

NP-COMPLETUDE

MC558 - Projeto e Análise de Algoritmos II

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

11/24

25

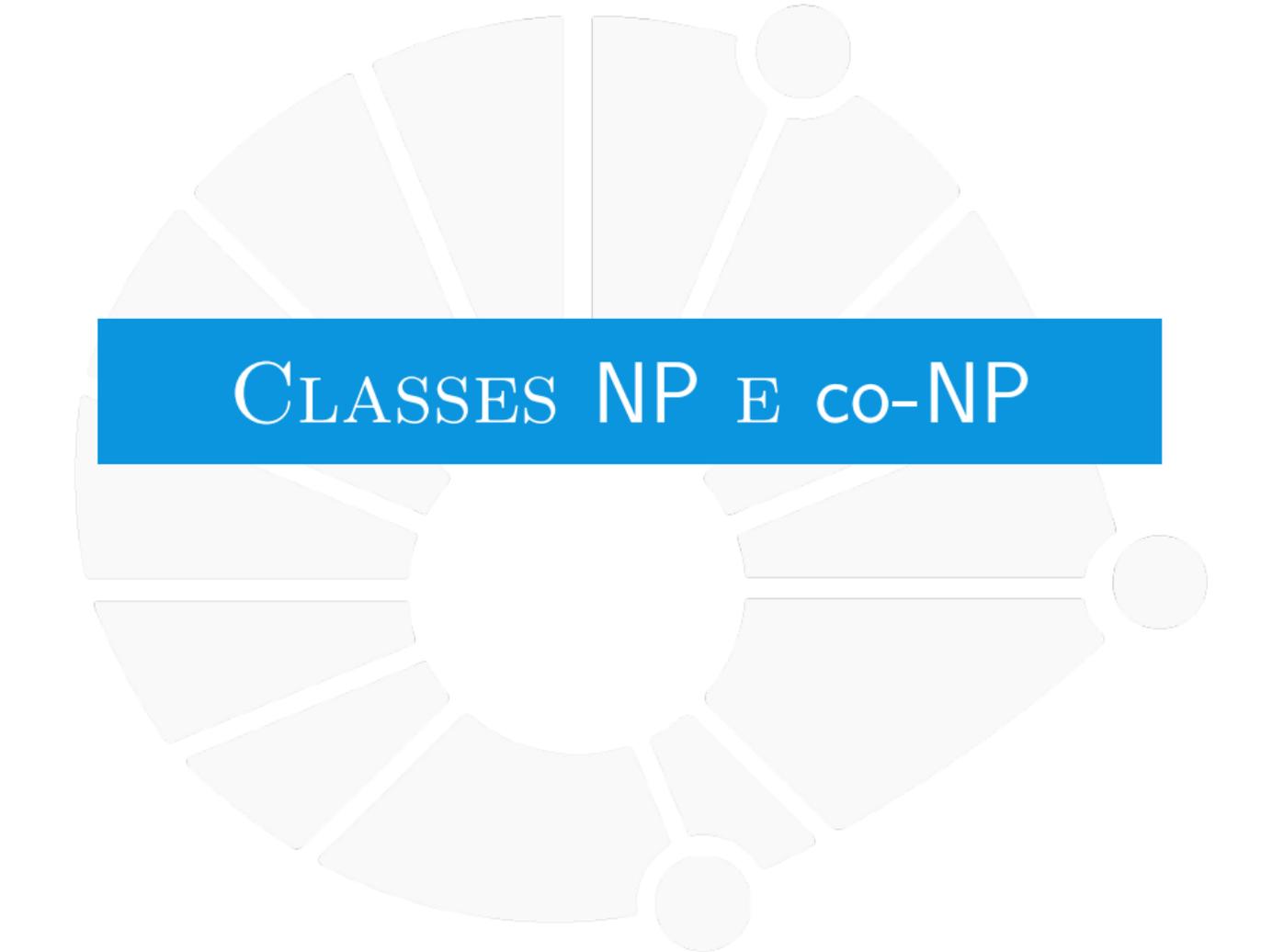


UNICAMP

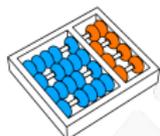


“Se $P = NP$ então o mundo seria um lugar profundamente diferente do que normalmente assumimos. Não haveria valor especial em “saltos criativos”, nenhuma lacuna fundamental entre resolver um problema e reconhecer a solução assim que ela fosse encontrada. Qualquer pessoa que pudesse apreciar uma sinfonia seria Mozart; qualquer um que pudesse acompanhar um argumento passo a passo seria Gauss; qualquer pessoa que pudesse reconhecer uma boa estratégia de investimento seria Warren Buffett.”

Scott Aaronson.



CLASSES NP E co-NP



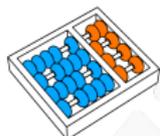
Certificado

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 $x \in \text{PATH}$.



Verificador

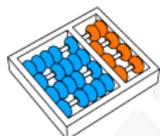
Um **VERIFICADOR** para uma linguagem L é um algoritmo que recebe uma instância x e um sequência de bits y tal que:

- ▶ 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 $\langle P \rangle$ 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.



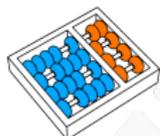
Tempo de verificação

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.

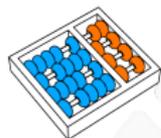
- ▶ Se houver esse ciclo, dizemos que G é hamiltoniano.

Problema (Ciclo hamiltoniano)

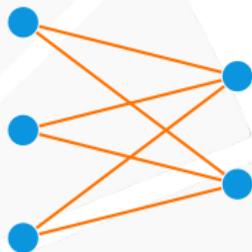
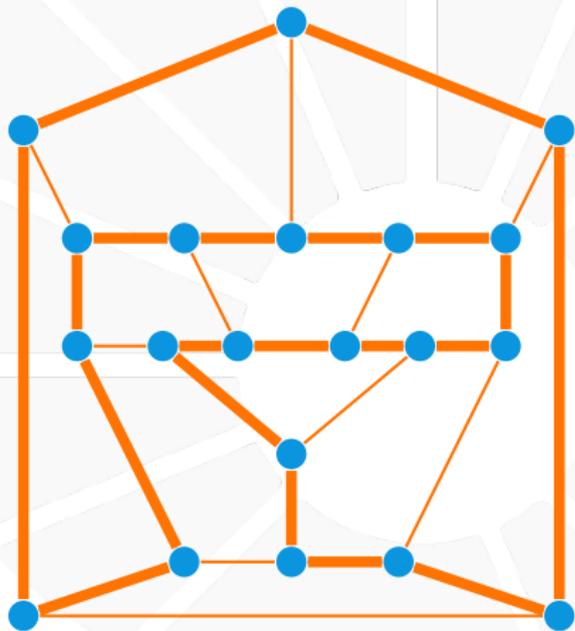
$$HAM-CYCLE = \{ \langle G \rangle : G \text{ é um grafo hamiltoniano} \}$$

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 :

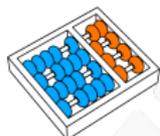
- ▶ É fácil verificar se ele é hamiltoniano em tempo linear.
- ▶ Basta sortear uma sequência S de $|V|$ vértices.
- ▶ O sorteio do ciclo é um processo não determinístico.



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  $n \leftarrow |S|$ 
4  $S[n + 1] \leftarrow S[1]$ 
5 para cada  $i = 1, 2, \dots, |S|$ 
6   └  $u \leftarrow S[i]$ 
7     └  $v \leftarrow S[i + 1]$ 
8       se  $(u, v)$  não é aresta de  $G$ 
9         └ devolva NAO
10 devolva SIM
```



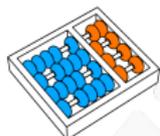
Linguagem verificada

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 $V(x,y) = 1$.



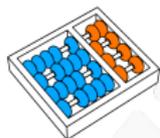
A classe NP

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 $L \in \text{NP}$, 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

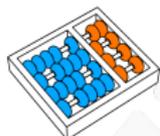
Dado problema L , devemos seguir esses passos:

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

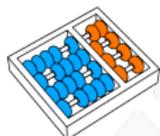
Seja $L \in P$:

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

Algoritmo: 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 $\text{VERIFICA-}L(x, \varepsilon) = 1$.
 - ▶ Se $x \notin L$, então $A(x) = 0$, daí $\text{VERIFICA-}L(x, y) = 0$ para todo y .
- ▶ Assim $L \in \text{NP}$ e concluímos que $P \subseteq \text{NP}$.



O problema complementar

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

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

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

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

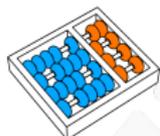
Definição

A classe complementar de NP é o conjunto de linguagens

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

Exemplos:

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



Outro exemplo: tautologia

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

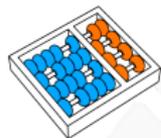
Problema (Tautologia)

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-COMPLETUDE

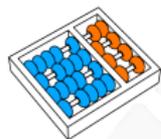


Resumo das classes até agora

$P = \{L \subseteq \Sigma^* : \text{há algoritmo que } \mathbf{DECIDE} L \text{ em tempo polinomial}\}$

$NP = \{L \subseteq \Sigma^* : \text{há algoritmo que } \mathbf{VERIFICA} L \text{ em tempo polinomial}\}$

$\text{co-NP} = \{L \subseteq \Sigma^* : \text{há algoritmo que } \mathbf{VERIFICA} \bar{L} \text{ em tempo polinomial}\}$



Possíveis configurações dessas classes

$P = NP = \text{co-NP}$

$NP = \text{co-NP}$

P

NP

P

co-NP

NP

P

co-NP



Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

- ▶ Podemos **DECIDIR** em tempo polinomial: PATH.
- ▶ Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

Por que não conhecemos algoritmo rápido para HAM-CYCLE?

- ▶ Será que HAM-CYCLE é mais difícil do que PATH?
- ▶ Será que HAM-CYCLE é mais difícil que qualquer um em P?

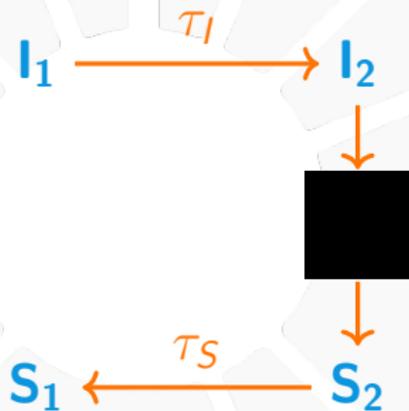
HAM-CYCLE é NP-**DIFÍCIL**:

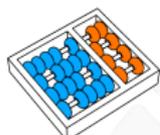
- ▶ Não sabemos se é mais difícil do que algum problema P.
- ▶ Mas sabemos que é **TÃO DIFÍCIL QUANTO** qualquer problema NP.



Como comparar problemas?

A ferramenta adequada para comparar problemas são as reduções





Reduções de Karp

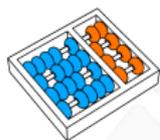
Estamos interessados em reduções em que:

1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S .

Definição (Redução de Karp)

A linguagem L_1 é **REDUTÍVEL EM TEMPO POLINOMIAL** para L_2 se:

- ▶ Existe algoritmo $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$,
 - ▶ $f(x)$ leva tempo polinomial em $|x|$,
 - ▶ $x \in L_1$ se e somente se $f(x) \in L_2$.
-
- ▶ Escrevemos $L_1 \preceq_p L_2$.
 - ▶ Dizemos que f reduz L_1 para L_2 .



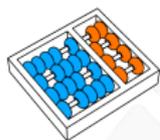
Reduções de tempo polinomial

Teorema

Considere linguagens $L_1, L_2 \subseteq \{0, 1\}^*$ tais que $L_1 \preceq_p L_2$.
Se $L_2 \in P$, então $L_1 \in P$.

Demonstração:

- ▶ Suponha que $L_2 \in P$.
- ▶ Seja A_2 um algoritmo que decide L_2 em tempo polinomial.
- ▶ Seja f um algoritmo polinomial que reduz L_1 a L_2 .
- ▶ Crie um algoritmo A_1 fazendo $A_1(x) = A_2(f(x))$.
- ▶ Já que f é uma redução, obtemos:
 - ▶ Se $x \in L_1$, então $f(x) \in L_2$ e $A_2(f(x)) = 1$, daí $A_1(x) = 1$.
 - ▶ Se $x \notin L_1$, então $f(x) \notin L_2$ e $A_2(f(x)) = 0$, daí $A_1(x) = 0$.
- ▶ Assim, A_1 decide L_1 em tempo polinomial.



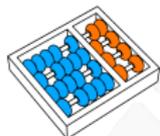
Classe NP-completo

Definição

A **CLASSE NP-COMPLETO** é o conjunto de linguagens $L \subseteq \{0, 1\}^*$ tais que:

1. $L \in \text{NP}$.
2. $L' \preceq_p L$ para todo $L' \in \text{NP}$.

► Se apenas 2 for satisfeita, dizemos que L é **NP-DIFÍCIL**.



Condição para $NP = P$

Teorema

Se existe algoritmo que decide $L \in NP$ -completo em tempo polinomial, então $P = NP$.

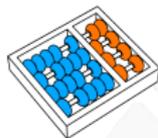
Demonstração:

- ▶ Suponha que existe $L \in P \cap NP$ -completo.
- ▶ Como $L \in NP$ -completo, para toda $L' \in NP$, temos $L' \leq_p L$.
- ▶ Mas como $L \in P$, isso implica $L' \in P$ pelo teorema anterior.
- ▶ Então, $NP \subseteq P$ e, portanto, $NP = P$.

Teorema

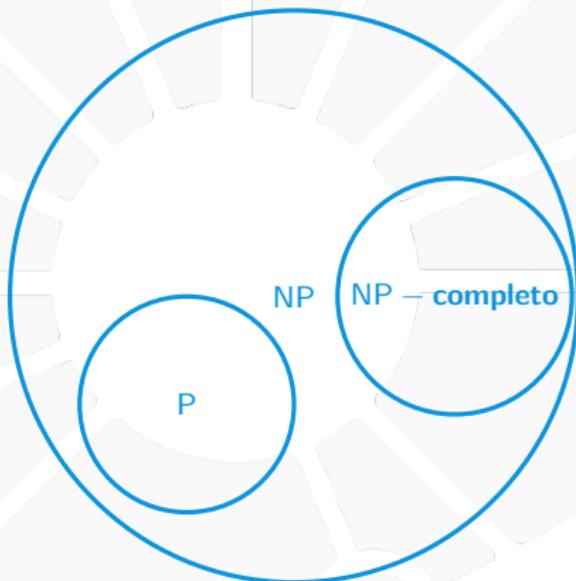
Se existe uma linguagem $L \in NP$ tal que $L \notin P$, então NP -completo $\cap P = \emptyset$.

- ▶ Exercício.



Possível configuração de NP-completo

Como acreditamos que é a relação das classes:





A classe NP-completo não é vazia

Mas será que a classe NP-completo é vazia?

- ▶ Cook e Levin responderam que **NÃO** (independentemente).
- ▶ Mostraram que $SAT \in NP\text{-completo}$.

Teorema (Cook-Levin)

SAT é NP-completo.

- ▶ SAT é o problema da satisfatibilidade.
- ▶ Não vamos demonstrar esse teorema.
- ▶ Mas veremos um rascunho para um problema parecido.



Satisfatibilidade

Considere uma **FÓRMULA BOOLEANA**:

- ▶ Contém um conjunto de variáveis booleanas.
- ▶ É escrita usando os seguintes operadores:
 1. Negação (\neg).
 2. Conjunção (\wedge).
 3. Disjunção (\vee).
 4. Implicação (\rightarrow).
 5. Equivalência (\leftrightarrow).

Problema (Satisfatibilidade (SAT))

- ▶ **Entrada:** Uma fórmula booleana.
- ▶ **Saída:** Decidir se existe atribuição de variáveis booleana para a qual a avaliação da fórmula é verdadeira.

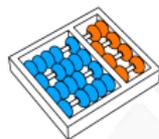
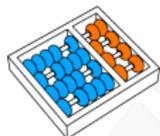


Tabela de operadores

a	b	$\neg a$	$a \wedge b$	$a \vee b$	$a \rightarrow b$	$a \leftrightarrow b$
F	F	V	F	F	V	V
F	V	V	F	V	V	F
V	F	F	F	V	F	F
V	V	F	V	V	V	V



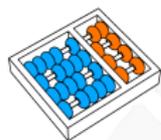
Fórmula satisfazível

Uma fórmula é **SATISFAZÍVEL** se houver atribuição das variáveis para a qual a avaliação é verdadeira.

Exemplo:

- ▶ Fórmula: $f = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.
- ▶ Atribuição: $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$
- ▶ Avaliação:

$$\begin{aligned} f &= ((0 \rightarrow 0) \vee \neg((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0 \\ &= (1 \vee \neg((1 \leftrightarrow 1) \vee 1)) \wedge 1 \\ &= (1 \vee \neg(1 \vee 1)) \wedge 1 \\ &= (1 \vee 0) \wedge 1 \\ &= 1 \end{aligned}$$

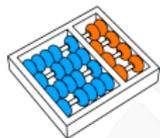


Linguagem correspondente

A linguagem SAT é aquela que contém fórmulas booleanas com uma atribuição verdadeira.

Problema (Satisfatibilidade)

$$SAT = \{ \langle f \rangle : f \text{ é uma fórmula booleana satisfazível} \}$$



É fácil verificar

Lema

SAT está em NP.



Prova

Escrevemos um algoritmo verificador:

Algoritmo: VERIFICA-SAT($\langle f \rangle, \langle y \rangle$)

- 1 se $f(y) = 1$
 - 2 └─ devolva SIM
 - 3 senão
 - 4 └─ devolva NAO
-

Observe que o algoritmo verifica SAT em tempo polinomial:

1. ▶ Se $\langle f \rangle \in \text{SAT}$, então f é satisfazível.
 - ▶ Portanto, existe uma atribuição y das variáveis que faz f verdadeira.
 - ▶ Logo, VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.
2. ▶ Suponha VERIFICA-SAT($\langle f \rangle, \langle y \rangle$) devolve SIM.
 - ▶ Então, y é uma atribuição verdadeira para a fórmula.
 - ▶ Portanto, f é satisfazível e $\langle f \rangle \in \text{SAT}$.

NP-COMPLETUDE

MC558 - Projeto e Análise de Algoritmos II

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

11/24

25



UNICAMP

