NP-completude

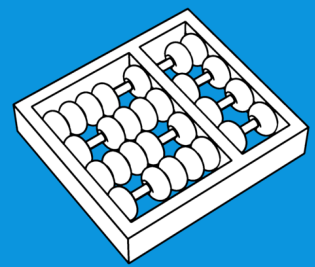
MO417 - Complexidade de Algoritmos I

06/24

26

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

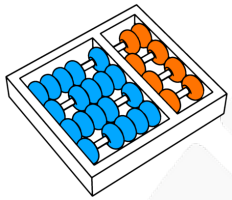
UNICAMP



"If $P = NP$, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps,"no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett."

Scott Aaronson.

NP-completude

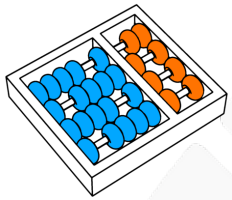


Resumo das classes até agora

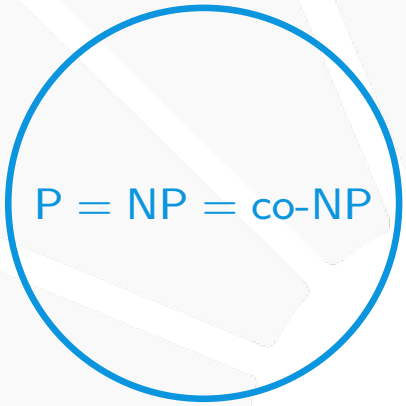
$$\mathrm{P} = \{L \subseteq \Sigma^* : \text{há algoritmo que } \textbf{DECIDE } L \text{ em tempo polinomial}\}$$

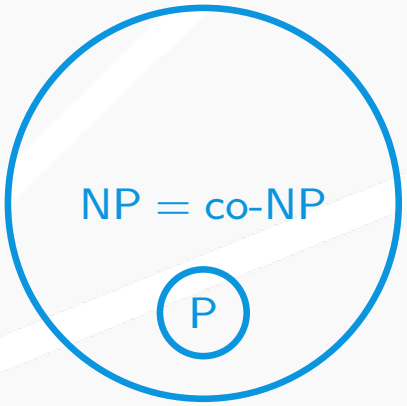
$$\mathrm{NP} = \{L \subseteq \Sigma^* : \text{há algoritmo que } \textbf{VERIFICA } L \text{ em tempo polinomial}\}$$

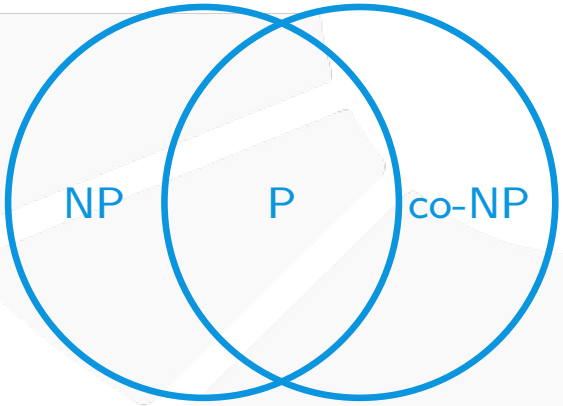
$$\text{co-NP} = \{L \subseteq \Sigma^* : \text{há algoritmo que } \textbf{VERIFICA } \overline{L} \text{ em tempo polinomial}\}$$

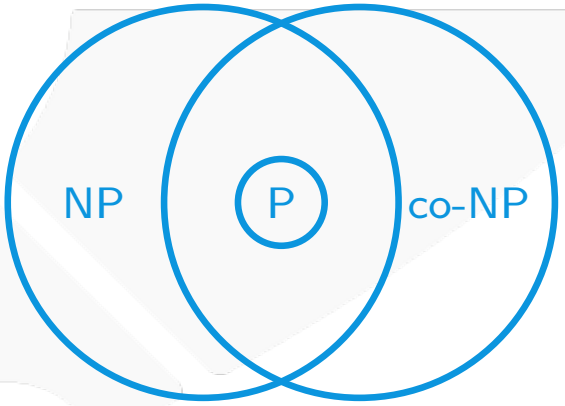


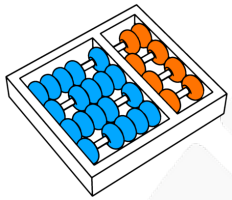
Possíveis configurações dessas classes











Refletindo sobre o que vimos

Vimos alguns exemplos de problemas que:

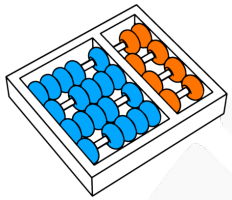
- Podemos **DECIDIR** em tempo polinomial: PATH.
- Só sabemos **VERIFICAR** em tempo polinomial: HAM-CYCLE.

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?

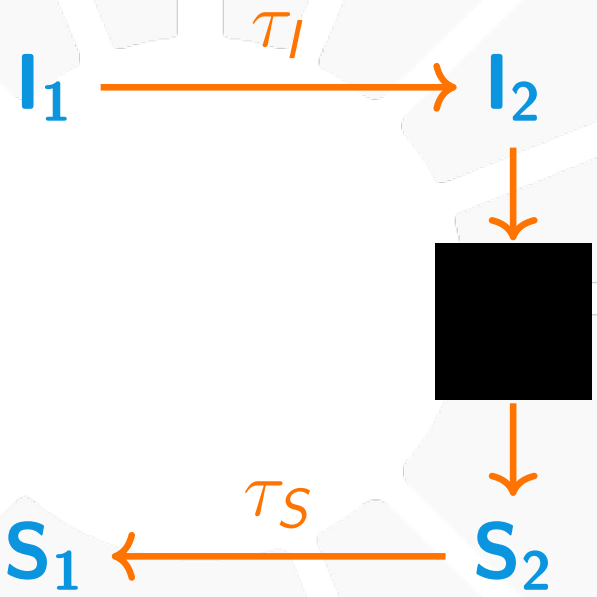
Vamos mostrar que HAM-CYCLE é NP-**DIFÍCIL**:

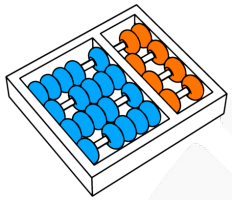
- Não sabemos se é mais difícil do que algum problema P.
- Mas sabemos que é **TÃO DIFÍCIL QUANTO** qualquer problema NP.



Como comparar problemas?

A ferramenta adequada para comparar problemas são as reduções





Reduções de Karp

Estamos interessados em reduções em que:

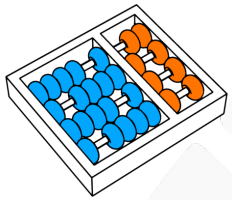
1. A transformação de entrada τ_I leva tempo polinomial.
2. **NÃO** há transformação de saída τ_S.

Definição (Redução de Karp)

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

- *Existe algoritmo $f : \{0,1\}^* \to \{0,1\}^*$,*
- *$f(x)$ leva tempo polinomial em $|x|$,*
- *$x \in L_1$ se e somente se $f(x) \in L_2$.*

- Escrevemos $L_1 \preccurlyeq_p L_2$.
- Dizemos que f reduz L_1 para L_2.



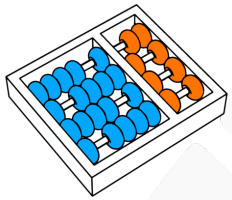
Reduções de tempo polinomial

Teorema

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

Demonstração:

- Suponha que $L_2 \in \mathrm{P}$.
- 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.



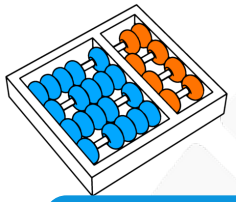
Classe NP-completo

Definição

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

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

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



Condição para NP = NP

Teorema

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

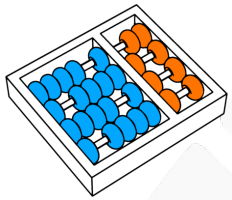
Demonstração:

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

Teorema

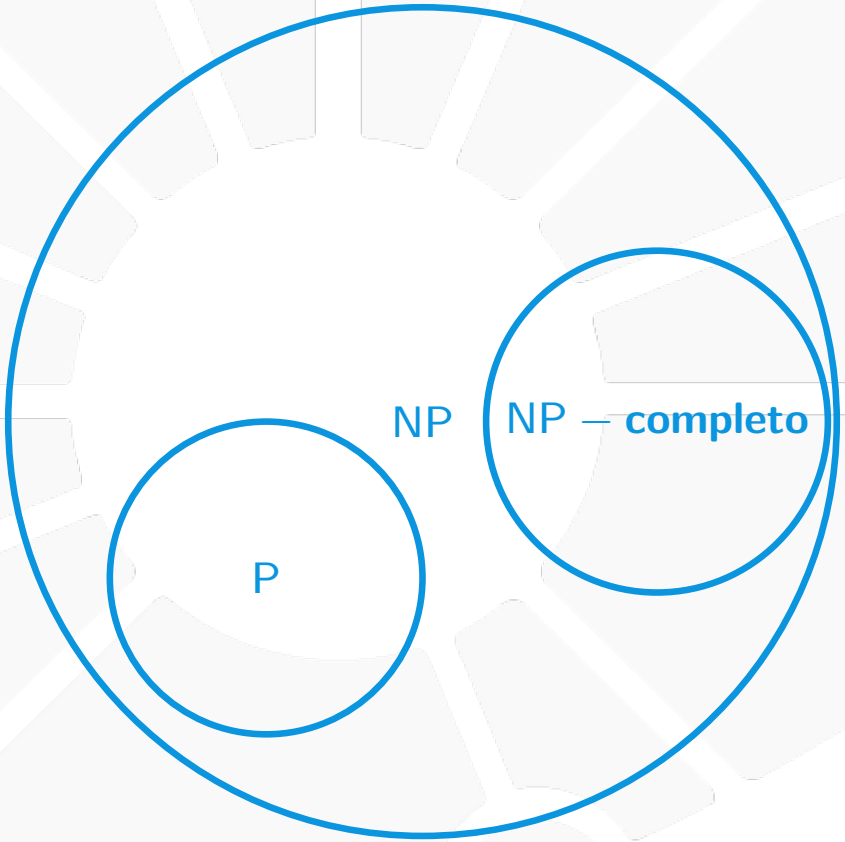
Se existe uma linguagem $L \in \mathrm{NP}$ tal que $L \notin \mathrm{P}$, então NP-*completo* $\cap \mathrm{P} = \emptyset$.

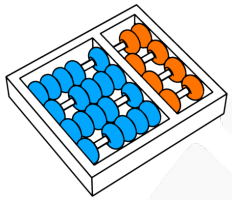
- Exercício.



Possível configuração de NP-completo

Como acreditamos que é a relação das classes:





A classe NP-completo não é vazia

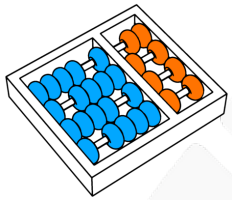
Mas será que a classe NP-completo é vazia?

- Cook e Levin responderam que **NÃO** (independentemente).
- Mostraram que SAT $\in$ NP-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 conjuntos 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.*

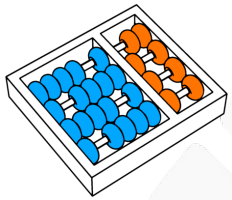
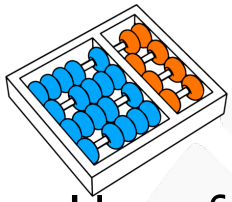


Tabela de operadores

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



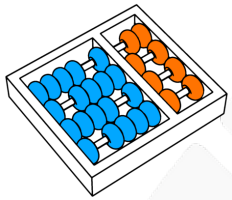
Fórmula satisfazível

Uma formula 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:

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

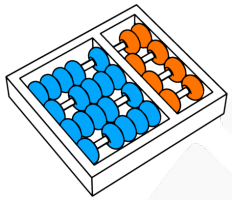


Linguagem correspondente

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

Problema (Satisfatibilidade)

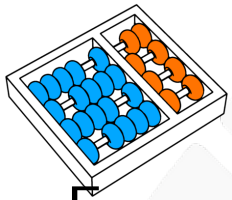
$$SAT = \{\langle f \rangle : f \text{ é uma fórmula booleana satisfazível}\}$$



É fácil verificar

Lema

SAT está em NP*.*



Prova

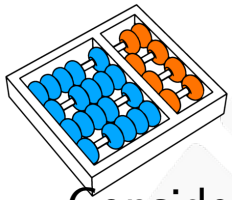
Escrevemos um algoritmo verificador:

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 $\langle f \rangle \in$ 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 $\langle f \rangle \in$ SAT.



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

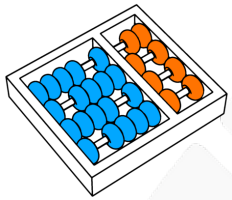
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

Uma **ATRIBUIÇÃO** de um circuito é uma atribuição de um bit 0 ou 1 para cada fio de entrada.

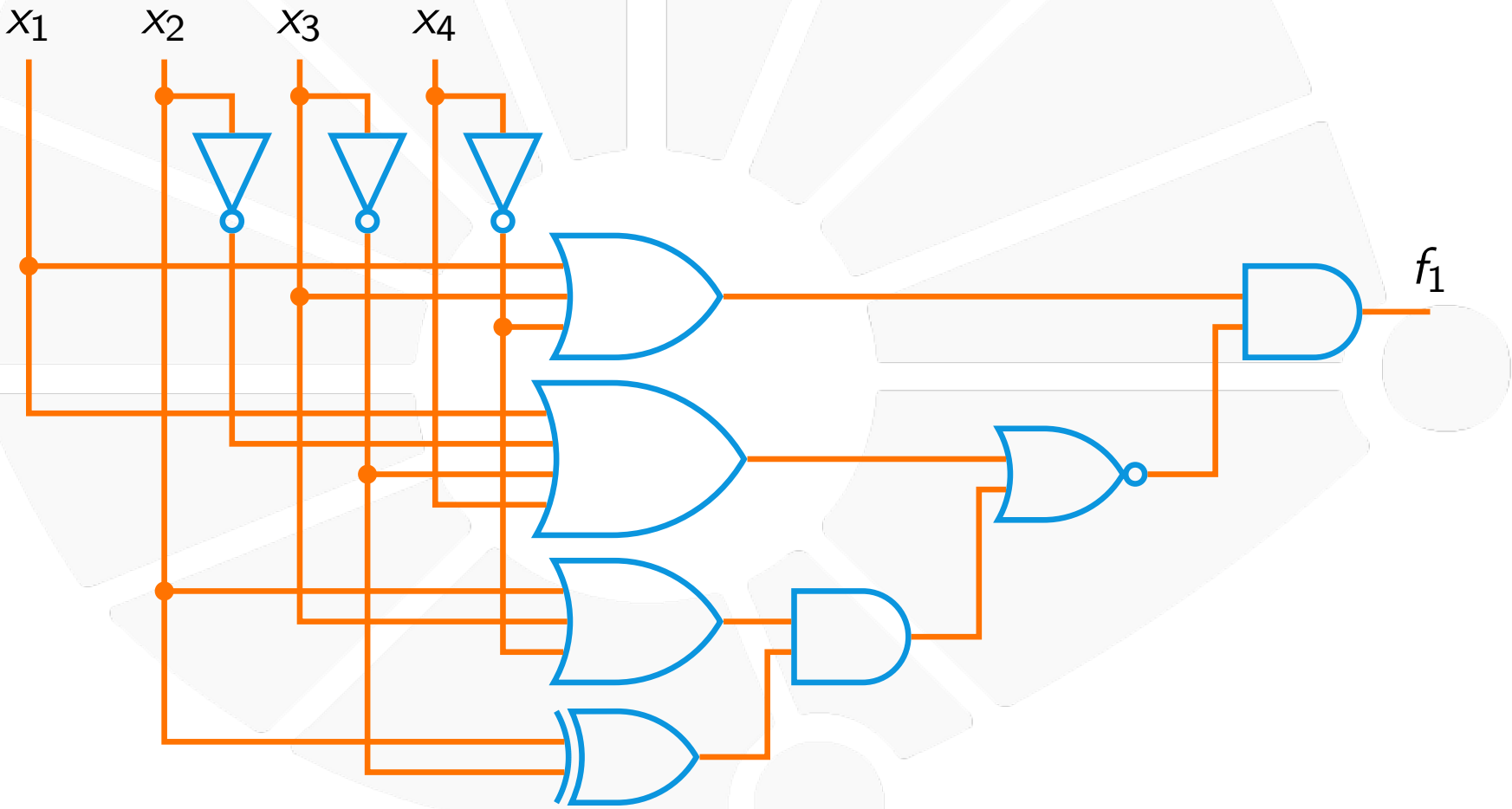
- Um circuito é **SATISFAZÍVEL** se ele possuir uma atribuição que resulta em bit 1 no fio de saída.

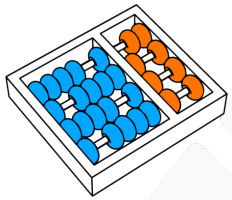
Problema (Satisfatibilidade de circuito (C-SAT))

- ***Entrada:*** *Um circuito lógico.*
- ***Saída:*** *Decidir se o circuito é satisfazível.*



Exemplo de circuito





Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$

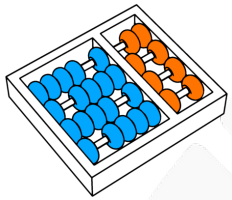
Há um algoritmo simples de tempo $O(2^n(n+m))$:

- n é o número de fios de entrada.
- m é o número de ligações entre portas.

Lema

C-SAT é NP.

- Análogo à demonstração de que SAT $\in$ NP. (exercício)



Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

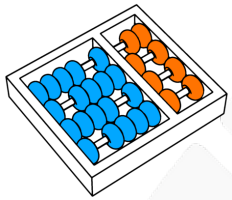
- Falta mostrar então que C-SAT é NP-difícil.
- Queremos que para todo $Q \in \text{NP}$, tenhamos $Q \preccurlyeq_p \text{C-SAT}$.

Nosso objetivo é projetar uma redução polinomial F:

Problema

- ***Entrada:*** *Instância* $x \in \{0,1\}^*$ *de* Q.
- ***Saída:*** *Circuito* $F(x)$ *tal que:*

$$x \in Q \quad \textbf{\textit{se e somente se}} \quad F(x) \in \textit{C-SAT}.$$



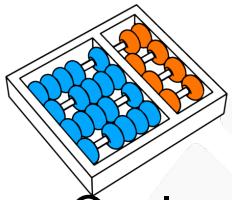
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

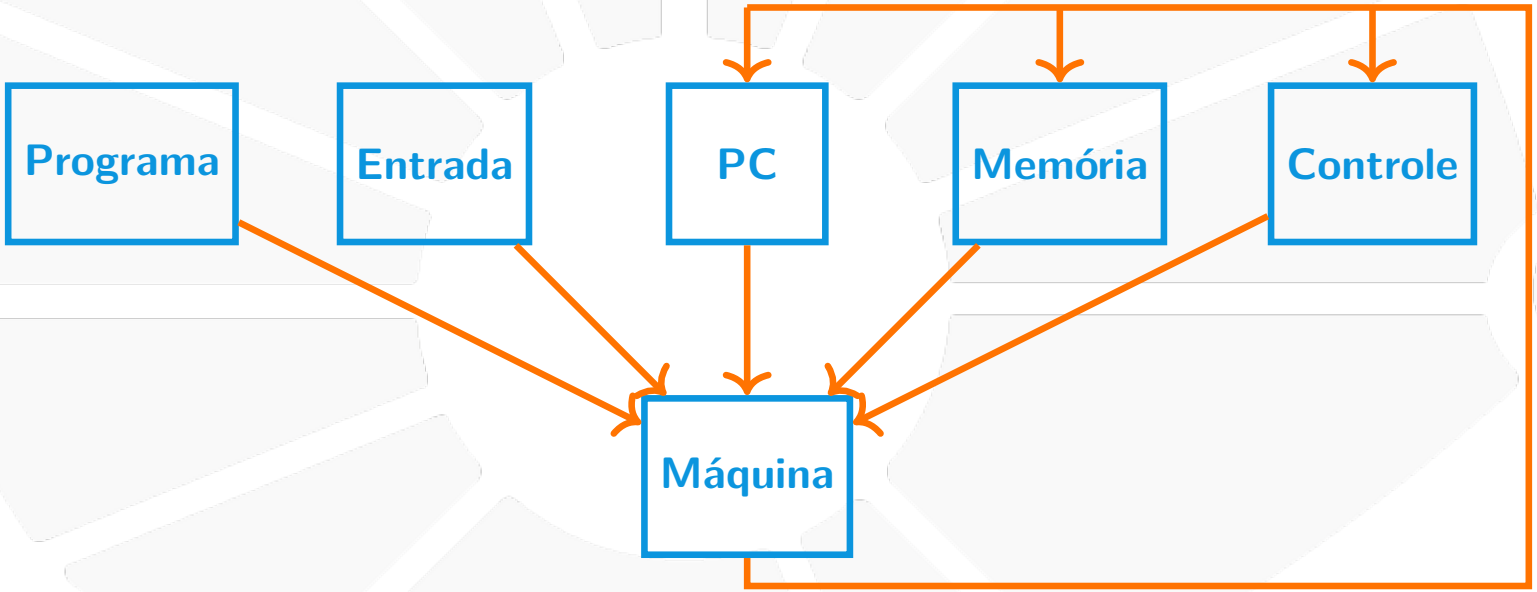
Demonstração:

- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:
  - A entrada x tem tamanho $|x| = n$.
  - O certificado y tem tamanho $|y| = n^{k'}$, para k' constante.
- Relembre que A leva tempo polinomial em $|x|$ e $|y|$.
- Assim, ele executa até n^k passos, para k constante.
- A ideia é montar um circuito que simula n^k passos de A.

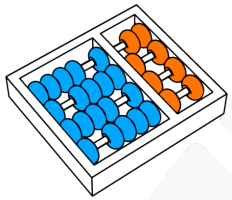


Continuação da demonstração

O algoritmo A pode ser implementado por um computador de circuitos lógicos.

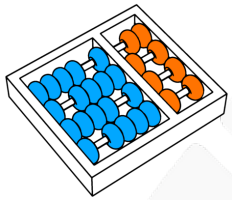


- Os circuitos têm **RETROALIMENTAÇÃO**.
- Após executar uma instrução, a memória é modificada

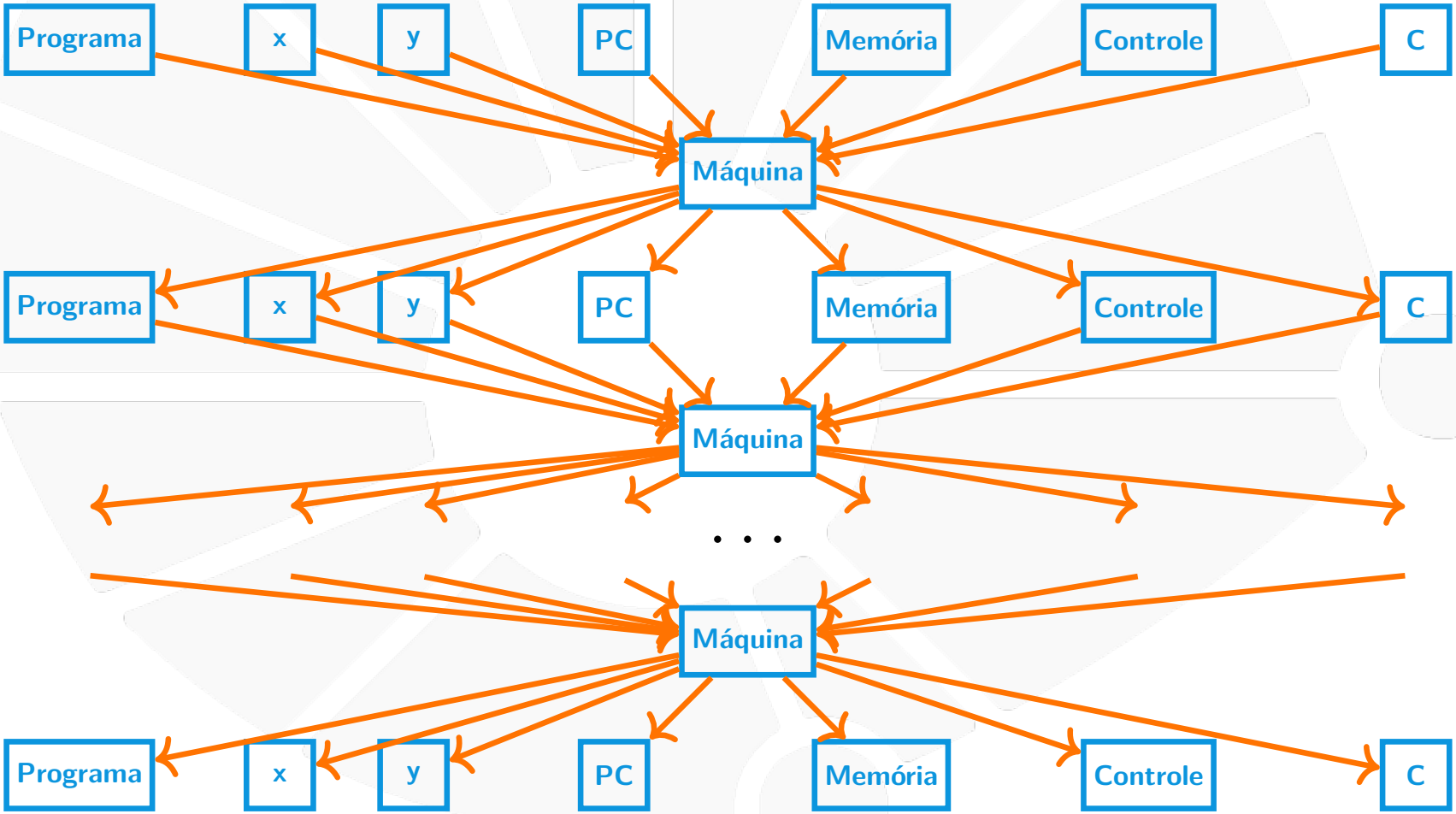


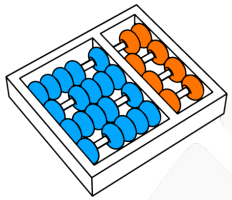
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.
- Cada instrução executada pelo algoritmo A:
  - Começa em um estado de memória, PC e controle.
  - Modifica esse estado de memória, PC e controle.
- Criamos n^k cópias da máquina, uma para cada instrução:
  1. O estado de entrada da primeira máquina corresponde ao estado inicial da execução do algoritmo.
  2. A saída de uma instrução correspondente a uma cópia modifica o estado de entrada da cópia seguinte.
  3. A saída do circuito é a saída de uma conjunção (porta $\vee$) ligando os bits correspondentes a C.



Continuação da demonstração





Continuação da demonstração

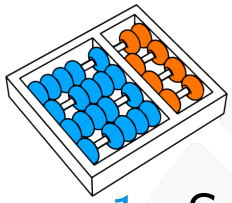
O circuito construído tem tamanho polinomial.

- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.
- Fazemos apenas n^k cópias da máquina.

Fios de entrada e de saída:

- Todo circuito é fixo e pode ser construído a partir de x.
- Há um **FIO DE ENTRADA** para cada bit do certificado y.
- O **FIO DE SAÍDA** vale 1 se algum campo C foi mudado para 1.

Como a redução levou tempo polinomial, falta mostrar apenas que $x \in Q$ se e somente se $F(x) \in$ C-SAT.



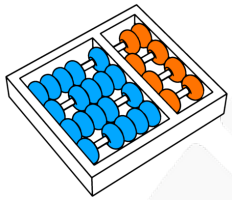
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.
   - Então, alguma cópia da máquina muda C para 1.
   - A saída do circuito será 1.
2. Suponha que o circuito $F(x)$ é satisfazível:
   - Então, para algum valor y dos fios de entrada a saída do circuito é 1.
   - Logo, para esse y, alguma cópia da máquina muda o campo C para 1.
   - Isso só acontece quando $A(x, y) = 1$.
   - Portanto, y é um certificado tal que $A(x, y) = 1$.
   - Assim, $x \in Q$.

Teorema

C-SAT é NP*-completo.*

Demonstrando NP-completude



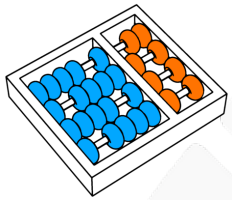
Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para **TODO** problema $L \in$ NP.
  2. Mostrar que $L \preccurlyeq_p Q$ para **ALGUM** problema $L \in$ NP-difícil.

Normalmente usamos a segunda opção:

- Basta reduzir um problema NP-difícil para o problema Q.
- Ou seja, mostramos que Q é **TÃO DIFÍCIL** quanto um NP-difícil.
- Esse problema NP-difícil pode ser C-SAT, por exemplo.



NP-completude via redução

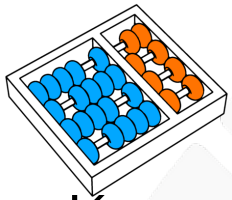
Teorema

Considere uma linguagem Q e seja $L \in$ NP-difícil.
Se $L \preccurlyeq_p Q$, então Q é NP-difícil.

Demonstração:

- Como L é NP-difícil, para todo $L' \in$ NP temos $L' \preccurlyeq_p L$.
- Assim, $L' \preccurlyeq_p L$ e $L \preccurlyeq_p Q$, o que implica $L' \preccurlyeq_p Q$.
- Portanto Q é NP-difícil.

Se além disso $Q \in$ NP, então Q é NP-completo.



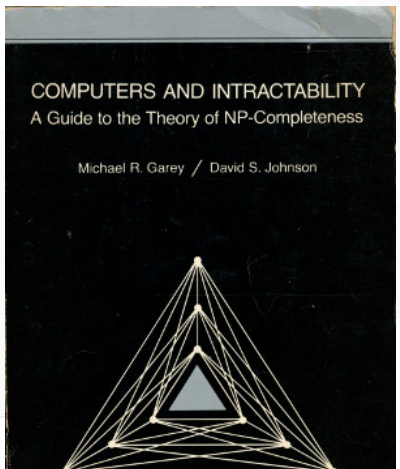
Literatura

Karp mostrou em 1972 que 21 problemas são NP-completos:

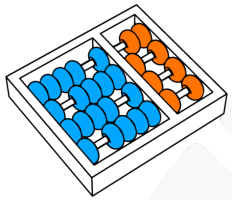
- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

Com sorte, seu problema foi estudado por Garey e Johnson:

- Livro publicado em 1979.
- Estuda e classifica dezenas de problemas.
- Entre os mais citados em Ciência da Computação.



Reduções



Satisfatibilidade

Queremos provar que o seguinte problema é NP-completo:

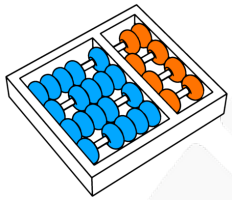
Problema (Satisfatibilidade)

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

Para isso, consideraremos um caso particular: quando a fórmula está escrita em formal normal conjuntiva: SAT_{FNC}:

- A fórmula é composta por conjunção de cláusulas (cláusulas relacionadas pelo operador $\wedge$).
- Cada cláusula é composta por disjunção de literais (literais relacionados pelo operador $\vee$).
- Cada literal é uma variável ou a negação de uma variável.

Exemplo: $(\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee z)$.



Demonstração

Estratégia a seguir:

1. Provar que $\text{SAT}_{FNC} \in \text{NP}$.
2. Encontrar um problema NP-completo Π.
3. Provar que $\Pi \preccurlyeq_p \text{SAT}_{FNC}$.

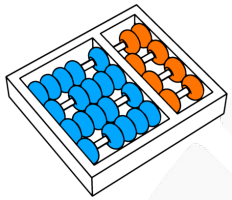
SAT_{FNC} é um caso particular de SAT, que está em NP, portanto SAT_{FNC} também está.

Sabemos que o seguinte problema é NP-completo:

Problema (Satisfatibilidade de circuito (C-SAT))

$$\textit{C-SAT} = \{\langle c \rangle : c \textit{ é um circuito lógico satisfazível}\}$$

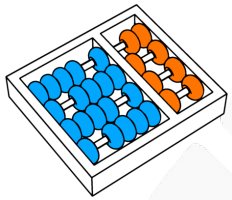
Basta provar que $\text{C-SAT} \preccurlyeq_p \text{SAT}_{FNC}$.



C-SAT $\preccurlyeq_p$ SAT$_{FNC}$

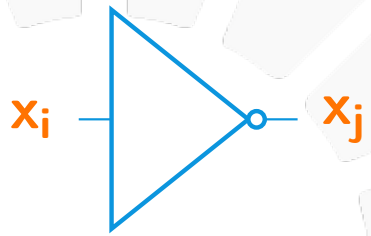
Dado um circuito lógico C, definimos a seguinte fórmula booleana em forma normal conjuntiva $F(C)$:

1. Por cada fio **i** de C, definimos a variável $\mathbf{x_i}$.
2. Por cada porta lógica de C, definimos cláusulas que garantem **equivalência** lógica com a relação entre as entradas da porta e suas saídas.
3. Finalmente, identificamos por **o** o fio de saída de C.

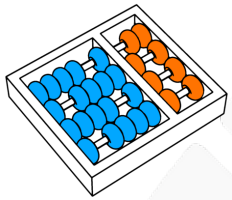


C-SAT $\preceq_p$ SAT$_{FNC}$

Por cada porta **NOT** de C definimos duas cláusulas em $F(C)$:

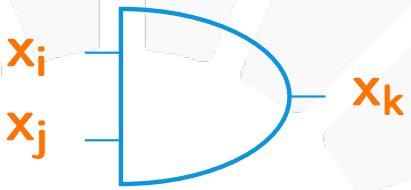


$$\neg x_i \Leftrightarrow x_j$$
$$\equiv \quad (\neg x_i \vee \neg x_j) \wedge (x_i \vee x_j)$$



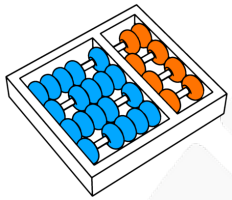
C-SAT $\preceq_p$ SAT$_{FNC}$

Por cada porta **AND** de C definimos três cláusulas em $F(C)$:



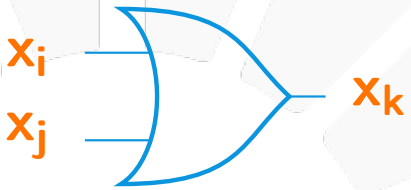
$$x_i \wedge x_j \Leftrightarrow x_k$$
$$\equiv \quad (\neg x_i \vee \neg x_j \vee x_k) \wedge (x_i \vee \neg x_k) \wedge (x_j \vee \neg x_k)$$

Se uma porta **AND** tiver m fios de entrada, então definimos $m+1$ cláusulas.



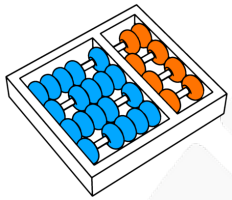
C-SAT $\preceq_p$ SAT$_{FNC}$

Por cada porta **OR** de C definimos três cláusulas em $F(C)$:



$$\begin{aligned} & x_i \vee x_j \Leftrightarrow x_k \\ \equiv \quad & (x_i \vee x_j \vee \neg x_k) \wedge (\neg x_i \vee x_k) \wedge (\neg x_j \vee x_k) \end{aligned}$$

Se uma porta **OR** tiver m fios de entrada, então definimos $m + 1$ cláusulas.



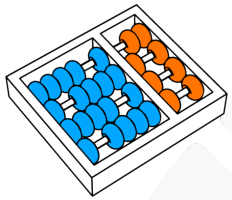
C-SAT $\preccurlyeq_p$ SAT$_{FNC}$

Dado um circuito lógico C, $F(C)$ será da forma:

$$\mathbf{x_o} \wedge \psi_1 \wedge \psi_2 \wedge \ldots \wedge \psi_m$$

Onde $\mathbf{x_o}$ corresponde ao fio de saída o de C e cada fórmula ψ_k corresponde às cláusulas em forma normal conjuntiva geradas pela porta k de C. Portanto, C é satisfazível se e somente se $F(C)$ for satisfazível.

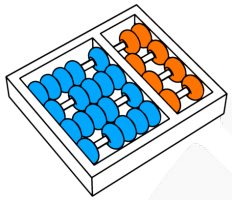
O número de variáveis em $F(C)$ é igual ao número de fios em C e o número de cláusulas não é maior que o número de fios multiplicado pelo número de portas em C. Portanto, $F(C)$ pode ser construída em tempo polinomial a partir de C.



Teoremas

Teorema

SAT_{FNC} *é* NP*-completo.*



3-SAT

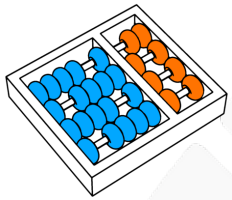
Outro caso especial de SAT é quando a fórmula está em forma normal conjuntiva e cada cláusula tem exatamente três literais.

Teorema

3-SAT é NP*-completo.*

Por ser um caso particular de SAT, sabemos que 3-SAT está em NP.

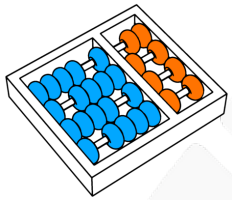
Basta provar que é NP-difícil.



$\text{SAT}_{FNC} \preceq_p$ 3-SAT

Considere uma fórmula ψ em forma normal conjuntiva, construímos a fórmula $F(\psi)$ em forma normal conjuntiva com exatamente três literais por cláusula, tal que $\psi \in \text{SAT}_{FNC}$ se e somente se $F(\psi) \in$ 3-SAT.

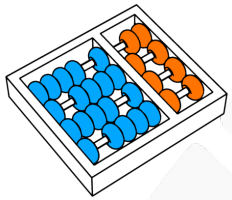
Para construir $F(\psi)$, cada cláusula de ψ com número de literais diferentes de três será substituída por novas cláusulas, podendo adicionar novas variáveis.



$\text{SAT}_{FNC} \preceq_p$ 3-SAT

Cada cláusula c ψ com somente um literal ($c = \ell$), é substituída por quatro novas cláusulas que adicionam duas novas variáveis ($x_{c,1}$ e $x_{c,2}$):

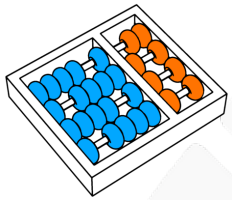
$$(\ell \vee x_{c,1} \vee x_{c,2}) \wedge (\ell \vee x_{c,1} \vee \neg x_{c,2}) \wedge (\ell \vee \neg x_{c,1} \vee x_{c,2}) \wedge (\ell \vee \neg x_{c,1} \vee \neg x_{c,2})$$



$\text{SAT}_{FNC} \preceq_p$ 3-SAT

Cada cláusula c de ψ com somente dois literais ($c = \ell_1 \vee \ell_2$), é substituída por dois novas cláusulas que adicionam uma nova variável (x_c):

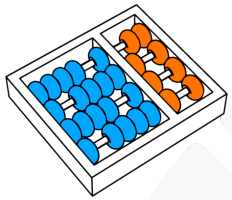
$$(\ell_1 \vee \ell_2 \vee x_c) \wedge (\ell_1 \vee \ell_2 \vee \neg x_c)$$



$\text{SAT}_{FNC} \preceq_p$ 3-SAT

Cada cláusula c de ψ com $k > 3$ literais
$(c = \ell_1 \vee \ell_2 \vee \ell_3 \vee \ldots \vee \ell_k)$ é substituída por $k - 2$ novas cláusulas que adicionam $k - 3$ novas variáveis $(x_{c,1}, x_{c,2}, \ldots, x_{c,k-3})$:

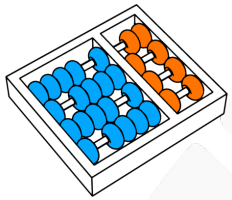
$$(\ell_1 \vee \ell_2 \vee x_{c,1}) \wedge (\neg x_{c,1} \vee \ell_3 \vee x_{c,2}) \wedge (\neg x_{c,2} \vee \ell_4 \vee x_{c,3})$$
$$\wedge \ldots \wedge (\neg x_{c,i-2} \vee \ell_i \vee x_{c,i-1}) \wedge \ldots$$
$$\wedge (\neg x_{c,k-4} \vee \ell_{k-2} \vee x_{c,k-3}) \wedge (\neg x_{c,k-3} \vee \ell_{k-1} \vee \ell_k)$$



$\text{SAT}_{FNC} \preceq_p$ 3-SAT

Dada uma fórmula booleana ψ em forma normal conjuntiva, a fórmula booleana $F(\psi)$ em forma normal conjuntiva com exatamente três literais por cláusula garante que $\psi \in \text{SAT}_{FNC}$ se e somente se $F(\psi) \in$ 3-SAT.

Por cada cláusula c de ψ o número de novas cláusulas e variáveis em $F(\psi)$ é no máximo 4 ou o número de literais em c. Portanto, a construção de $F(\psi)$ pode ser feita em tempo polinomial no tamanho de ψ.



Um problema em grafos

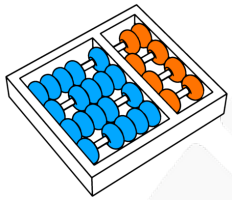
Uma **clique** de um grafo G é um subgrafo completo de G. Ou seja, um subgrafo de G com uma aresta entre cada par de vértices.

Problema (Clique)

$$CLIQUE = \{\langle G, k\rangle : G \text{ é um grafo com uma clique de } k \text{ vértices}\}$$

Teorema

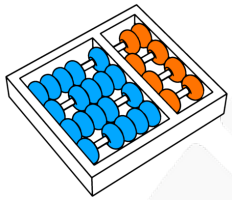
CLIQUE é NP*-completo.*



CLIQUE $\in$ NP

Algoritmo: VERIFICA-CLIQUE($\langle G, k \rangle, \langle C \rangle$)

```
1 se C não for subgrafo de G
2     devolva NAO
3 se C não for uma clique
4     devolva NAO
5 se C não tiver k vértices
6     devolva NAO
7 devolva SIM
```

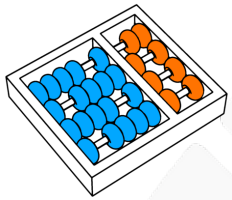


3-SAT $\preceq_p$ CLIQUE

Dada uma fórmula booleana ψ em forma normal conjuntiva com m cláusulas e exatamente três literais por cláusula, construímos uma instância $\langle G, k \rangle = F(\psi)$ da CLIQUE como segue:

- Por cada cláusula $c = \ell_1 \vee \ell_2 \vee \ell_3$ de ψ, definimos um **cluster** em G que consiste em três vértices (um por cada literal de c): v_{c,ℓ_1}, v_{c,ℓ_2} e v_{c,ℓ_3}.
- Adicionamos uma aresta $(v_{c,\ell}, v_{c',\neg\ell'})$ entre cada par de vértices $v_{c,\ell}$ e $v_{c',\ell'}$ em clusters diferentes ($c \neq c'$), a menos que um dos literais associados seja a negação do outro ($\ell = \neg\ell'$).
- Não adicionamos arestas entre vértices do mesmo cluster.
- Finalmente, definimos o parâmetro k como o número de cláusulas ($k = m$).

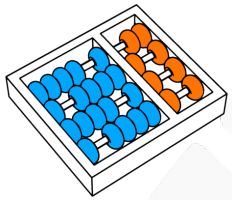
Note que $\langle G, k \rangle$ pode ser construída em tempo polinomial no tamanho de ψ.



3-SAT $\preceq_p$ CLIQUE

Antes de provar que $\langle \psi \rangle \in$ 3-SAT se e somente se $\langle G, m \rangle \in$ CLIQUE, consideremos as seguintes propriedade:

- Se dois vértices $v_{c,\ell}$ e $v_{c',\ell'}$ em G são adjacentes, então os literais associados podem ser simultaneamente verdadeiros $(\ell = \ell' = 1)$.
- Se dois litareis ℓ e ℓ' de cláusulas diferentes c e c' $(c \neq c')$ podem ser simultaneamente verdadeiros, então os vértices associados $v_{c,\ell}$ e $v_{c',\ell'}$ são adjacentes.

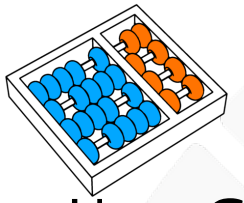


3-SAT $\preceq_p$ CLIQUE

$\langle \psi \rangle \in$ 3-SAT se e somente se $\langle G, m \rangle \in$ CLIQUE:

$\Rightarrow$ Se ψ é satisfazível, então denote por α uma atribuição de valores que a satisfaz. Por cada cláusula, selecione um literal que faz essa cláusula verdadeira com a atribuição α. Os m vértices associados aos literais selecionados são adjacentes em G, portanto formam uma clique de m vértices.

$\Leftarrow$ Se G tem uma clique com m vértices, então há um vértice de cada cluster nela (vértices do mesmo cluster não podem estar na mesma clique). Como todos os vértices da clique são adjacentes, significa que os literais associados podem ser simultaneamente verdadeiros e como cada literal associado corresponde exatamente a uma das m cláusulas de ψ, temos que a fórmula é satisfazível.



Cobertura por vértices

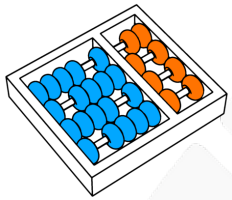
Uma **COBERTURA POR VÉRTICE** (*vertex cover*) de um grafo não direcionado $G = (V, E)$ é um subconjunto de vértices $V' \subseteq V$ tal que qualquer aresta do grafo é incidente a pelo menos um vértice em V'.

Problema da cobertura por vértices mínima:

- É a tarefa de achar uma cobertura de menor cardinalidade.
- A versão de decisão é saber se há cobertura de tamanho k.

Problema (Cobertura por vértices)

$$\textit{VERTEX-COVER} = \{\langle G, k\rangle : G \textit{ é um grafo que possui uma cobertura por vértices de tamanho } k\}$$



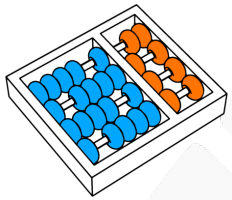
Cobertura por vértices está em NP

Lema

VERTEX-COVER está em NP*.*

Demonstração:

- Exercício.

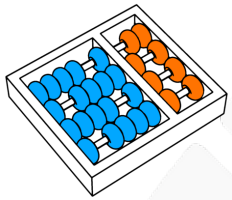


Cobertura por vértices é NP-difícil

Lema

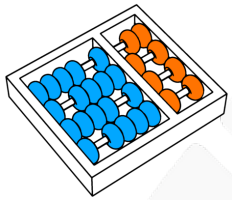
VERTEX-COVER é NP*-difícil.*

- Reduziremos CLIQUE para VERTEX-COVER.
- Dado grafo $G = (V, E)$, criamos o complemento $\overline{G} = (V, \overline{E})$.
- Vamos mostrar que G terá uma clique de tamanho k **sse** $\overline{G}$ tiver uma cobertura por vértices de tamanho $|V| - k$.



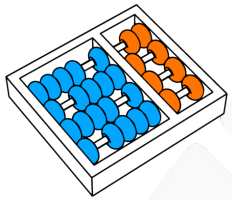
Demonstração

1. Suponha que G tem uma clique C de tamanho k:
   - Defina o conjunto $V' = V \setminus C$.
   - Mostraremos que V' é uma cobertura por vértices de $\overline{G}$:
     - Considere dois vértices $u, v \in C$.
     - Como C é clique de G, (u, v) é uma aresta de G.
     - Então, (u, v) **não** é uma aresta de $\overline{G}$.
     - Isso significa que $\overline{G}[C] = \overline{G} - V'$ não tem arestas.
     - Portanto, V' é uma cobertura por vértices de $\overline{G}$.



Demonstração (cont)

2. Suponha que $\overline{G}$ tem cobertura V' de tamanho $|V| - k$:
   - Defina o conjunto $C = V \setminus V'$.
   - Mostraremos que C é uma clique em G:
     - Considere dois vértices $u, v \in C$.
     - Como V' é cobertura de $\overline{G}$, (u, v) **não** é uma aresta de G.
     - Então, (u, v) é uma aresta em G.
     - Portanto, C é uma clique em G de tamanho k.



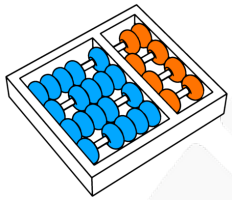
Soma de subconjunto

Problema da soma de subconjunto

- **Entrada:** um conjunto de naturais S e um natural t
- **Pergunta:** existe subconjunto $S' \subseteq S$ tal que $\sum_{s \in S'} s = t$?

Problema (Soma de subconjunto)

$$SUBSET\text{-}SUM = \{\langle S, t\rangle : \textit{existe subconjunto } S' \subseteq S \textit{ tal que } \textstyle\sum_{s \in S'} s = t\}$$

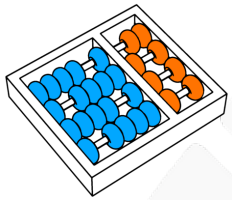


Exemplo

Exemplo de instância

- Sejam $S = \{1, 4, 10, 14, 54, 100, 1004, 1003\}$ e $t = 1027$.
- Nesse caso, $S' = \{10, 14, 1003\}$ é uma solução.
- Então, $\langle S, t \rangle$ é uma instância SIMdo problema.

Vamos mostrar que SUBSET-SUM é NP-completo.



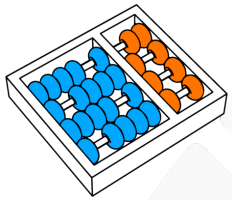
Soma de subconjunto está em NP

Lema

SUBSET-SUM está em NP*.*

Demonstração:

- Exercício.

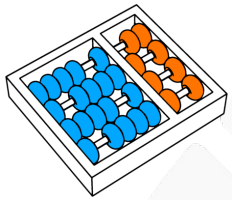


Soma de subconjunto é NP-difícil

Teorema

SUBSET-SUM é NP*-difícil.*

- Reduziremos 3CNF-SAT para SUBSET-SUM.
- Recebemos uma fórmula f com variáveis $x_1, \ldots, x_n$ e cláusulas $C_1, \ldots, C_k$ com 3 literais cada.
- Sem perda de generalidade, supomos que uma variável e sua negação não aparecem na mesma cláusula.



Redução

Criamos uma série de números naturais:

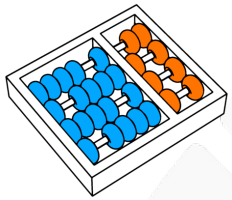
- Cada um tem $n + k$ dígitos **decimais**.
- Cada dígito corresponde a uma variável ou uma cláusula.
- Os dígitos estão em ordem $x_1, x_2, \ldots, x_n, C_1, \ldots, C_k$.

Conjunto de números S:

- Criamos **dois** números para cada variável e cada cláusula.

Número alvo t:

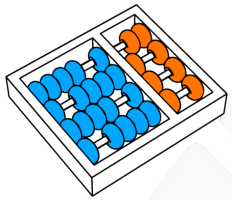
- Criamos **um** número especial.



Exemplo de instância reduzida

$$f = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

	x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	1	0	0	1	0	0	1
v_1'	1	0	0	0	1	1	0
v_2	0	1	0	0	0	0	1
v_2'	0	1	0	1	1	1	0
v_3	0	0	1	0	0	1	1
v_3'	0	0	1	1	1	0	0
s_1	0	0	0	1	0	0	0
s_1'	0	0	0	2	0	0	0
s_2	0	0	0	0	1	0	0
s_2'	0	0	0	0	2	0	0
s_3	0	0	0	0	0	1	0
s_3'	0	0	0	0	0	2	0
s_4	0	0	0	0	0	0	1
s_4'	0	0	0	0	0	0	2



Redução

Para uma variável x_i:

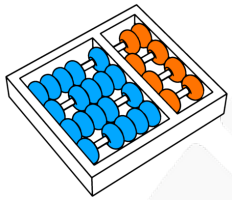
1. Criamos um número v_i com um 1 no dígito x_i e um 1 em cada dígito C_j tal que x_i aparece em C_j.
2. Também um número v_i' com 1 no dígito x_i e um 1 em cada dígito C_i tal que $\neg x_i$ aparece em C_i.

Para cada cláusula C_i:

1. Criamos um número s_i com um 1 no dígito C_i.
2. Também um número s_i' com um 2 no dígito C_i.

Para o número t:

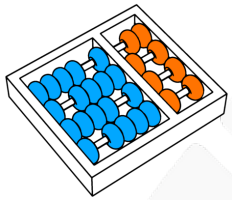
1. Criamos um número com um 1 em cada dígito x_i e um 4 em cada dígito C_i.



Exemplo de instância reduzida

$$f = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

	x_1	x_2	x_3	C_1	C_2	C_3	C_4
v_1	1	0	0	1	0	0	1
v_1'	1	0	0	0	1	1	0
v_2	0	1	0	0	0	0	1
v_2'	0	1	0	1	1	1	0
v_3	0	0	1	0	0	1	1
v_3'	0	0	1	1	1	0	0
s_1	0	0	0	1	0	0	0
s_1'	0	0	0	2	0	0	0
s_2	0	0	0	0	1	0	0
s_2'	0	0	0	0	2	0	0
s_3	0	0	0	0	0	1	0
s_3'	0	0	0	0	0	2	0
s_4	0	0	0	0	0	0	1
s_4'	0	0	0	0	0	0	2



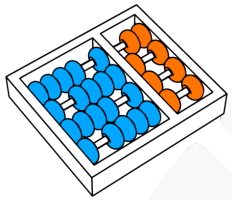
Demonstração

Propriedades da instância reduzida:

- Todos os números são distintos, pois uma cláusula não tem uma variável e sua negação.
- Os dígitos $x_1, \ldots, x_n$ são diferentes para todos os números correspondentes a variáveis ou cláusulas.

Se somarmos todos os números de S

- O maior valor que um dígito pode atingir é 6, pois um dígito C_j corresponde a um cláusula com 3 literais.
- Então, a soma em um dígito não interfere na soma de outro dígito mais significativo.



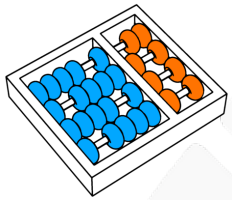
Demonstração

A redução leva tempo polinomial.

- Além de t, o conjunto S tem $2(n+k)$ números decimais.
- Cada um dos números tem $n+k$ dígitos.

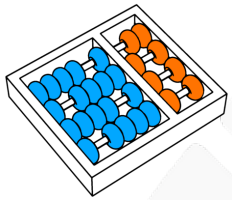
Vamos mostrar a equivalência das instâncias:

$$f \text{ é satisfazível } \textbf{sse} \text{ existe } S' \subseteq S \text{ tal que } \sum_{s \in S'} s = t$$



Demonstração

1. Suponha que f é satisfazível:
   - Seja $x_1, \ldots, x_n$ uma atribuição de variáveis que satisfaça f.
   - Vamos criar um subconjunto $S' \subseteq S$.
   - Para cada variável x_i:
     - Se $x_i = 1$, adicione v_i ao conjunto S'.
     - Se $x_i = 0$, adicione v_i' ao conjunto S'.
   - Para cada cláusula C_j:
     - Se todos três literais de C_j forem satisfeitos, adicione s_j a S'.
     - Se apenas dois literais forem satisfeitos, adicione s_j' a S'.
     - Se apenas um literal for satisfeito, adicione s_j e s_j' a S'.
   - Defina $t' = \sum_{s \in S'} s$.
   - Cada dígito x_i de t' vale 1, pois v_i ou v_i' está em S'.
   - Cada dígito C_j de t' vale 4 pela forma que inserimos s_i e s_i'.
   - Concluímos que $t' = t$ e a instância reduzida é SIM.



Demonstração

2. Suponha que existe $S' \subseteq S$ tal que $\sum_{s \in S'} s = t$:
   - Relembre que temos 1 no dígito x_i de t.
   - Então, exatamente um número de v_i e v'_i está em S'.
   - Isso induz uma atribuição de cada variável x_i:
     - Se $v_i \in S'$, então faça $x_i = 1$.
     - Se $v'_i \in S'$, então faça $x_i = 0$.
   - Também, temos 4 no dígito C_j de t.
   - Assim há pelo menos um número em S' que tem 1 em C_j.
   - Esse número corresponde a um literal de C_j.
   - Então, a atribuição induzida satisfaz cada cláusula.
   - Portanto f é satisfazível.

NP-completude

MO417 - Complexidade de Algoritmos I

06/24

26

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

UNICAMP