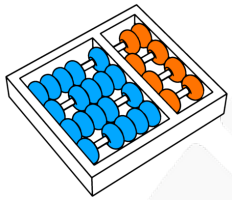
Um **PROBLEMA DE DECISÃO** é uma função $Q : \{0,1\}^* \to \{0,1\}$.

- O conjunto $\{0,1\}$ representa o alfabeto do computador.
- Uma sequência de bits $x \in \{0,1\}^*$ representa uma entrada.
- Um bit 0 ou 1 representa a resposta da pergunta.

A **LINGUAGEM** correspondente a um problema de decisão Q é o conjunto de instâncias L com resposta SIM, isso é:

$$L = \{x \in \{0,1\}^* \ : \ Q(x) = 1\}.$$

- Identificamos um problema Q com sua linguagem L.



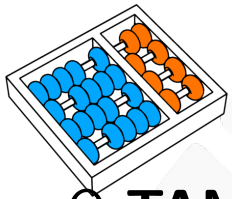
Codificação

Uma **CODIFICAÇÃO** é o mapeamento utilizado para representar um objeto como uma palavra de Σ^*.

- Dado um objeto A, sua codificação é denotada por $\langle A \rangle$.

Problema (Caminho mínimo)

$$PATH = \{\langle G, u, v, k\rangle \quad : \quad \begin{array}{l} G \text{ é um grafo,} \\ u, v \text{ são vértices de } G \text{ e} \\ k \text{ é um inteiro tais que} \\ \text{existe caminho de } u \text{ até } v \text{ em } G \\ \text{de tamanho no máximo } k\} \end{array}$$



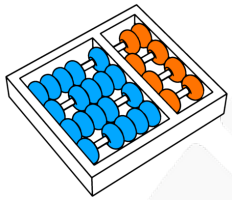
Tamanho da instância

O **TAMANHO** de uma instância $x \in \{0,1\}^*$ é o número de bits de x.

- Denotamos o tamanho de x por $n = |x|$.
- Para um objeto de alto nível A, temos $n = |\langle A \rangle|$.
- Precisamos tomar cuidado com a codificação.

Considere a linguagem PARES $= \{\langle k \rangle : k \text{ é par}\}$.

- Em unário:
  - Representamos 100 como 1111111 . . . 1111111.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \Theta(k)$.
- Em binário:
  - Representamos 100 como 1100100.
  - Nesse caso, o tamanho é $n = |\langle k \rangle| = \lceil \log_2 k \rceil = \Theta(\log k)$.
- Em ternário, quaternário etc:
  - Também, o tamanho é $n = |\langle k \rangle| = \lceil \log_b k \rceil = \Theta(\log k)$.



Linguagem aceita

Considere um algoritmo A e uma entrada $x \in \{0,1\}^*$.

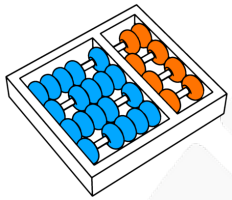
- Pode ser que A termina ao receber x e devolve $A(x) = 1$.
- Pode ser que A termina ao receber x e devolve $A(x) = 0$.
- Pode ser que A não termina ao receber x.

Um algoritmo A **ACEITA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ A(x) = 1\}.$$

Um algoritmo A **DECIDE** L se para todo $x \in \{0,1\}^*$:

- se $x \in L$, então $A(x) = 1$ (A aceita x).
- se $x \notin L$, então $A(x) = 0$ (A rejeita x).



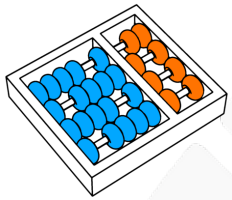
Classe P

Definição

*A **CLASSE** P é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo A que decide L em tempo polinomial.*

Em outras palavras, se $L \in \mathrm{P}$, então:

1. Existe algoritmo $A(x)$ que decide L.
2. Esse algoritmo executa em tempo polinomial em $|x|$.



Aceita e decide em tempo polinomial

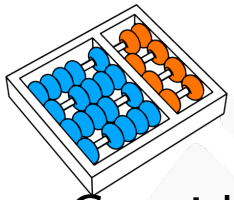
Teorema

$\mathrm{P} = \{L \subseteq \{0,1\}^* : L \text{ é aceita por um algoritmo polinomial}\}$

Demonstração:

$\subseteq$
- Considere $L \in \mathrm{P}$.
- Por definição de P, existe um algoritmo que decide L.
- Logo, também existe um algoritmo que aceita L.

$\supseteq$
- Suponha que L é aceita por um algoritmo polinomial A.
- O tempo do algoritmo é n^k, para alguma constante k.
- Construa um algoritmo A' para uma entrada x:
  1. Simule o algoritmo A executando **NO MÁXIMO** n^k passos.
  2. Se A aceitou x, então responda SIM.
  3. Se A rejeitou x ou não terminou, então responda NAO.
- Observe que o algoritmo A' decide L em tempo polinomial.



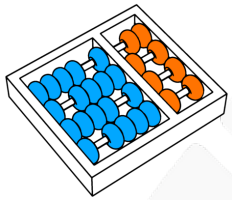
Certificado

Considere uma linguagem L:

- Tome uma instância x do problema correspondente.
- Queremos encontrar uma sequência de bits $y \in \{0,1\}^*$.
- De forma que verificar y permite concluir que $x \in L$.
- Normalmente y é a codificação de uma solução para x.
- Chamamos y de **CERTIFICADO** para x.

Problema (Certificado para PATH)

- *Tome uma instância $x = \langle G, u, v, k \rangle$ de PATH*
  1. *Se x é* SIM, ***EXISTE*** *caminho P de u a v com até k arestas.*
  2. *Se x é* NAO, ***NÃO EXISTE*** *caminho de u a v com até k arestas.*
- *No primeiro caso, $y = \langle P \rangle$ é um certificado de que $x \in PATH$.*



Verificador

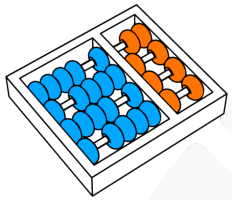
Um **VERIFICADOR** para uma linguagem L é um algoritmo que recebe uma instância x e um sequência de bits y tal que:

- Se $x \in L$, ele devolve SIM para algum certificado y.
- Se $x \notin L$, ele devolve NAO independentemente de y.

Algoritmo: VERIFICA-PATH$(\langle G, u, v, k\rangle, \langle P\rangle)$

```
1 se ⟨P⟩ não é codificação de um caminho de G
2     devolva NAO
3 se P tem mais que k arestas
4     devolva NAO
5 se P não sai de u e chega em v
6     devolva NAO
7 devolva SIM
```

- Normalmente, omitimos o passo que valida a codificação.



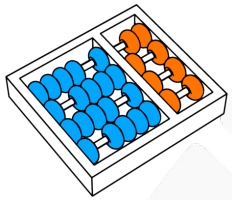
Tempo de verificação

Queremos executar o verificador em tempo polinomial:

1. O **TEMPO DO VERIFICADOR** deve ser polinomial em $|x|$ e $|y|$.
2. o **TAMANHO DO CERTIFICADO** $|y|$ deve ser polinomial em $|x|$.

Por quê?

- Queremos diferenciar as tarefas de decidir e verificar.
- Para certos problemas, **DECIDIR** uma instância é difícil.
- Mas pode ser que **VERIFICAR** uma solução seja fácil.
- Veremos um exemplo em seguida.



Ciclo hamiltoniano

Um **CICLO HAMILTONIANO** em um grafo G é um ciclo que passa por todos os vértices.

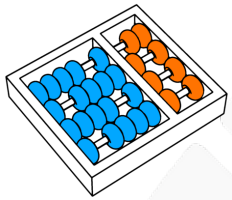
- Se houver esse ciclo, dizemos que G é hamiltoniano.

Problema (Ciclo hamiltoniano)

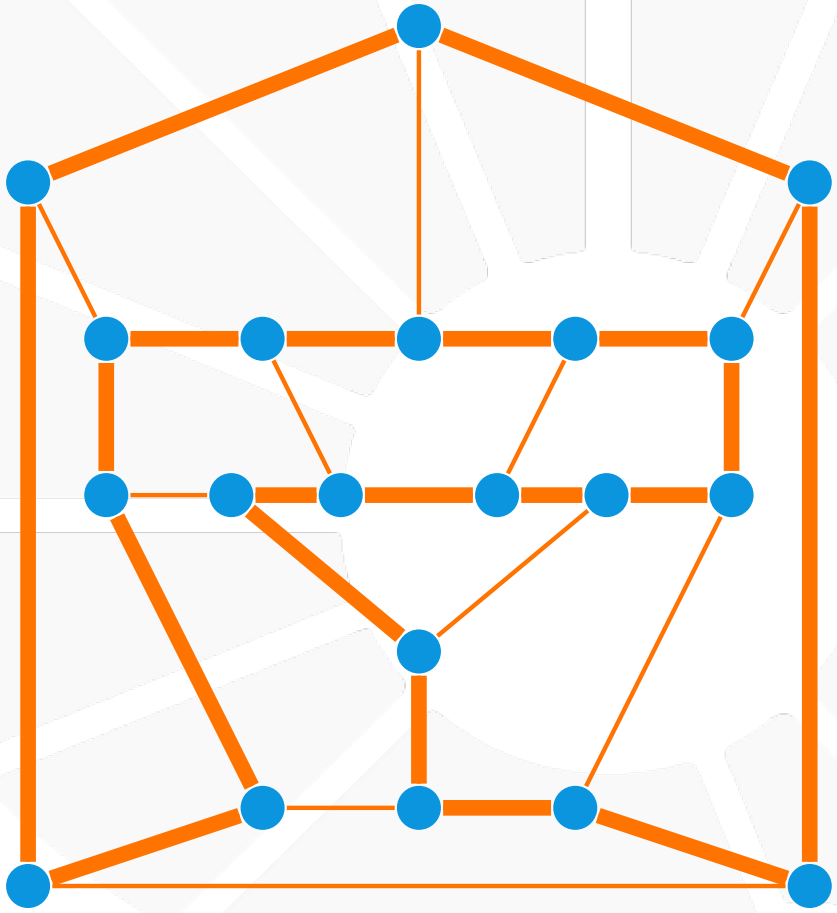
$$\mathit{HAM\text{-}CYCLE} = \{\langle G \rangle : G \textit{ é um grafo hamiltoniano } \}$$

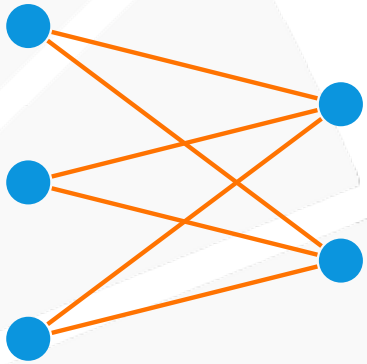
Escolhas para certificado:

- Um ciclo hamiltoniano de G.
- Uma sequência de vértices de G.

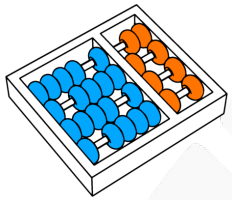


Exemplo de ciclo hamiltoniano





O grafo da direita não tem ciclo hamiltoniano!



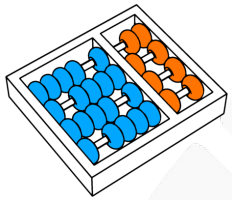
Procurando um ciclo hamiltoniano

Podemos **DECIDIR** se há um ciclo hamiltoniano:

- O algoritmo trivial gasta tempo $O(V!)$.
- Os melhores algoritmos têm tempo $O(n^2 2^n)$.
- Esses algoritmos são determinísticos.

Mas se **SORTEARMOS** um ciclo C:

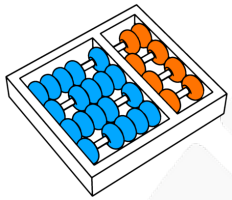
- É fácil verificar se ele é hamiltoniano em tempo linear.
- Basta sortear uma sequência S de $|V|$ vértices.
- O sorteio do ciclo é um processo não determinístico.



Verificando um ciclo

Algoritmo: VERIFICA-HAMCICLO($\langle G \rangle, \langle S \rangle$)

```
1  se S não contém todos os vértices de G
2      devolva NAO
3  n ← |S|
4  S[n + 1] ← S[1]
5  para cada i = 1, 2, ..., |S|
6      u ← S[i]
7      v ← S[i + 1]
8      se (u, v) não é aresta de G
9          devolva NAO
10 devolva SIM
```



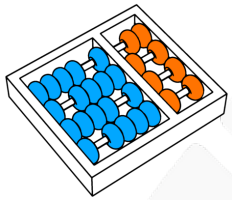
Linguagem verificada

Um algoritmo V **VERIFICA** uma linguagem L se:

$$L = \{x \in \{0,1\}^* \ : \ \text{existe } y \in \{0,1\}^* \text{ tal que } V(x,y) = 1\}$$

Em outras palavras:

1. Se $x \in L$, então **EXISTE** certificado y tal que $V(x,y) = 1$.
2. Se $x \notin L$, então **NÃO EXISTE** certificado y tal que $V(x,y) = 1$.



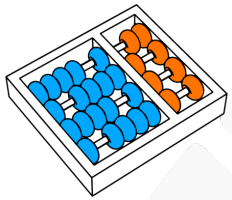
A classe NP

Definição

*A **CLASSE** NP é o conjunto de linguagens $L \subseteq \{0,1\}^*$ para as quais existe algoritmo V que verifica L em tempo polinomial.*

Em outras palavras, se $L \in$ NP, então:

1. Existe algoritmo $V(x, y)$ que verifica L.
2. Esse algoritmo executa em tempo polinomial em $|x|$ e $|y|$.
3. Para cada $x \in L$, existe certificado y polinomial em $|x|$.



Determinando se problema é NP

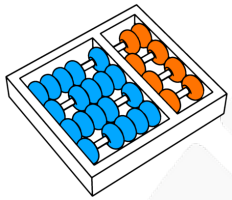
Dado problema L, devemos seguir esses passos:

1. Identifique um **CERTIFICADO** de tamanho polinomial para L.
2. Construa um **ALGORITMO VERIFICADOR** $V(x, y)$ polinomial.
3. Demonstre que:
   - Se $x \in L$, então **existe** y tal que $V(x, y) = 1$.
   - Se $x \notin L$, então para **qualquer** y, vale $V(x, y) = 0$.

Exemplos:

- VERIFICA-PATH é um verificador para PATH.
- VERIFICA-HAMCICLO é um verificador para HAM-CYCLE.

Portanto, PATH e HAM-CYCLE estão em NP.



NP contém P

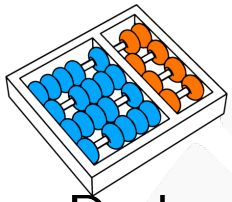
Seja $L \in \mathrm{P}$:

- Exite algoritmo A que decide L.
- Vamos construir um verificador para L.

Algoritmo: $\textsc{Verifica-L}(x, y)$:

1 **devolva** $A(x)$

- O verificador só precisa ignorar o argumento y.
- Podemos escolher qualquer certificado (e.g. $y = \varepsilon$).
- Analisamos:
  - Se $x \in L$, então $A(x) = 1$, daí $\textsc{Verifica-L}(x, \varepsilon) = 1$.
  - Se $x \notin L$, então $A(x) = 0$, daí $\textsc{Verifica-L}(x, y) = 0$ para todo y.
- Assim $L \in \mathrm{NP}$ e concluímos que $\mathrm{P} \subseteq \mathrm{NP}$.



O problema complementar

Dada uma linguagem L, o seu complementar é $\overline{L} = \{0,1\}^* \setminus L$.

- $\overline{L}$ é o conjunto de instâncias de L com resposta NAO.
- Exemplo:

$$\text{HAM-CYCLE} = \{\langle G \rangle : G \text{ possui ciclo hamiltoniano}\}$$

$$\overline{\text{HAM-CYCLE}} = \{\langle G \rangle : G \textbf{ NÃO} \text{ possui ciclo hamiltoniano}\}$$

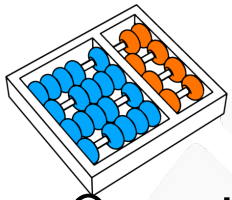
Definição

A classe complementar de NP *é o conjunto de linguagens*

$$\text{co-NP} = \{L \subseteq \Sigma^* : \overline{L} \in \text{NP}\}$$

Exemplos:

- $\overline{\text{HAM-CYCLE}} \in$ co-NP
- P $\subseteq$ co-NP



Outro exemplo: tautologia

O conjunto co-NP contém as linguagens que possuem um certificado curto para a resposta NAO.

Problema (Tautologia)

Dada uma fórmula booleana com um conjunto de n variáveis e operadores $\wedge$, $\vee$, $\neg$ etc, ela é verdadeira para toda atribuição de variáveis?

- $p \vee \neg p$, $((z \wedge y) \vee \neg x \vee (x \wedge \neg y)) \vee (\neg z \wedge y)$ são tautologias.
- p, $(\neg y \vee x) \wedge (y \vee \neg x)$ **NÃO** são tautologias.

- Certificado curto para instâncias NAO: $x = 0, y = 1$.
- Não conhecemos certificado curto para instâncias SIM.

NP-complexidade

MO417 - Complexidade de Algoritmos I

06/24

25

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP