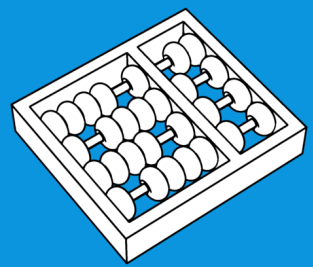
NP-completude e reduções

MC558 - Projeto e Análise de Algoritmos II

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

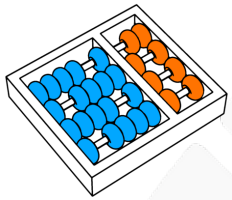
UNICAMP



"Se cientistas da computação fossem físicos, simplesmente teríamos declarado que $\mathsf{P} \neq \mathsf{NP}$ *é uma lei da Natureza, análoga às leis da termodinâmica. Um Prêmio Nobel seria concedido pela descoberta dessa lei. (E, no improvável caso de alguém provar mais tarde que* $\mathsf{P} = \mathsf{NP}$*, um segundo Prêmio Nobel seria concedido por provar que ela é falsa.)."*

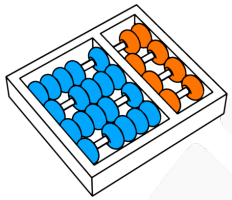
Scott Aaronson.

Como provar pertença a NP?



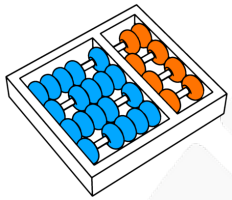
Duas opções

- Propor um verificador polinomial.



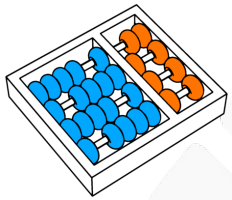
Duas opções

- Propor um verificador polinomial.
- Propor um algoritmo não-determinístico polinomial.



Algoritmos não-determinísticos

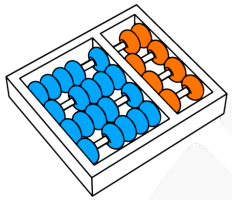
Um algoritmo **não-determinístico** é um algoritmo que para uma mesma entrada pode dar saídas diferentes.



Algoritmos não-determinísticos

Um algoritmo **não-determinístico** é um algoritmo que para uma mesma entrada pode dar saídas diferentes.

Algoritmos não-determinísticos possuem os mesmos elementos de algoritmos determinísticos.

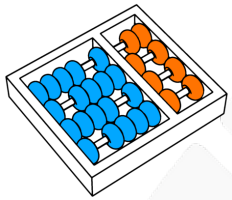


Algoritmos não-determinísticos

Um algoritmo **não-determinístico** é um algoritmo que para uma mesma entrada pode dar saídas diferentes.

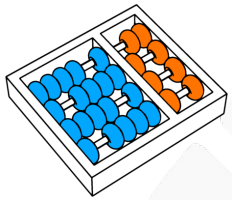
Algoritmos não-determinísticos possuem os mesmos elementos de algoritmos determinísticos.

Eles adicionam instruções de **"chute"**.



Algoritmos não-determinísticos

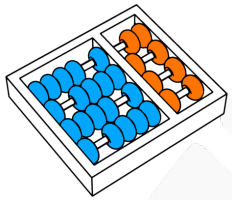
Dizemos que um algoritmo não-determinístico $A(x)$ aceita uma linguagem L se:



Algoritmos não-determinísticos

Dizemos que um algoritmo não-determinístico $A(x)$ aceita uma linguagem L se:

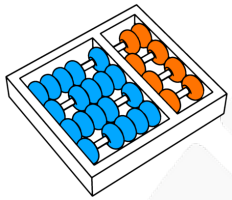
- Para cada $x \in L$, existe uma execução em que $A(x) = 1$.



Algoritmos não-determinísticos

Dizemos que um algoritmo não-determinístico $A(x)$ aceita uma linguagem L se:

- Para cada $x \in L$, existe uma execução em que $A(x) = 1$.
- Para cada $x \notin L$, não existe uma execução tal que $A(x) = 1$ (toda execução de $A(x)$ cicla ou devolve 0)

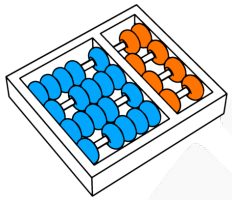


Algoritmos não-determinísticos

Dizemos que um algoritmo não-determinístico $A(x)$ aceita uma linguagem L se:

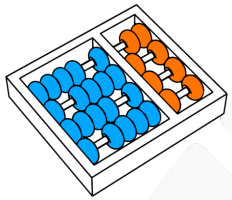
- Para cada $x \in L$, existe uma execução em que $A(x) = 1$.
- Para cada $x \notin L$, não existe uma execução tal que $A(x) = 1$ (toda execução de $A(x)$ cicla ou devolve 0)

Para o caso de decidir L a definição é análoga aos algoritmos determinísticos.



Algoritmos não-determinísticos

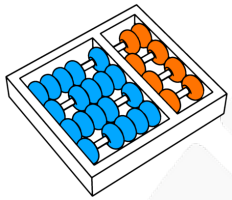
Dizemos que um algoritmo não-determinístico $A(x)$ decide uma linguagem L em tempo polinomial se decide A e qualquer execução de $A(x)$ requer no máximo $O(|x|^k)$ operações, para alguma constante k.



NP

Teorema

NP *é o conjunto de linguagens que são decidíveis por algoritmos polinomiais não-determinísticos.*



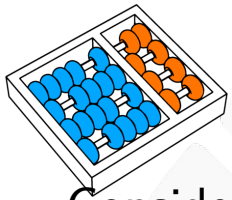
NP

Teorema

NP *é o conjunto de linguagens que são decidíveis por algoritmos polinomiais não-determinísticos.*

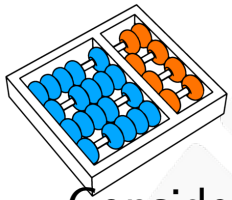
Demonstração discutida em aula.

Um esquema do teorema de Cook-Levin



Circuito lógico

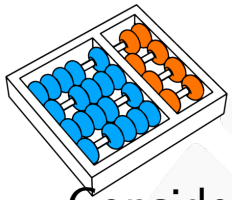
Considere um **CIRCUITO LÓGICO**:



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

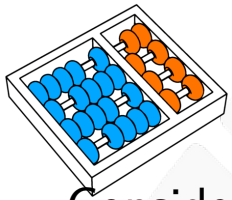
- Contém n fios de entrada.



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

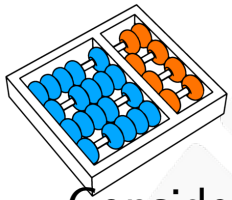
- Contém n fios de entrada.
- É formado combinando portas lógicas:



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

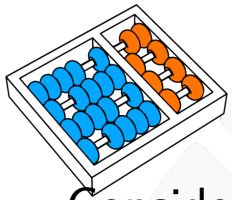
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

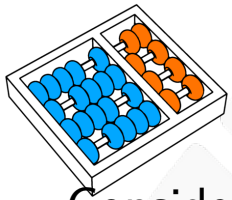
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

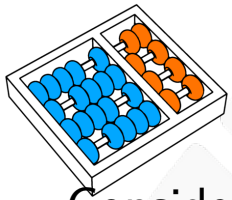
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).



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.

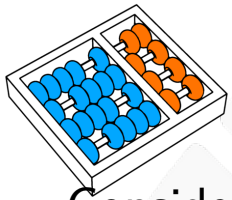


Circuito lógico

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.



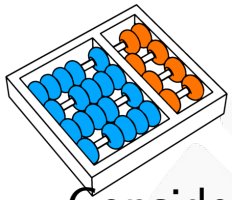
Circuito lógico

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.



Circuito lógico

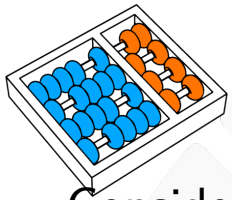
Considere um **CIRCUITO LÓGICO**:

- 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))



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

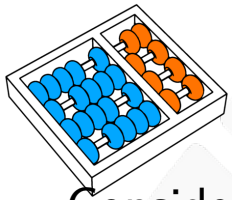
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

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.*



Circuito lógico

Considere um **CIRCUITO LÓGICO**:

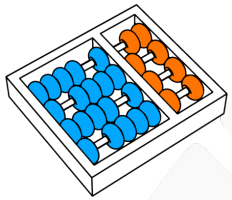
- Contém n fios de entrada.
- É formado combinando portas lógicas:
  - NOT ($\neg$).
  - AND ($\wedge$).
  - OR ($\vee$).
- A saída do circuito é dada por um fio.

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

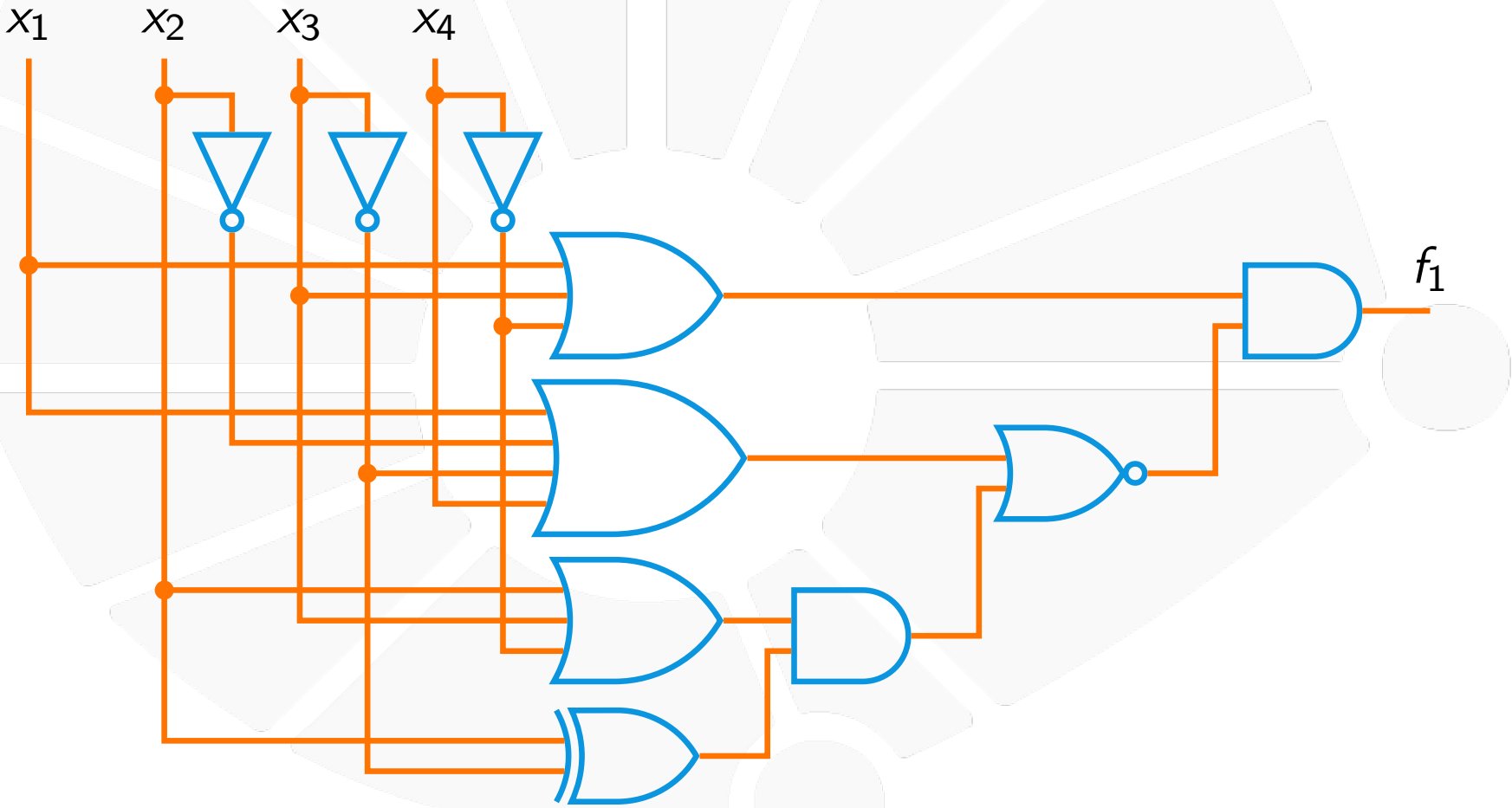
- Um circuito é **SATISFAZÍVEL** se ele possuir uma atribuição que resulta em bit 1 no fio de saída.

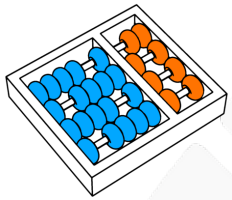
Problema (Satisfatibilidade de circuito (C-SAT))

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



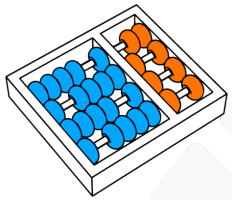
Exemplo de circuito





Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

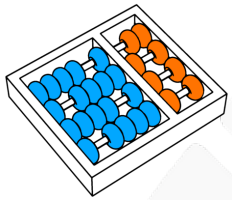


Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

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



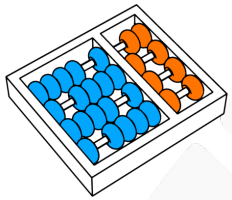
Linguagem correspondente

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 $O(2^n(n+m))$:



Linguagem correspondente

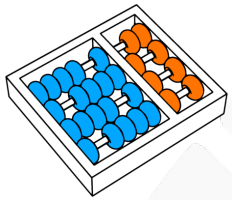
A linguagem C-SAT contém circuitos lógicos satisfatíveis.

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.



Linguagem correspondente

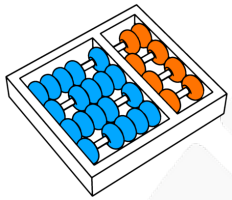
A linguagem C-SAT contém circuitos lógicos satisfatíveis.

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.



Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

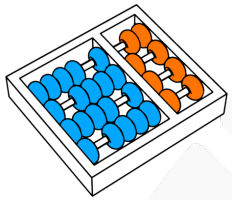
$$\textit{C-SAT} = \{\langle C \rangle : C \textit{ é um circuito lógico satisfazível}\}.$$

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*.*



Linguagem correspondente

A linguagem C-SAT contém circuitos lógicos satisfatíveis.

Problema (Satisfatibilidade de circuito)

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

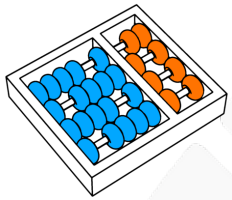
Há um algoritmo simples de tempo $O(2^n(n+m))$:

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

Lema

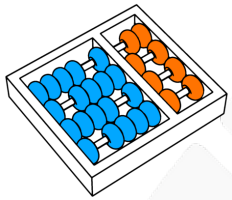
C-SAT é NP.

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



Redução de NP para circuitos

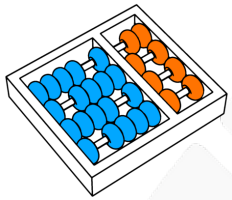
Queremos mostrar que C-SAT é NP-completo:



Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

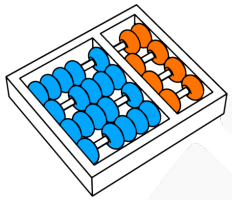
- Falta mostrar então que C-SAT é NP-difícil.



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 $Q \preccurlyeq_p \text{C-SAT}$.

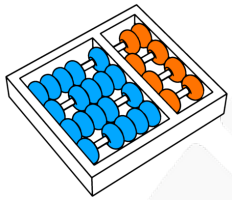


Redução de NP para circuitos

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:



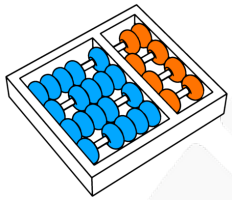
Redução de NP para circuitos

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



Redução de NP para circuitos

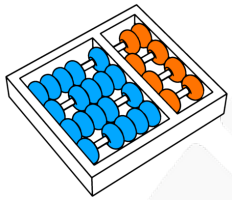
Queremos mostrar que C-SAT é NP-completo:

- 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.



Redução de NP para circuitos

Queremos mostrar que C-SAT é NP-completo:

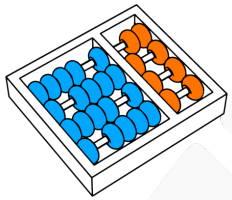
- Falta mostrar então que C-SAT é NP-difícil.
- Queremos que para todo $Q \in \text{NP}$, tenhamos $Q \preccurlyeq_p \text{C-SAT}$.

Nosso objetivo é projetar uma redução polinomial F:

Problema

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

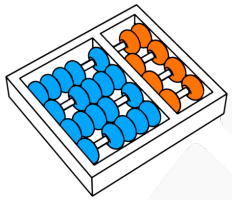
$$x \in Q \quad \textbf{\textit{se e somente se}} \quad F(x) \in \textit{C-SAT}.$$



NP-dificuldade

Lema

C-SAT é NP*-difícil.*

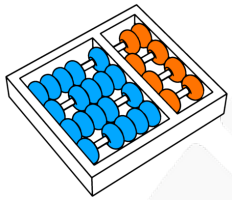


NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:



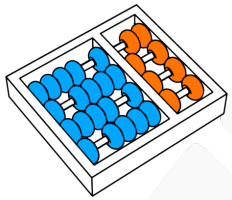
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in$ NP.



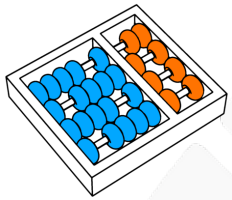
NP-dificuldade

Lema

C-SAT é NP-*difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in \text{NP}$.
- Existe um algoritmo verificador polinomial A para Q.



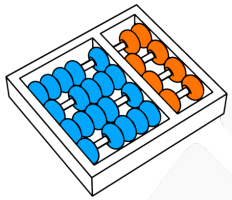
NP-dificuldade

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.



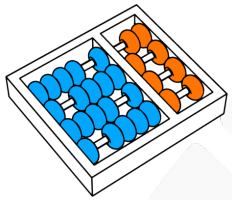
NP-dificuldade

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:



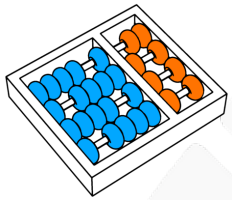
NP-dificuldade

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 $|x| = n$.



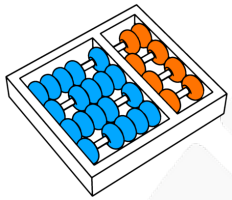
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

Demonstração:

- Considere um problema arbitrário $Q \in \text{NP}$.
- 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.



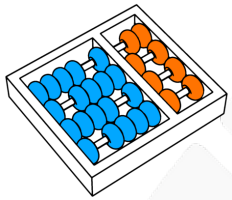
NP-dificuldade

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 $|y|$.



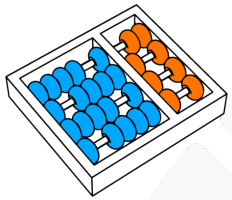
NP-dificuldade

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.



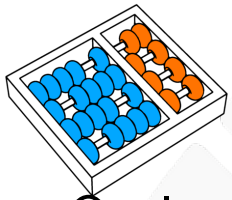
NP-dificuldade

Lema

C-SAT é NP*-difícil.*

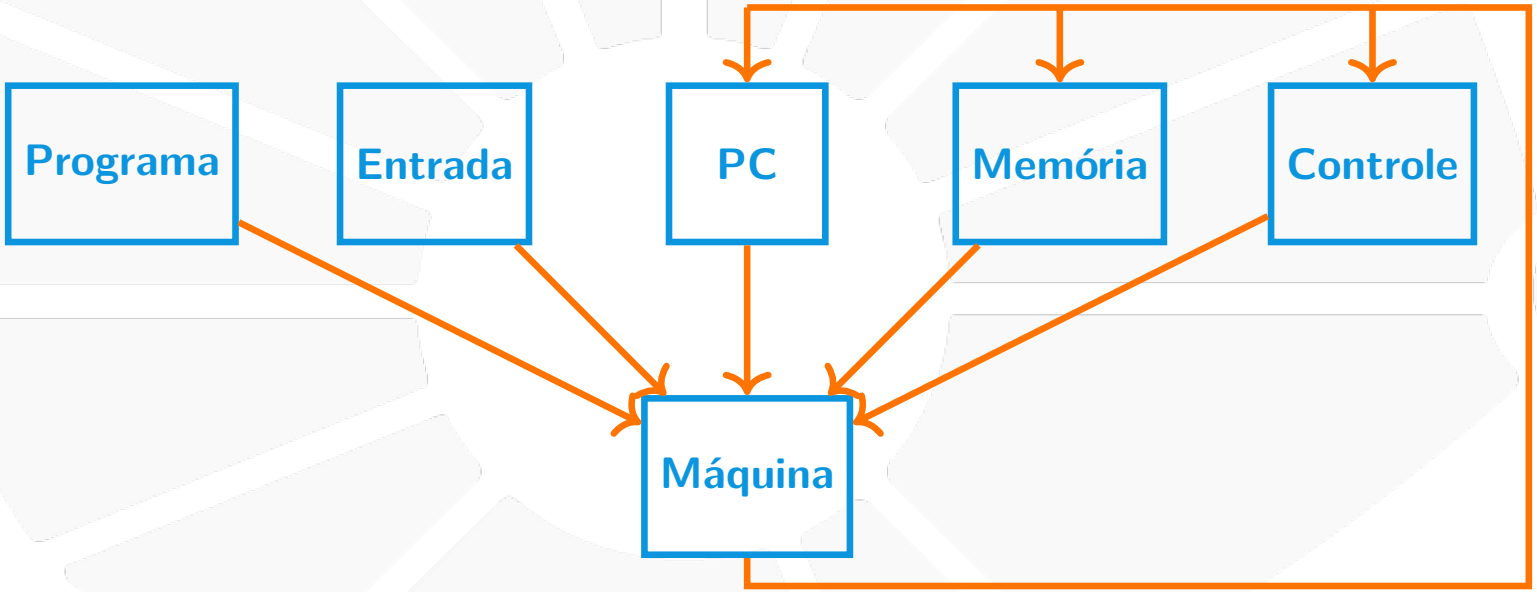
Demonstração:

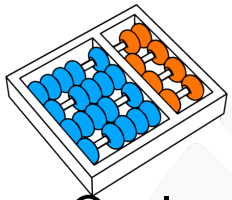
- Considere um problema arbitrário $Q \in$ NP.
- Existe um algoritmo verificador polinomial A para Q.
- Dada uma instância x de Q, criaremos a instância $F(x)$ de C-SAT.
- O algoritmo verificador de Q recebe duas strings, x e y:
  - A entrada x tem tamanho $|x| = n$.
  - O certificado y tem tamanho $|y| = n^{k'}$, 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

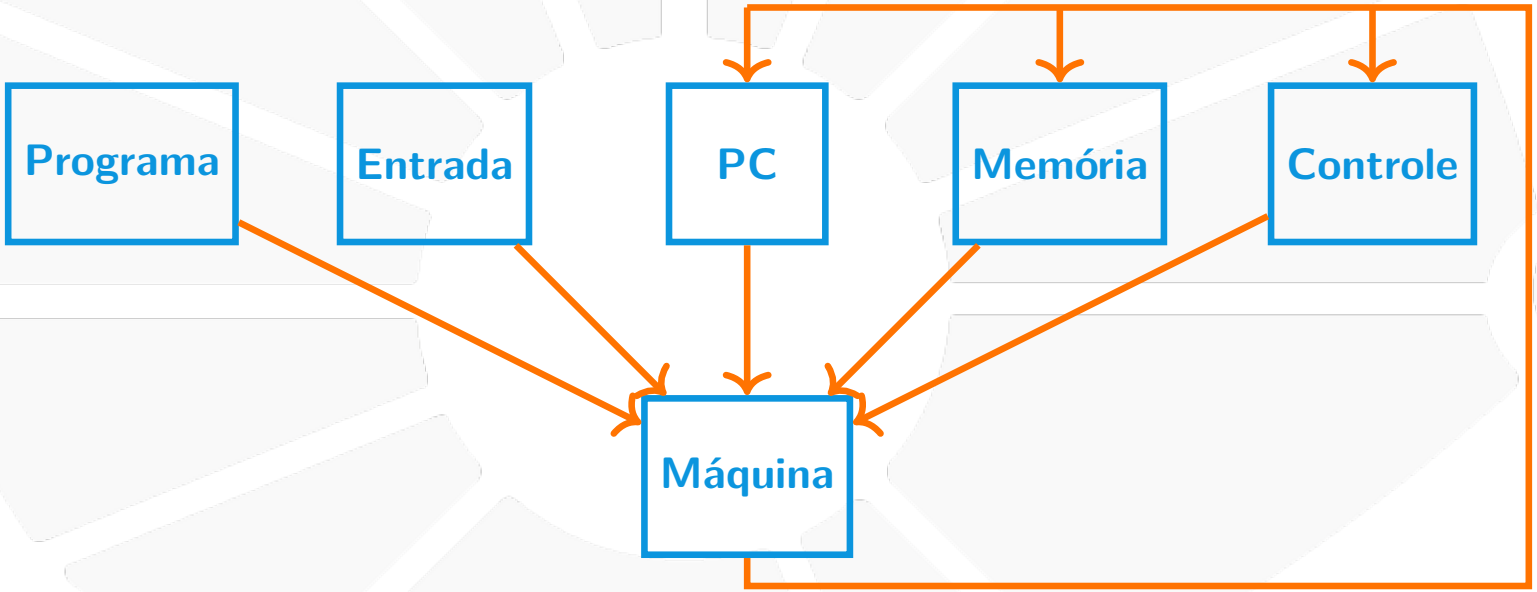
O algoritmo A pode ser implementado por um computador de circuitos lógicos.



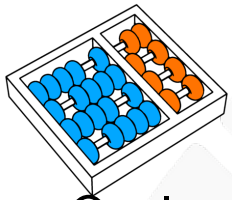


Continuação da demonstração

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

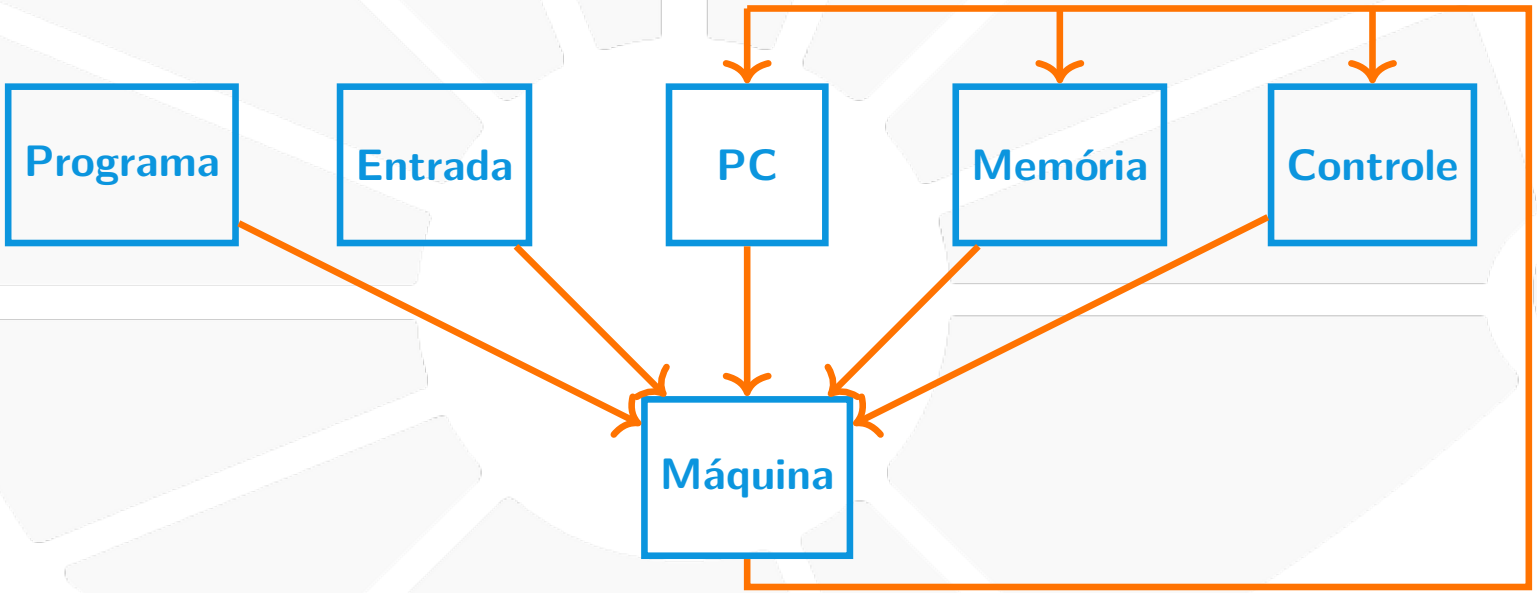


- Os circuitos têm **RETROALIMENTAÇÃO**.

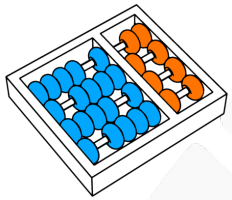


Continuação da demonstração

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

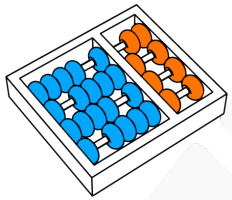


- Os circuitos têm **RETROALIMENTAÇÃO**.
- Após executar uma instrução, a memória é modificada



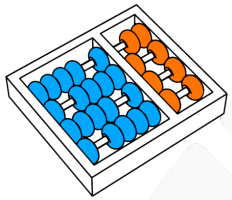
Continuação da demonstração

- Ajustamos o algoritmo A para que a saída seja escrita em um bit específico da memória denotado por C.



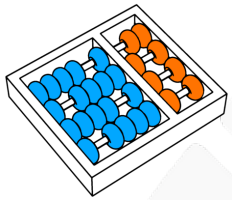
Continuação da demonstração

- 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:



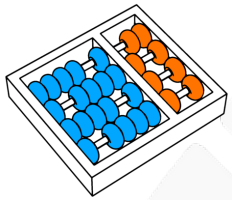
Continuação da demonstração

- 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.



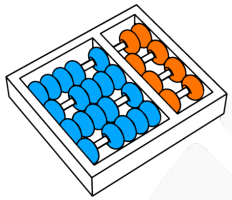
Continuação da demonstração

- 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.



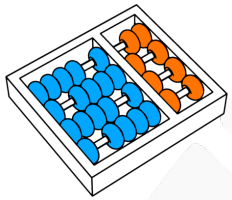
Continuação da demonstração

- 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:



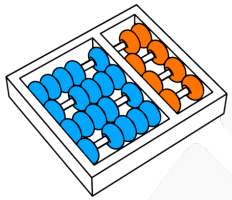
Continuação da demonstração

- 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.



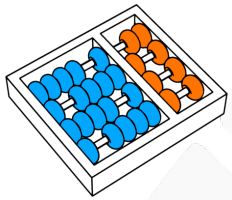
Continuação da demonstração

- 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.

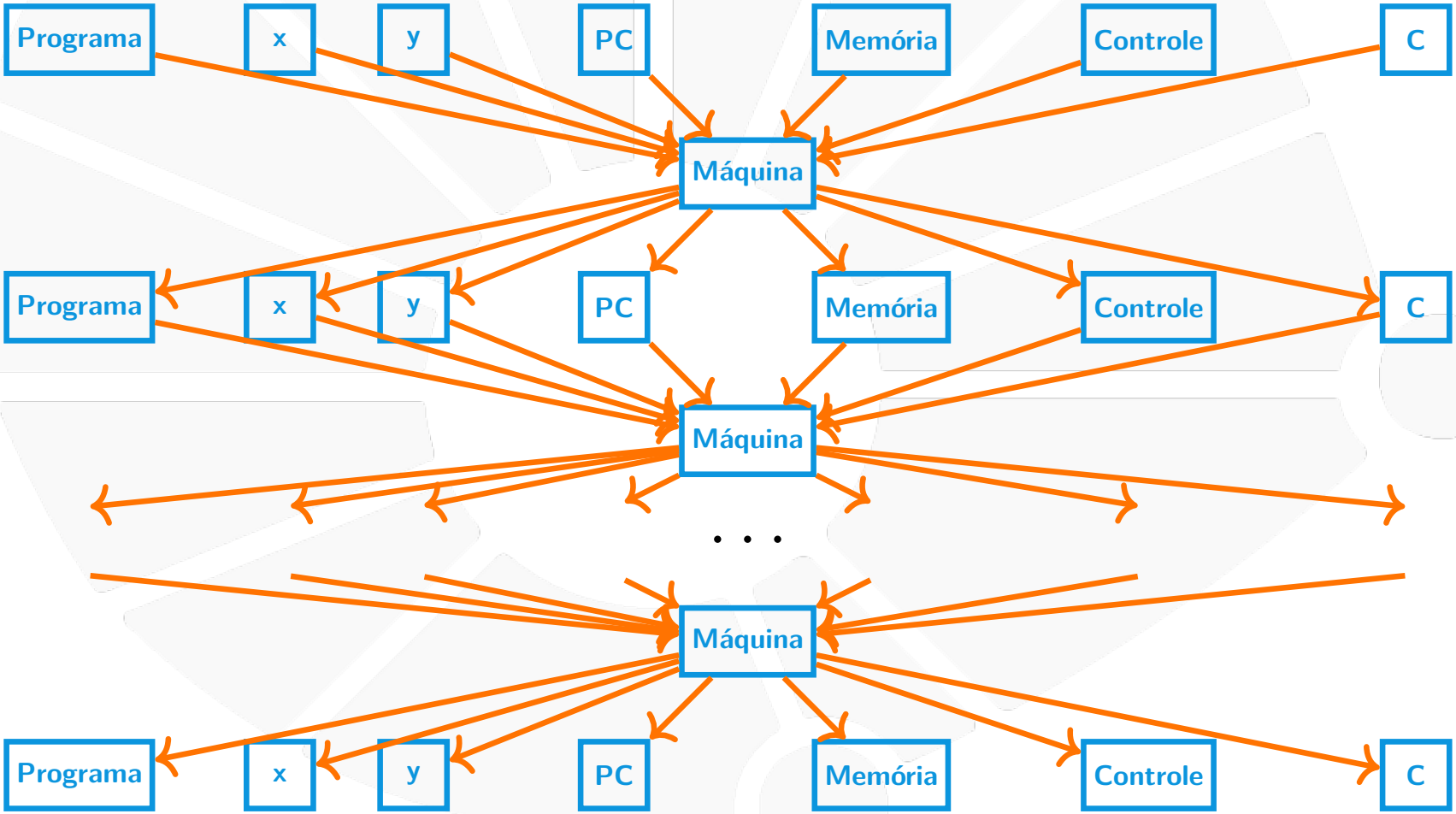


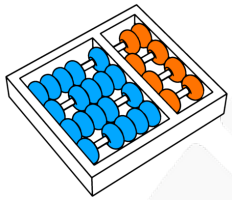
Continuação da demonstração

- 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.



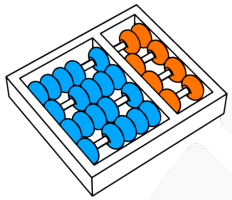
Continuação da demonstração





Continuação da demonstração

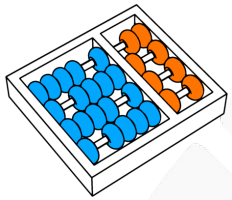
O circuito construído tem tamanho polinomial.



Continuação da demonstração

O circuito construído tem tamanho polinomial.

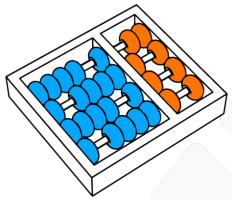
- O tamanho da máquina, controle, PC e A independem de x.



Continuação da demonstração

O circuito construído tem tamanho polinomial.

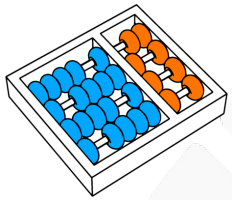
- O tamanho da máquina, controle, PC e A independem de x.
- A memória, y e x têm tamanho polinomial em $|x|$.



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.

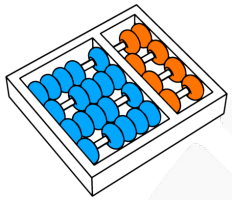


Continuação da demonstração

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:



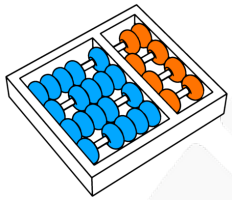
Continuação da demonstração

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.



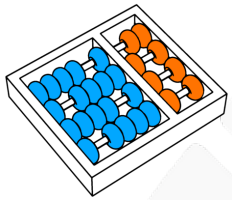
Continuação da demonstração

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.



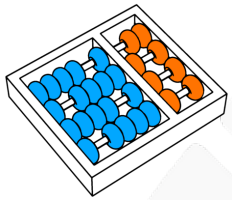
Continuação da demonstração

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.



Continuação da demonstração

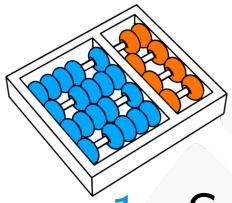
O circuito construído tem tamanho polinomial.

- 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:

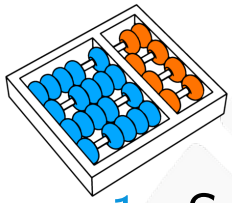
- Todo circuito é fixo e pode ser construído a partir de x.
- Há um **FIO DE ENTRADA** para cada bit do certificado y.
- O **FIO DE SAÍDA** vale 1 se algum campo C foi mudado para 1.

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



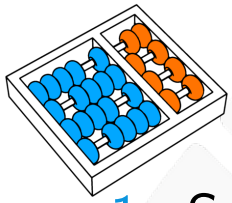
Continuação da demonstração

1. Suponha que $x \in Q$:



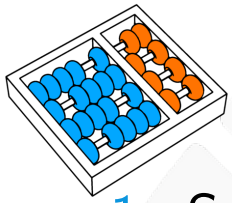
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.



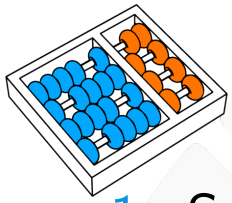
Continuação da demonstração

1. Suponha que $x \in Q$:
   - Então há certificado y tal que $A(x, y) = 1$.
   - Forneça y como entrada para o circuito $F(x)$.



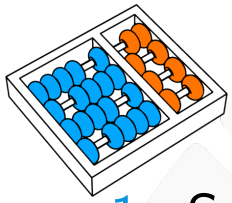
Continuação da demonstração

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.



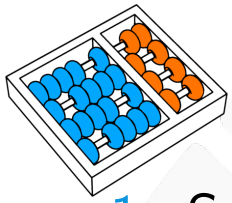
Continuação da demonstração

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.



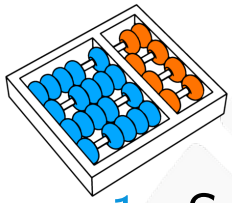
Continuação da demonstração

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:



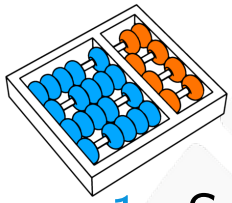
Continuação da demonstração

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.



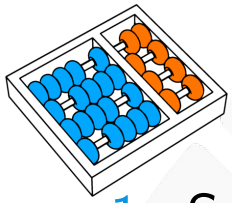
Continuação da demonstração

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.



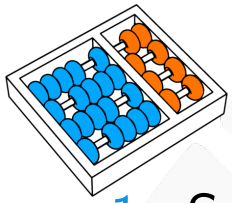
Continuação da demonstração

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 $A(x, y) = 1$.



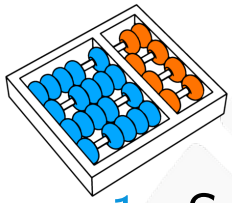
Continuação da demonstração

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 $A(x, y) = 1$.



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, $x \in Q$.



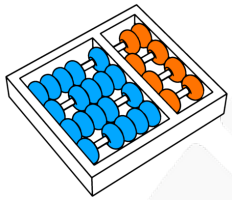
Continuação da demonstração

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

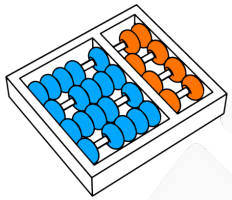
C-SAT é NP*-completo.*

Demonstrando NP-completude



Que outros problemas são NP-difíceis?

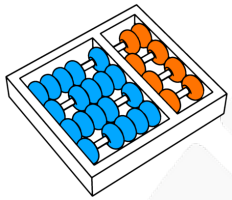
Já sabemos que C-SAT é NP-completo.



Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

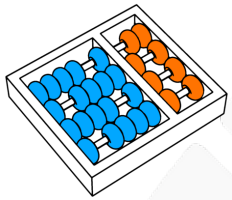
- Como descobrir se outro problema Q é NP-difícil?



Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

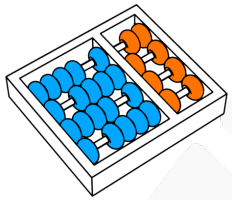
- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:



Que outros problemas são NP-difíceis?

Já sabemos que C-SAT é NP-completo.

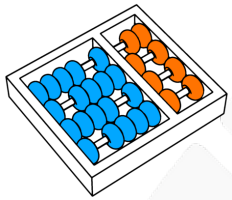
- Como descobrir se outro problema Q é NP-difícil?
- Temos duas possibilidades:
  1. Mostrar que $L \preccurlyeq_p Q$ para **TODO** problema $L \in \text{NP}$.



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 $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para **ALGUM** problema $L \in \text{NP-difícil}$.

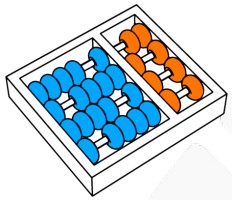


Que outros problemas são NP-difíceis?

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 $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para **ALGUM** problema $L \in \text{NP-difícil}$.

Normalmente usamos a segunda opção:



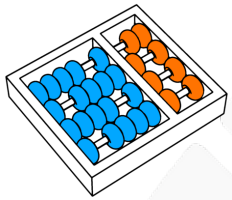
Que outros problemas são NP-difíceis?

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 $L \in \text{NP}$.
  2. Mostrar que $L \preccurlyeq_p Q$ para **ALGUM** problema $L \in \text{NP-difícil}$.

Normalmente usamos a segunda opção:

- Basta reduzir um problema NP-difícil para o problema Q.



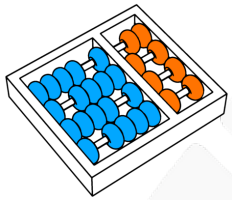
Que outros problemas são NP-difíceis?

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.



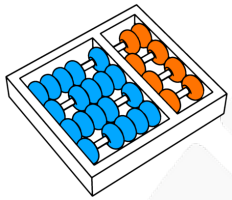
Que outros problemas são NP-difíceis?

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:

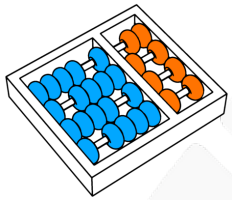
- Basta reduzir um problema NP-difícil para o problema Q.
- Ou seja, mostramos que Q é **TÃO DIFÍCIL** quanto um NP-difícil.
- Esse problema NP-difícil pode ser C-SAT, por exemplo.



NP-completude via redução

Teorema

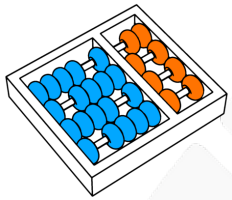
Considere uma linguagem Q e seja $L \in$ NP-difícil.



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.*

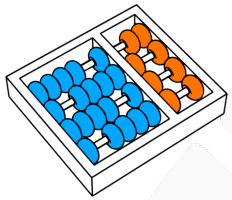


NP-completude via redução

Teorema

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

Demonstração:



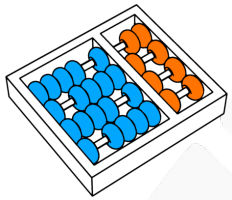
NP-completude via redução

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 $L' \preccurlyeq_p L$.



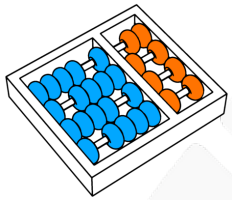
NP-completude via redução

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 $L' \preccurlyeq_p Q$.



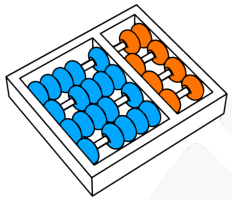
NP-completude via redução

Teorema

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

Demonstração:

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



NP-completude via redução

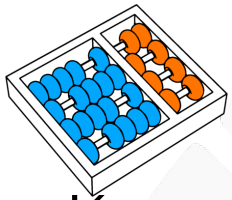
Teorema

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

Demonstração:

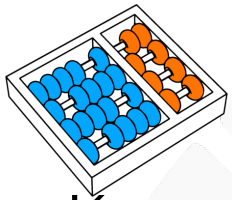
- Como L é NP-difícil, para todo $L' \in$ NP temos $L' \preccurlyeq_p L$.
- Assim, $L' \preccurlyeq_p L$ e $L \preccurlyeq_p Q$, o que implica $L' \preccurlyeq_p Q$.
- Portanto Q é NP-difícil.

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



Literatura

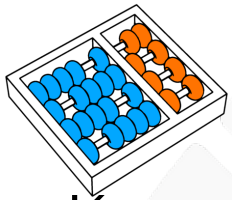
Karp mostrou em 1972 que 21 problemas são NP-completos:



Literatura

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

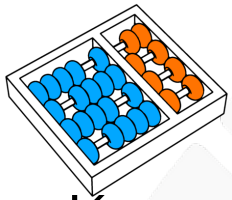
- Listou várias reduções de problemas NP-completos.



Literatura

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

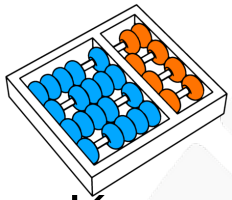
- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.



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!

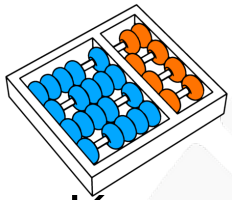


Literatura

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:



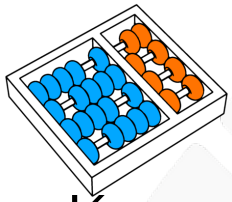
Literatura

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.



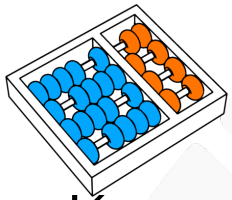
Literatura

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.



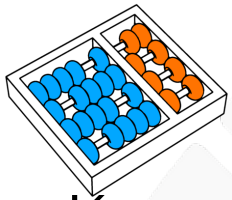
Literatura

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.



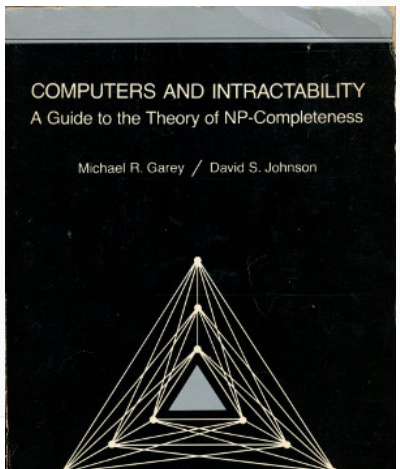
Literatura

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

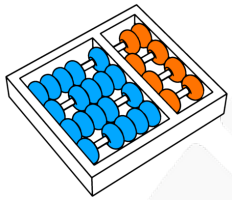
- Listou várias reduções de problemas NP-completos.
- As reduções induzem uma árvore com raiz em SAT.
- veja a Wikipédia!

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.

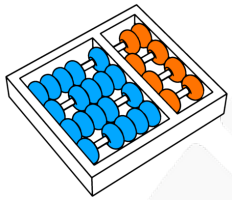


Reduções



Satisfatibilidade

Queremos provar que o seguinte problema é NP-completo:

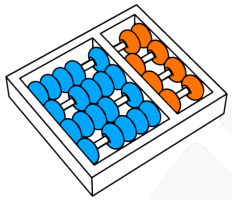


Satisfatibilidade

Queremos provar que o seguinte problema é NP-completo:

Problema (Satisfatibilidade)

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



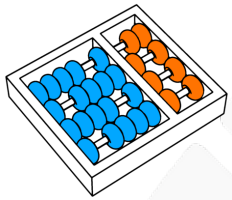
Satisfatibilidade

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}:



Satisfatibilidade

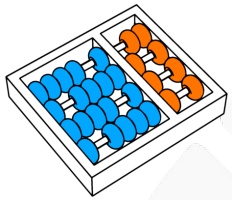
Queremos provar que o seguinte problema é NP-completo:

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 $\wedge$).



Satisfatibilidade

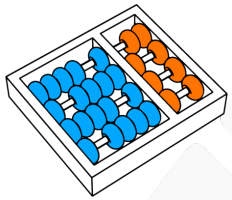
Queremos provar que o seguinte problema é NP-completo:

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 $\vee$).



Satisfatibilidade

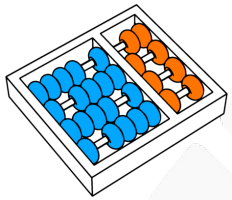
Queremos provar que o seguinte problema é NP-completo:

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.



Satisfatibilidade

Queremos provar que o seguinte problema é NP-completo:

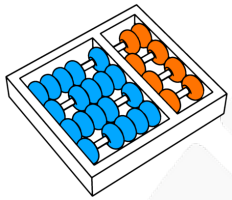
Problema (Satisfatibilidade)

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

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

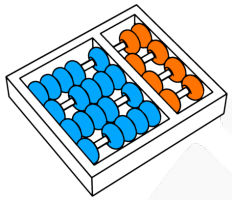
- A fórmula é composta por conjunção de cláusulas (cláusulas relacionadas pelo operador $\wedge$).
- Cada cláusula é composta por disjunção de literais (literais relacionados pelo operador $\vee$).
- Cada literal é uma variável ou a negação de uma variável.

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



Demonstração

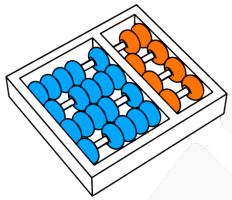
Estratégia a seguir:



Demonstração

Estratégia a seguir:

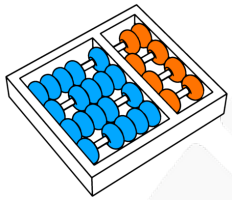
1. Provar que $\text{SAT}_{FNC} \in \text{NP}$.



Demonstração

Estratégia a seguir:

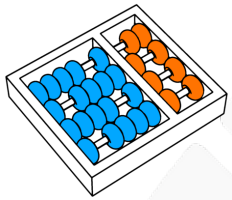
1. Provar que $\text{SAT}_{FNC} \in \text{NP}$.
2. Encontrar um problema NP-completo Π.



Demonstração

Estratégia a seguir:

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

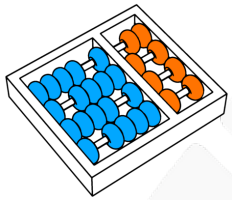


Demonstração

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.



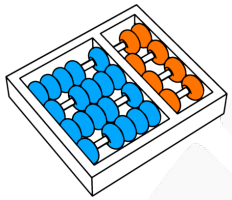
Demonstração

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:



Demonstração

Estratégia a seguir:

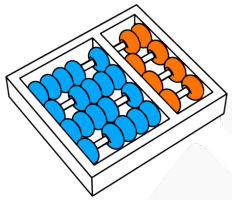
1. Provar que $\text{SAT}_{FNC} \in \text{NP}$.
2. Encontrar um problema NP-completo Π.
3. Provar que $\Pi \preccurlyeq_p \text{SAT}_{FNC}$.

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))

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



Demonstração

Estratégia a seguir:

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

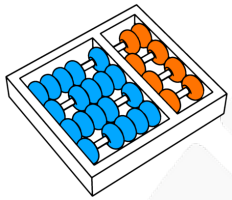
SAT_{FNC} é um caso particular de SAT, que está em NP, portanto SAT_{FNC} também está.

Sabemos que o seguinte problema é NP-completo:

Problema (Satisfatibilidade de circuito (C-SAT))

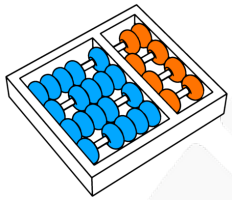
$$\textit{C-SAT} = \{\langle c \rangle : c \textit{ é um circuito lógico satisfazível}\}$$

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



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

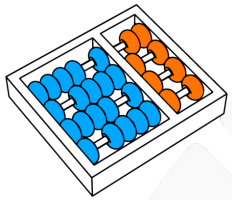
Dado um circuito lógico C, definimos a seguinte fórmula booleana em forma normal conjuntiva $F(C)$:



C-SAT $\preceq_p$ 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 $\mathbf{x_i}$.

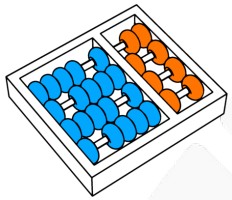


C-SAT $\preccurlyeq_p$ SAT$_{FNC}$

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 $\mathbf{x_i}$.

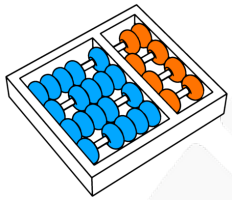
2. Por cada porta lógica de C, definimos cláusulas que garantem **equivalência** lógica com a relação entre as entradas da porta e suas saídas.



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

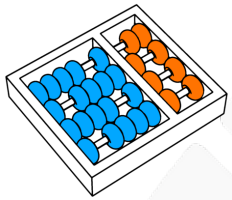
Dado um circuito lógico C, definimos a seguinte fórmula booleana em forma normal conjuntiva $F(C)$:

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.



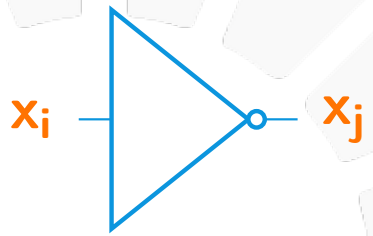
C-SAT $\preceq_p$ SAT$_{FNC}$

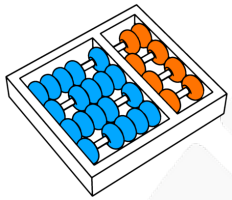
Por cada porta **NOT** de C definimos duas cláusulas em $F(C)$:



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

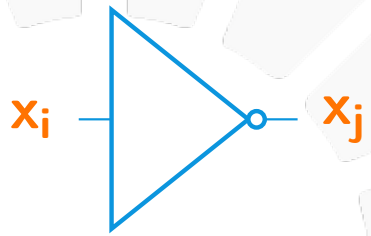
Por cada porta **NOT** de C definimos duas cláusulas em $F(C)$:



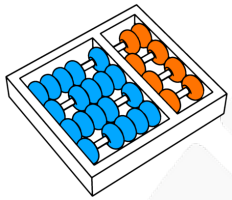


C-SAT $\prec_p$ SAT$_{FNC}$

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

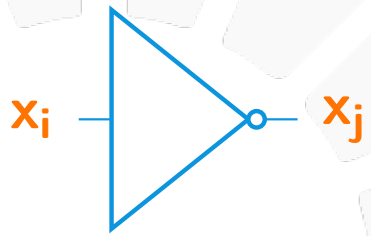


$$\neg x_i \Leftrightarrow x_j$$

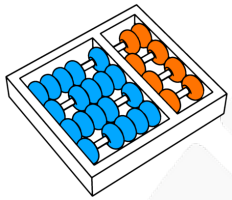


C-SAT $\preceq_p$ SAT$_{FNC}$

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

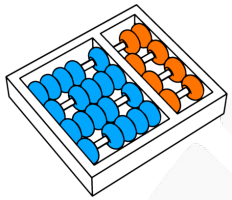


$$\neg x_i \Leftrightarrow x_j$$
$$\equiv \quad (\neg x_i \vee \neg x_j) \wedge (x_i \vee x_j)$$



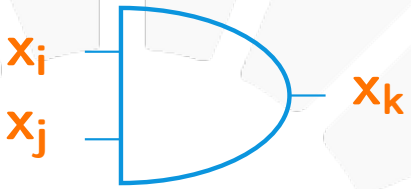
C-SAT $\prec_p$ SAT$_{FNC}$

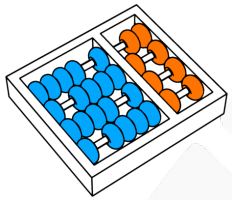
Por cada porta **AND** de C definimos três cláusulas em $F(C)$:



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

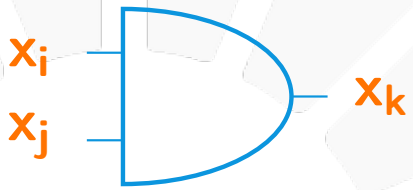
Por cada porta **AND** de C definimos três cláusulas em $F(C)$:



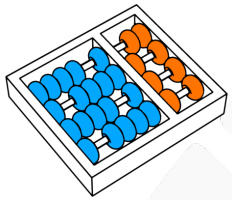


C-SAT $\preceq_p$ SAT$_{FNC}$

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

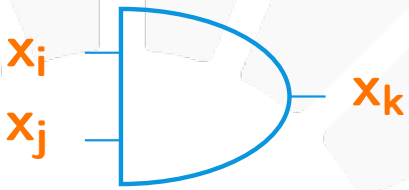


$$x_i \wedge x_j \Leftrightarrow x_k$$

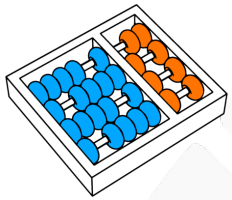


C-SAT $\prec_p$ SAT$_{FNC}$

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

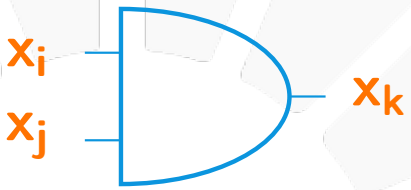


$$x_i \wedge x_j \Leftrightarrow x_k$$
$$\equiv \quad (\neg x_i \vee \neg x_j \vee x_k) \wedge (x_i \vee \neg x_k) \wedge (x_j \vee \neg x_k)$$



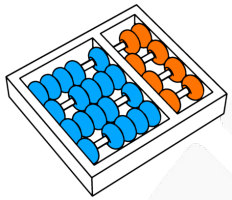
C-SAT $\preceq_p$ SAT$_{FNC}$

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



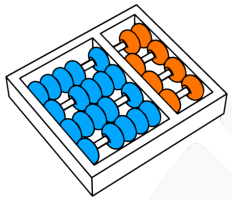
$$x_i \wedge x_j \Leftrightarrow x_k$$
$$\equiv \quad (\neg x_i \vee \neg x_j \vee x_k) \wedge (x_i \vee \neg x_k) \wedge (x_j \vee \neg x_k)$$

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



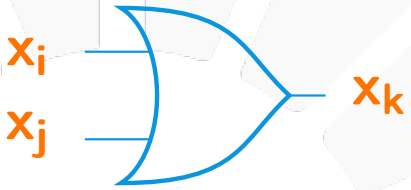
C-SAT $\prec_p$ SAT$_{FNC}$

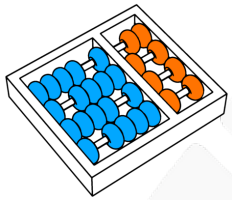
Por cada porta **OR** de C definimos três cláusulas em $F(C)$:



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

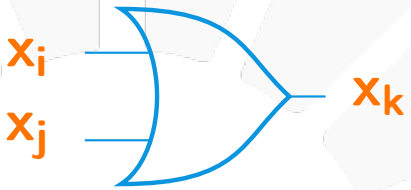
Por cada porta **OR** de C definimos três cláusulas em $F(C)$:



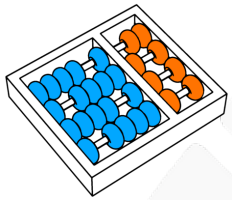


C-SAT $\preceq_p$ SAT$_{FNC}$

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

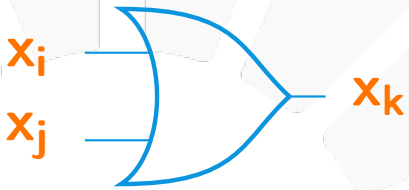


$$x_i \vee x_j \Leftrightarrow x_k$$



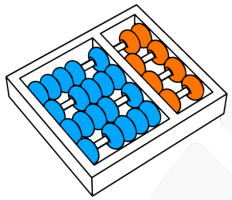
C-SAT $\preceq_p$ SAT$_{FNC}$

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



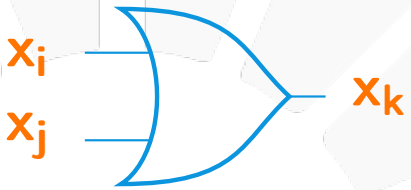
$$x_i \vee x_j \Leftrightarrow x_k$$

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



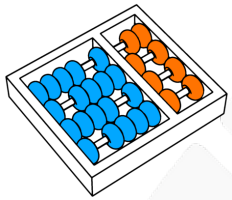
C-SAT $\preceq_p$ SAT$_{FNC}$

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.



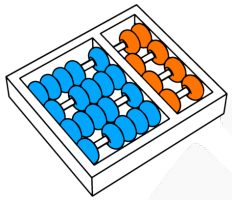
C-SAT $\preccurlyeq_p$ SAT$_{FNC}$

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.

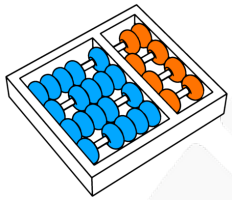
O número de variáveis em $F(C)$ é igual ao número de fios em C e o número de cláusulas não é maior que o número de fios multiplicado pelo número de portas em C. Portanto, $F(C)$ pode ser construída em tempo polinomial a partir de C.



Teoremas

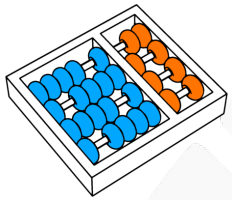
Teorema

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.

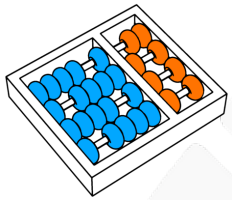


3-SAT

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.*



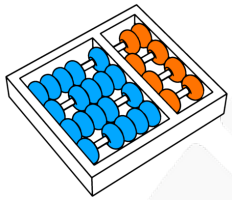
3-SAT

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.



3-SAT

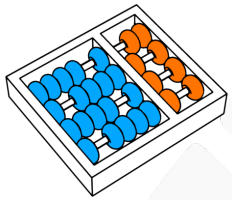
Outro caso especial de SAT é quando a fórmula está em forma normal conjuntiva e cada cláusula tem exatamente três literais.

Teorema

3-SAT é NP*-completo.*

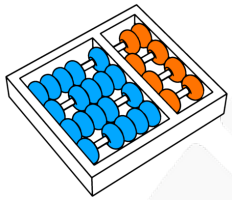
Por ser um caso particular de SAT, sabemos que 3-SAT está em NP.

Basta provar que é NP-difícil.



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

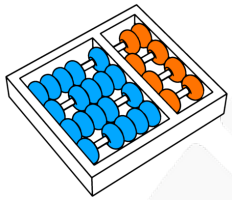
Considere uma fórmula ψ em forma normal conjuntiva, construímos a fórmula $F(\psi)$ em forma normal conjuntiva com exatamente três literais por cláusula, tal que $\psi \in \text{SAT}_{FNC}$ se e somente se $F(\psi) \in$ 3-SAT.



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

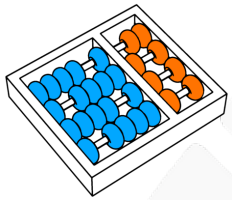
Considere uma fórmula ψ em forma normal conjuntiva, construímos a fórmula $F(\psi)$ em forma normal conjuntiva com exatamente três literais por cláusula, tal que $\psi \in \text{SAT}_{FNC}$ se e somente se $F(\psi) \in$ 3-SAT.

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

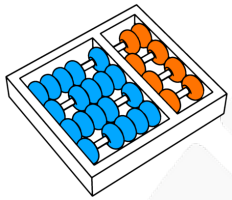
Cada cláusula c ψ com somente um literal ($c = \ell$), é substituída por quatro novas cláusulas que adicionam duas novas variáveis ($x_{c,1}$ e $x_{c,2}$):



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

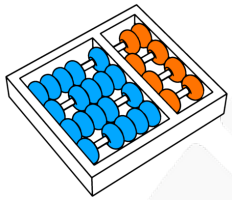
Cada cláusula c ψ com somente um literal ($c = \ell$), é substituída por quatro novas cláusulas que adicionam duas novas variáveis ($x_{c,1}$ e $x_{c,2}$):

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



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

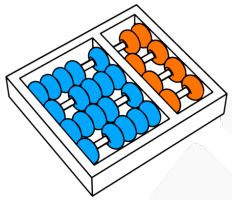
Cada cláusula c de ψ com somente dois literais ($c = \ell_1 \vee \ell_2$), é substituída por dois novas cláusulas que adicionam uma nova variável (x_c):



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

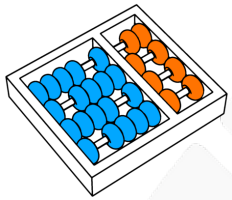
Cada cláusula c de ψ com somente dois literais ($c = \ell_1 \vee \ell_2$), é substituída por dois novas cláusulas que adicionam uma nova variável (x_c):

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



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

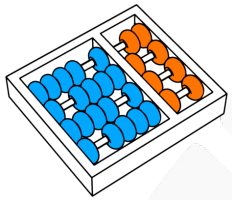
Cada cláusula c de ψ com $k > 3$ literais
$(c = \ell_1 \vee \ell_2 \vee \ell_3 \vee \ldots \vee \ell_k)$ é substituída por $k - 2$ novas cláusulas que adicionam $k - 3$ novas variáveis $(x_{c,1}, x_{c,2}, \ldots, x_{c,k-3})$:



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

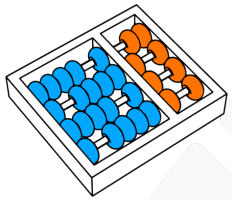
Cada cláusula c de ψ com $k > 3$ literais
$(c = \ell_1 \vee \ell_2 \vee \ell_3 \vee \ldots \vee \ell_k)$ é substituída por $k - 2$ novas cláusulas que adicionam $k - 3$ novas variáveis $(x_{c,1}, x_{c,2}, \ldots, x_{c,k-3})$:

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



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

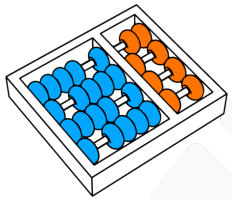
Dada uma fórmula booleana ψ em forma normal conjuntiva, a fórmula booleana $F(\psi)$ em forma normal conjuntiva com exatamente três literais por cláusula garante que $\psi \in \text{SAT}_{FNC}$ se e somente se $F(\psi) \in$ 3-SAT.



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

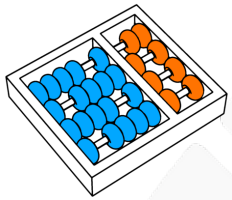
Dada uma fórmula booleana ψ em forma normal conjuntiva, a fórmula booleana $F(\psi)$ em forma normal conjuntiva com exatamente três literais por cláusula garante que $\psi \in \text{SAT}_{FNC}$ se e somente se $F(\psi) \in$ 3-SAT.

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.



Programação linear inteira

Considere uma versão de decisão para programação linear binária:

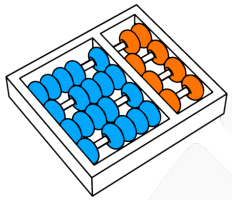


Programação linear inteira

Considere uma versão de decisão para programação linear binária:

Problema (Programação linear inteira (PLI))

$$PLI = \begin{array}{l} \{\langle c, A, b, k\rangle : c \in \mathbb{Z}^n, A \in \mathbb{Z}^{n\times m}, b \in \mathbb{Z}^m, k \in \mathbb{Z}, \\ \textit{existe } x \in \{0,1\}^n \textit{ tal que } Ax \leq b \textit{ e } c^t x \geq k\} \end{array}$$



Programação linear inteira

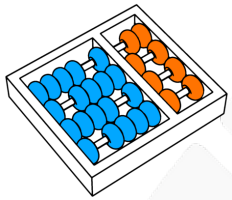
Considere uma versão de decisão para programação linear binária:

Problema (Programação linear inteira (PLI))

$$PLI = \begin{array}{l}\{\langle c, A, b, k\rangle : c \in \mathbb{Z}^n, A \in \mathbb{Z}^{n\times m}, b \in \mathbb{Z}^m, k \in \mathbb{Z}, \\ \textit{existe } x \in \{0,1\}^n \textit{ tal que } Ax \leq b \textit{ e } c^t x \geq k\}\end{array}$$

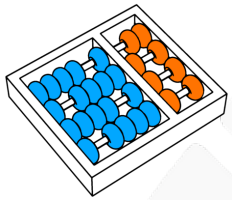
Teorema

PLI é NP*-completo.*



3-SAT $\prec_p$ PLI

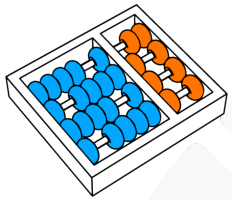
Dada ψ uma fórmula booleana em forma normal conjuntiva com três literais por cláusula, definimos o seguinte problema de programação linear inteira:



3-SAT $\prec_p$ PLI

Dada ψ uma fórmula booleana em forma normal conjuntiva com três literais por cláusula, definimos o seguinte problema de programação linear inteira:

Variáveis:

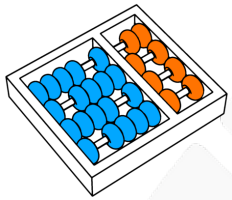


3-SAT $\prec_p$ PLI

Dada ψ uma fórmula booleana em forma normal conjuntiva com três literais por cláusula, definimos o seguinte problema de programação linear inteira:

Variáveis:

- Por cada variável v de ψ definimos a variável $x_v \in \{0, 1\}$.

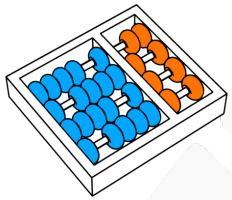


3-SAT $\preceq_p$ PLI

Dada ψ uma fórmula booleana em forma normal conjuntiva com três literais por cláusula, definimos o seguinte problema de programação linear inteira:

Variáveis:

- Por cada variável v de ψ definimos a variável $x_v \in \{0, 1\}$.
- Por cada cláusula c de ψ definimos a variável $y_c \in \{0, 1\}$.



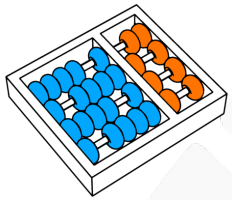
3-SAT $\preceq_p$ PLI

Dada ψ uma fórmula booleana em forma normal conjuntiva com três literais por cláusula, definimos o seguinte problema de programação linear inteira:

Variáveis:

- Por cada variável v de ψ definimos a variável $x_v \in \{0, 1\}$.
- Por cada cláusula c de ψ definimos a variável $y_c \in \{0, 1\}$.

Função objetivo:



3-SAT $\preceq_p$ PLI

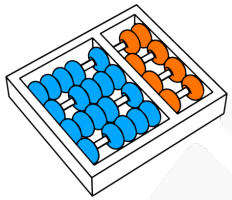
Dada ψ uma fórmula booleana em forma normal conjuntiva com três literais por cláusula, definimos o seguinte problema de programação linear inteira:

Variáveis:

- Por cada variável v de ψ definimos a variável $x_v \in \{0, 1\}$.
- Por cada cláusula c de ψ definimos a variável $y_c \in \{0, 1\}$.

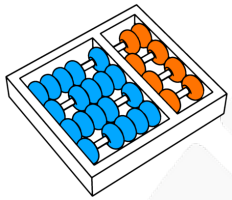
Função objetivo:

- max $\sum_{j=0}^{m} z_j$.



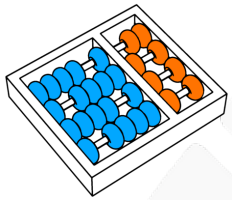
3-SAT $\preceq_p$ PLI. Restrições

- Por cada cláusula $c = (u \vee v \vee w)$, definimos a restrição: $y_c - x_u - x_v - x_w \leq 0$.



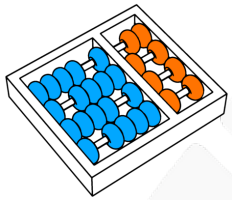
3-SAT $\prec_p$ PLI. Restrições

- Por cada cláusula $c = (u \vee v \vee w)$, definimos a restrição: $y_c - x_u - x_v - x_w \leq 0$.
- Por cada cláusula $c = (u \vee v \vee \neg w)$, definimos a restrição: $y_c - x_u - x_v + x_w \leq 1$.



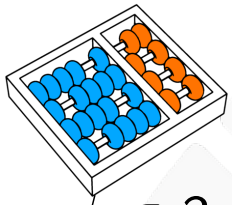
3-SAT $\preceq_p$ PLI. Restrições

- Por cada cláusula $c = (u \vee v \vee w)$, definimos a restrição: $y_c - x_u - x_v - x_w \leq 0$.
- Por cada cláusula $c = (u \vee v \vee \neg w)$, definimos a restrição: $y_c - x_u - x_v + x_w \leq 1$.
- Por cada cláusula $c = (u \vee \neg v \vee \neg w)$, definimos a restrição: $y_c - x_u + x_v + x_w \leq 2$.



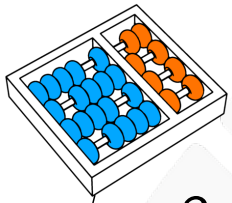
3-SAT $\prec_p$ PLI. Restrições

- Por cada cláusula $c = (u \vee v \vee w)$, definimos a restrição: $y_c - x_u - x_v - x_w \leq 0$.
- Por cada cláusula $c = (u \vee v \vee \neg w)$, definimos a restrição: $y_c - x_u - x_v + x_w \leq 1$.
- Por cada cláusula $c = (u \vee \neg v \vee \neg w)$, definimos a restrição: $y_c - x_u + x_v + x_w \leq 2$.
- Por cada cláusula $c = (\neg u \vee \neg v \vee \neg w)$, definimos a restrição: $y_c + x_u + x_v + x_w \leq 3$.



3-SAT $\prec_p$ PLI

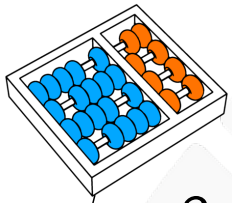
$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.



3-SAT $\preceq_p$ PLI

$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.

Provamos que uma atribuição de valores faz ψ satisfazível se e somente se a mesma atribuição for viável para o programa binário e o valor objetivo for exatamente $k = m$:

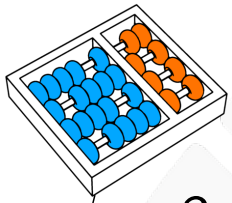


3-SAT $\preceq_p$ PLI

$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.

Provamos que uma atribuição de valores faz ψ satisfazível se e somente se a mesma atribuição for viável para o programa binário e o valor objetivo for exatamente $k = m$:

Para toda variável v de ψ, $x_v = v$.



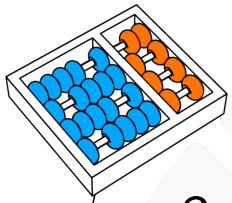
3-SAT $\preceq_p$ PLI

$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.

Provamos que uma atribuição de valores faz ψ satisfazível se e somente se a mesma atribuição for viável para o programa binário e o valor objetivo for exatamente $k = m$:

Para toda variável v de ψ, $x_v = v$.

A cláusula c é verdadeira se e somente se $y_c = 1$:



3-SAT $\preceq_p$ PLI

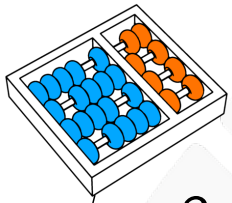
$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.

Provamos que uma atribuição de valores faz ψ satisfazível se e somente se a mesma atribuição for viável para o programa binário e o valor objetivo for exatamente $k = m$:

Para toda variável v de ψ, $x_v = v$.

A cláusula c é verdadeira se e somente se $y_c = 1$:

- $c = (u \vee v \vee w) \Leftrightarrow y_c - x_u - x_v - x_w \leq 0$.



3-SAT $\preceq_p$ PLI

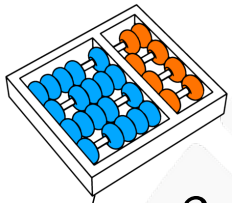
$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.

Provamos que uma atribuição de valores faz ψ satisfazível se e somente se a mesma atribuição for viável para o programa binário e o valor objetivo for exatamente $k = m$:

Para toda variável v de ψ, $x_v = v$.

A cláusula c é verdadeira se e somente se $y_c = 1$:

- $c = (u \vee v \vee w) \Leftrightarrow y_c - x_u - x_v - x_w \leq 0$.
- $c = (u \vee v \vee \neg w) \Leftrightarrow y_c - x_u - x_v + x_w \leq 1$.



3-SAT $\preceq_p$ PLI

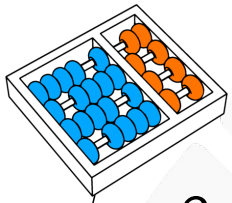
$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.

Provamos que uma atribuição de valores faz ψ satisfazível se e somente se a mesma atribuição for viável para o programa binário e o valor objetivo for exatamente $k = m$:

Para toda variável v de ψ, $x_v = v$.

A cláusula c é verdadeira se e somente se $y_c = 1$:

- $c = (u \vee v \vee w) \Leftrightarrow y_c - x_u - x_v - x_w \leq 0$.
- $c = (u \vee v \vee \neg w) \Leftrightarrow y_c - x_u - x_v + x_w \leq 1$.
- $c = (u \vee \neg v \vee \neg w) \Leftrightarrow y_c - x_u + x_v + x_w \leq 2$.



3-SAT $\preceq_p$ PLI

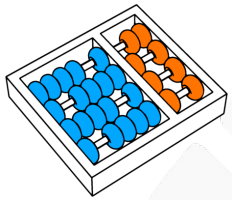
$\psi \in$ 3-SAT se e somente se o seguinte programa linear binário tiver solução com valor objetivo de pelo menos $k = m$.

Provamos que uma atribuição de valores faz ψ satisfazível se e somente se a mesma atribuição for viável para o programa binário e o valor objetivo for exatamente $k = m$:

Para toda variável v de ψ, $x_v = v$.

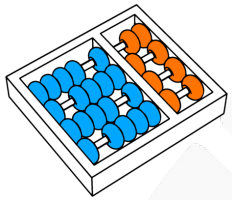
A cláusula c é verdadeira se e somente se $y_c = 1$:

- $c = (u \vee v \vee w) \Leftrightarrow y_c - x_u - x_v - x_w \leq 0$.
- $c = (u \vee v \vee \neg w) \Leftrightarrow y_c - x_u - x_v + x_w \leq 1$.
- $c = (u \vee \neg v \vee \neg w) \Leftrightarrow y_c - x_u + x_v + x_w \leq 2$.
- $c = (\neg u \vee \neg v \vee \neg w) \Leftrightarrow y_c + x_u + x_v + x_w \leq 3$.



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.

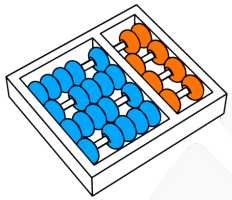


Um problema em grafos

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)

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



Um problema em grafos

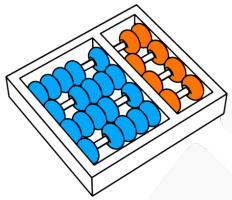
Uma **clique** de um grafo G é um subgrafo completo de G. Ou seja, um subgrafo de G com uma aresta entre cada par de vértices.

Problema (Clique)

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

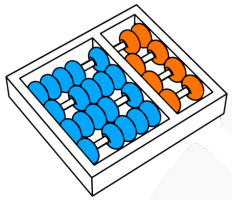
Teorema

CLIQUE é NP*-completo.*



3-SAT $\prec_p$ CLIQUE

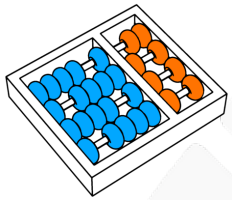
Dada uma fórmula booleana ψ em forma normal conjuntiva com m cláusulas e exatamente três literais por cláusula, construímos uma instância $\langle G, k \rangle = F(\psi)$ da CLIQUE como segue:



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:

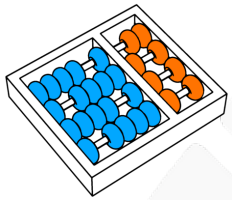
- Por cada cláusula $c = \ell_1 \vee \ell_2 \vee \ell_3$ de ψ, definimos um **cluster** em G que consiste em três vértices (um por cada literal de c): v_{c,ℓ_1}, v_{c,ℓ_2} e v_{c,ℓ_3}.



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:

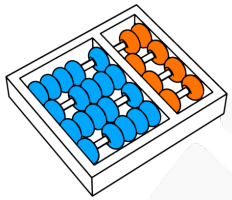
- Por cada cláusula $c = \ell_1 \vee \ell_2 \vee \ell_3$ de ψ, definimos um **cluster** em G que consiste em três vértices (um por cada literal de c): v_{c,ℓ_1}, v_{c,ℓ_2} e v_{c,ℓ_3}.
- Adicionamos uma aresta $(v_{c,\ell}, v_{c',\neg\ell'})$ entre cada par de vértices $v_{c,\ell}$ e $v_{c',\ell'}$ em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



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:

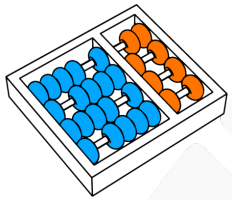
- Por cada cláusula $c = \ell_1 \vee \ell_2 \vee \ell_3$ de ψ, definimos um **cluster** em G que consiste em três vértices (um por cada literal de c): v_{c,ℓ_1}, v_{c,ℓ_2} e v_{c,ℓ_3}.
- Adicionamos uma aresta $(v_{c,\ell}, v_{c',\neg\ell'})$ entre cada par de vértices $v_{c,\ell}$ e $v_{c',\ell'}$ em 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.



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 ($k = m$).

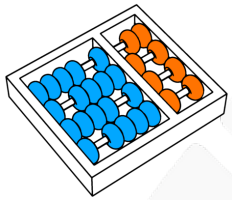


3-SAT $\preceq_p$ CLIQUE

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:

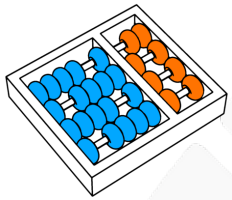
- Por cada cláusula $c = \ell_1 \vee \ell_2 \vee \ell_3$ de ψ, definimos um **cluster** em G que consiste em três vértices (um por cada literal de c): v_{c,ℓ_1}, v_{c,ℓ_2} e v_{c,ℓ_3}.
- Adicionamos uma aresta $(v_{c,\ell}, v_{c',\neg\ell'})$ entre cada par de vértices $v_{c,\ell}$ e $v_{c',\ell'}$ em 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

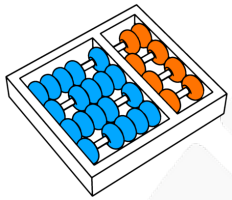
Antes de provar que $\langle \psi \rangle \in$ 3-SAT se e somente se $\langle G, m \rangle \in$ CLIQUE, consideremos as seguintes propriedade:



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:

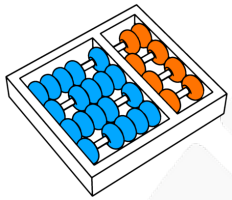
- Se dois vértices $v_{c,\ell}$ e $v_{c',\ell'}$ em G são adjacentes, então os literais associados podem ser simultaneamente verdadeiros $(\ell = \ell' = 1)$.



3-SAT $\preceq_p$ CLIQUE

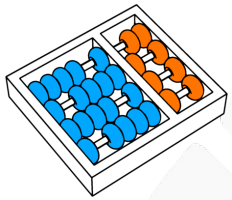
Antes de provar que $\langle \psi \rangle \in$ 3-SAT se e somente se $\langle G, m \rangle \in$ CLIQUE, consideremos as seguintes propriedade:

- 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

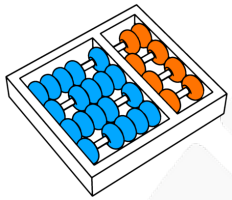
$\langle \psi \rangle \in$ 3-SAT se e somente se $\langle G, m \rangle \in$ CLIQUE:



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.

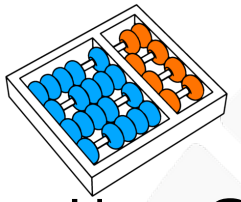


3-SAT $\preceq_p$ CLIQUE

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

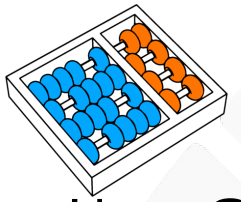
$\Rightarrow$ Se ψ é satisfazível, então denote por α uma atribuição de valores que a satisfaz. Por cada cláusula, selecione um literal que faz essa cláusula verdadeira com a atribuição α. Os m vértices associados aos literais selecionados são adjacentes em G, portanto formam uma clique de m vértices.

$\Leftarrow$ 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

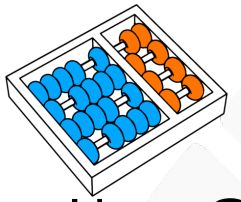
Uma **COBERTURA POR VÉRTICE** (*vertex cover*) de um grafo não direcionado $G = (V, E)$ é um subconjunto de vértices $V' \subseteq V$ tal que qualquer aresta do grafo é incidente a pelo menos um vértice em V'.



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:

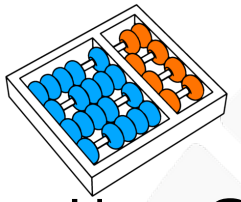


Cobertura por vértices

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.

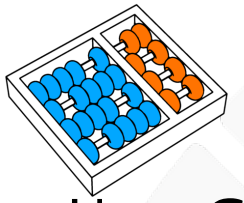


Cobertura por vértices

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.



Cobertura por vértices

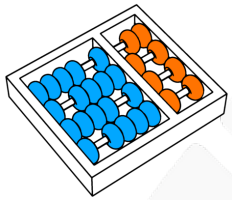
Uma **COBERTURA POR VÉRTICE** (*vertex cover*) de um grafo não direcionado $G = (V, E)$ é um subconjunto de vértices $V' \subseteq V$ tal que qualquer aresta do grafo é incidente a pelo menos um vértice em V'.

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)

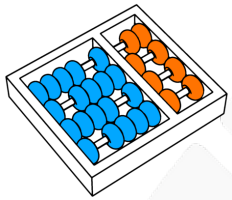
$$\textit{VERTEX-COVER} = \{\langle G, k\rangle : G \textit{ é um grafo que possui uma cobertura por vértices de tamanho } k\}$$



Cobertura por vértices está em NP

Lema

VERTEX-COVER está em NP*.*

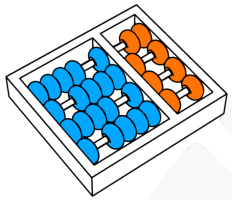


Cobertura por vértices está em NP

Lema

VERTEX-COVER está em NP*.*

Demonstração:



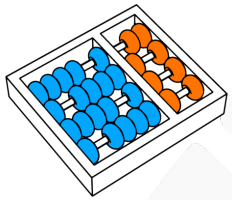
Cobertura por vértices está em NP

Lema

VERTEX-COVER está em NP*.*

Demonstração:

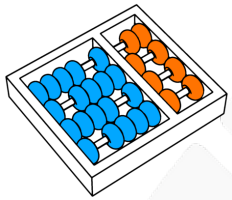
- Exercício.



Cobertura por vértices é NP-difícil

Lema

VERTEX-COVER é NP*-difícil.*

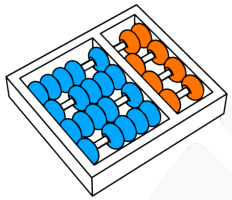


Cobertura por vértices é NP-difícil

Lema

VERTEX-COVER é NP*-difícil.*

- Reduziremos CLIQUE para VERTEX-COVER.

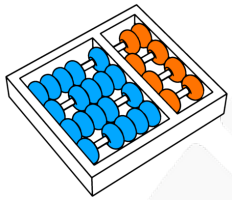


Cobertura por vértices é NP-difícil

Lema

VERTEX-COVER é NP*-difícil.*

- Reduziremos CLIQUE para VERTEX-COVER.
- Dado grafo $G = (V, E)$, criamos o complemento $\overline{G} = (V, \overline{E})$.

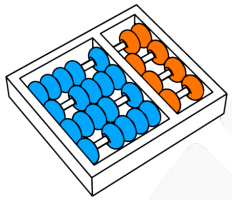


Cobertura por vértices é NP-difícil

Lema

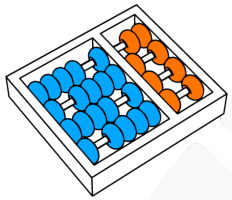
VERTEX-COVER é NP*-difícil.*

- 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 $|V| - k$.



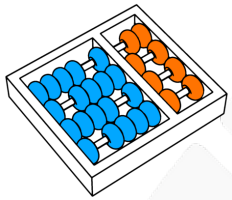
Demonstração

1. Suponha que G tem uma clique C de tamanho k:



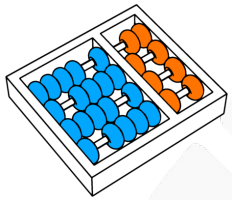
Demonstração

1. Suponha que G tem uma clique C de tamanho k:
   - Defina o conjunto $V' = V \setminus C$.



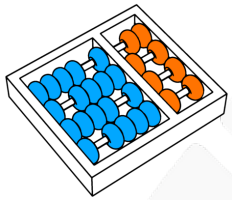
Demonstração

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 $\overline{G}$:



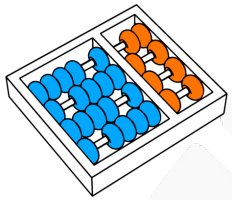
Demonstração

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 $u, v \in C$.



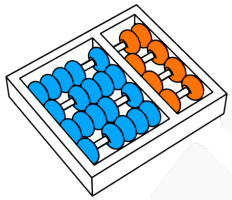
Demonstração

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.



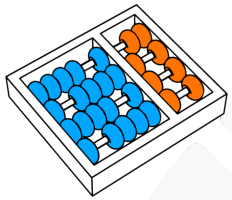
Demonstração

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 $\overline{G}$.



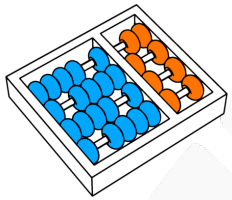
Demonstração

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.



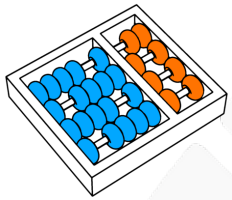
Demonstração

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 $\overline{G}$.



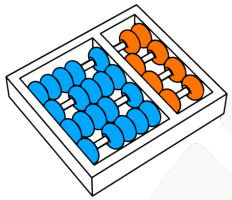
Demonstração

2. Suponha que $\overline{G}$ tem cobertura V' de tamanho $|V| - k$:



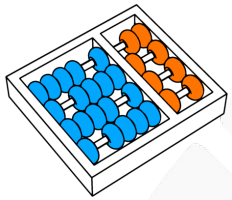
Demonstração

2. Suponha que $\overline{G}$ tem cobertura V' de tamanho $|V| - k$:
   - Defina o conjunto $C = V \setminus V'$.



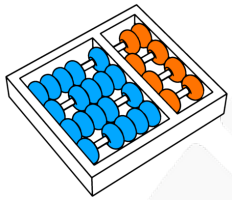
Demonstração

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:



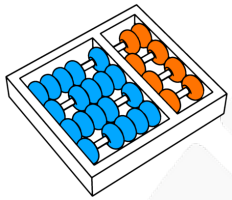
Demonstração

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 $u, v \in C$.



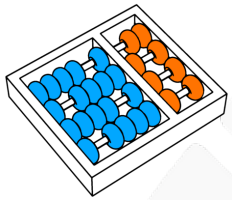
Demonstração

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.



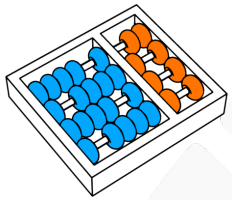
Demonstração

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.



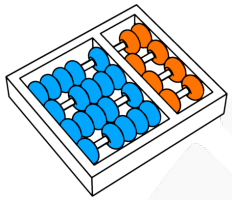
Demonstração

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

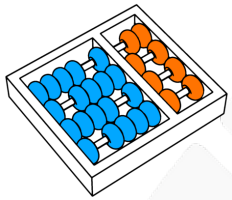
Problema da soma de subconjunto



Soma de subconjunto

Problema da soma de subconjunto

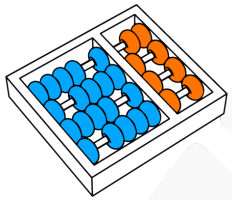
- **Entrada:** um conjunto de naturais S e um natural t



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 $\sum_{s \in S'} s = t$?



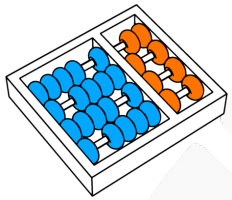
Soma de subconjunto

Problema da soma de subconjunto

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

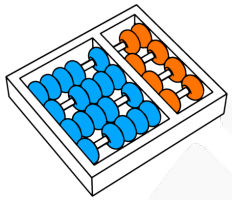
Problema (Soma de subconjunto)

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



Exemplo

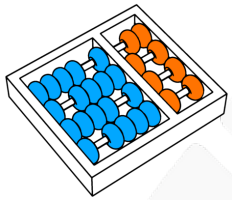
Exemplo de instância



Exemplo

Exemplo de instância

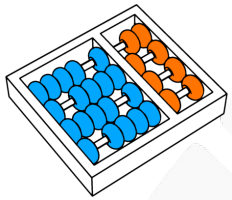
- Sejam $S = \{1, 4, 10, 14, 54, 100, 1004, 1003\}$ e $t = 1027$.



Exemplo

Exemplo de instância

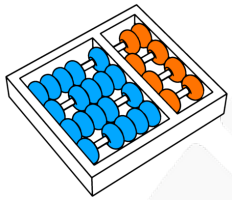
- Sejam $S = \{1, 4, 10, 14, 54, 100, 1004, 1003\}$ e $t = 1027$.
- Nesse caso, $S' = \{10, 14, 1003\}$ é uma solução.



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.

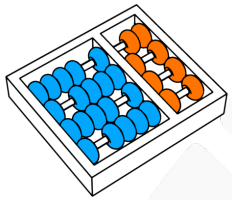


Exemplo

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.

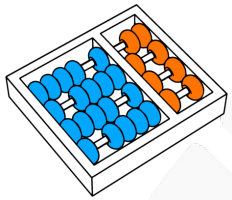
Vamos mostrar que SUBSET-SUM é NP-completo.



Soma de subconjunto está em NP

Lema

SUBSET-SUM está em NP*.*

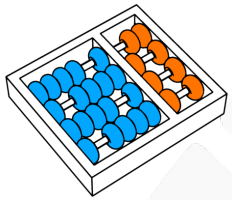


Soma de subconjunto está em NP

Lema

SUBSET-SUM está em NP*.*

Demonstração:



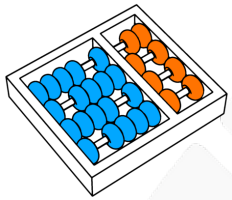
Soma de subconjunto está em NP

Lema

SUBSET-SUM está em NP*.*

Demonstração:

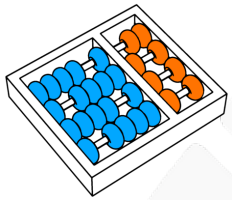
- Exercício.



Soma de subconjunto é NP-difícil

Teorema

SUBSET-SUM é NP*-difícil.*

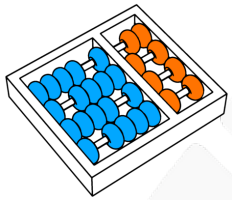


Soma de subconjunto é NP-difícil

Teorema

SUBSET-SUM é NP*-difícil.*

- Reduziremos 3CNF-SAT para SUBSET-SUM.

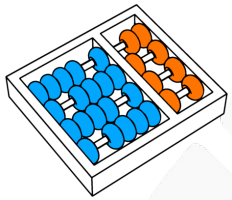


Soma de subconjunto é NP-difícil

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.

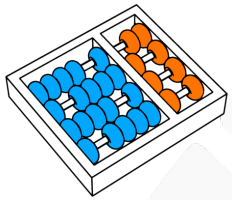


Soma de subconjunto é NP-difícil

Teorema

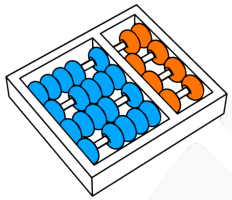
SUBSET-SUM é NP*-difícil.*

- 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

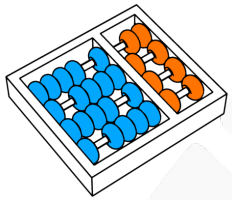
Criamos uma série de números naturais:



Redução

Criamos uma série de números naturais:

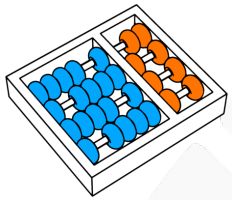
- Cada um tem $n + k$ dígitos **decimais**.



Redução

Criamos uma série de números naturais:

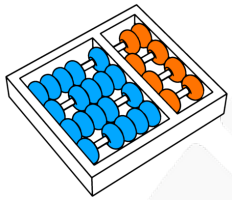
- Cada um tem $n + k$ dígitos **decimais**.
- Cada dígito corresponde a uma variável ou uma cláusula.



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 $x_1, x_2, \ldots, x_n, C_1, \ldots, C_k$.

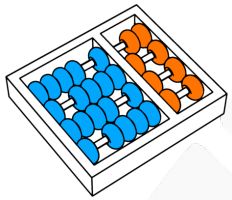


Redução

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:



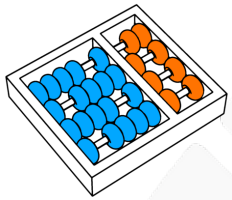
Redução

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.



Redução

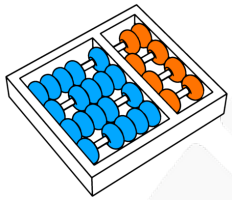
Criamos uma série de números naturais:

- 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:



Redução

Criamos uma série de números naturais:

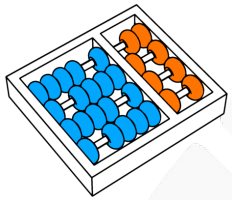
- Cada um tem $n + k$ dígitos **decimais**.
- Cada dígito corresponde a uma variável ou uma cláusula.
- Os dígitos estão em ordem $x_1, x_2, \ldots, x_n, C_1, \ldots, C_k$.

Conjunto de números S:

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

Número alvo t:

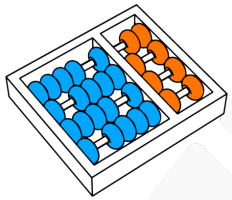
- Criamos **um** número especial.



Exemplo de instância reduzida

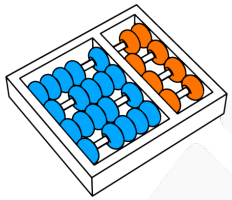
$$f = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

	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

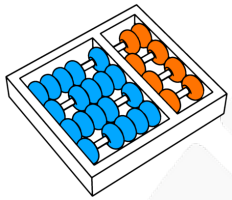
Para uma variável x_i:



Redução

Para uma variável x_i:

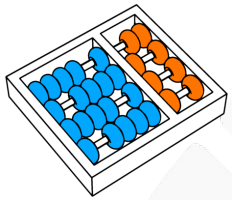
1. Criamos um número v_i com um 1 no dígito x_i e um 1 em cada dígito C_j tal que x_i aparece em C_j.



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.

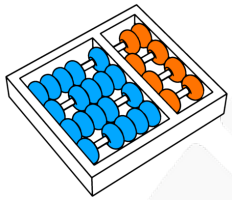


Redução

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:



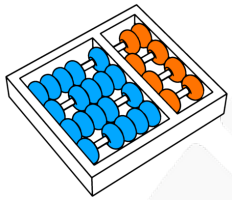
Redução

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.



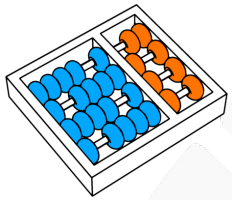
Redução

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.



Redução

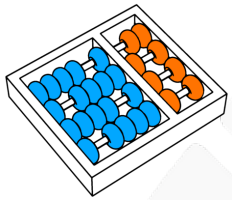
Para uma variável x_i:

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:



Redução

Para uma variável x_i:

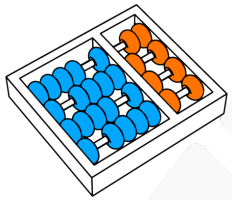
1. Criamos um número v_i com um 1 no dígito x_i e um 1 em cada dígito C_j tal que x_i aparece em C_j.
2. Também um número v_i' com 1 no dígito x_i e um 1 em cada dígito C_i tal que $\neg x_i$ aparece em C_i.

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:

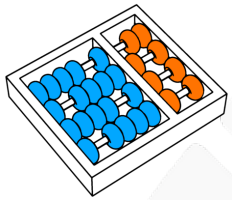
1. Criamos um número com um 1 em cada dígito x_i e um 4 em cada dígito C_i.



Exemplo de instância reduzida

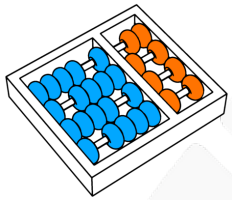
$$f = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$$

	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

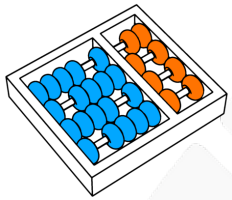
Propriedades da instância reduzida:



Demonstração

Propriedades da instância reduzida:

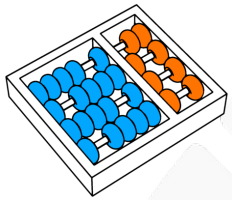
- Todos os números são distintos, pois uma cláusula não tem uma variável e sua negação.



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.

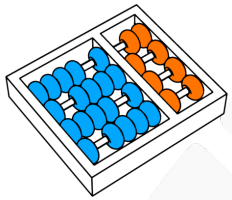


Demonstração

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



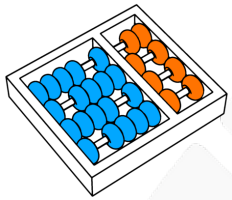
Demonstração

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.



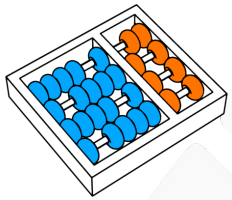
Demonstração

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.

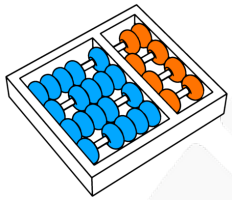
Se somarmos todos os números de S

- 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

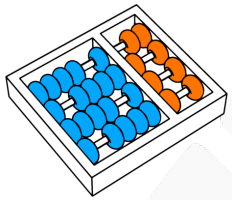
A redução leva tempo polinomial.



Demonstração

A redução leva tempo polinomial.

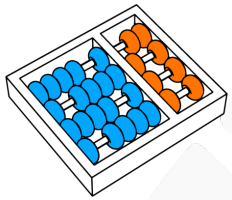
- Além de t, o conjunto S tem $2(n+k)$ números decimais.



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.

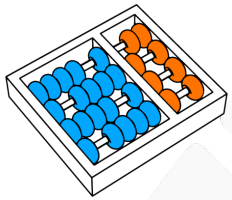


Demonstração

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:



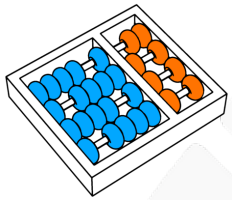
Demonstração

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.

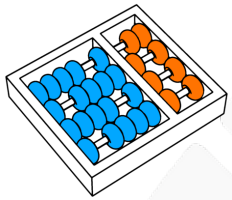
Vamos mostrar a equivalência das instâncias:

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



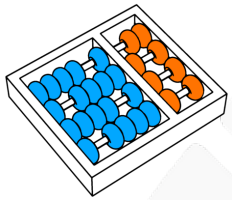
Demonstração

1. Suponha que f é satisfazível:



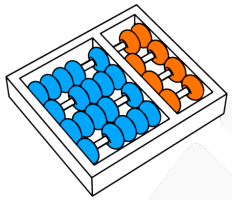
Demonstração

1. Suponha que f é satisfazível:
   - Seja $x_1, \ldots, x_n$ uma atribuição de variáveis que satisfaça f.



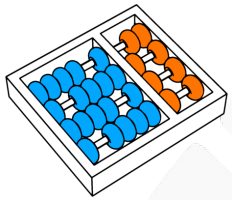
Demonstração

1. Suponha que f é satisfazível:
   - Seja $x_1, \ldots, x_n$ uma atribuição de variáveis que satisfaça f.
   - Vamos criar um subconjunto $S' \subseteq S$.



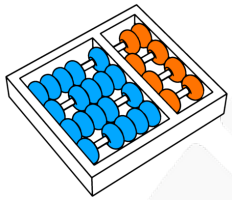
Demonstração

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:



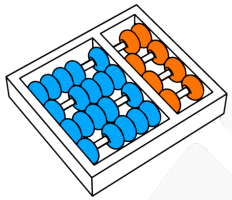
Demonstração

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'.



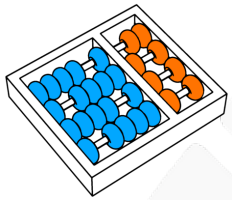
Demonstração

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'.



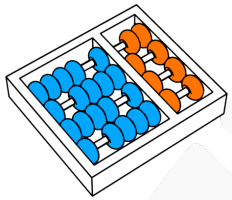
Demonstração

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:



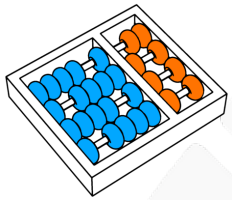
Demonstração

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'.



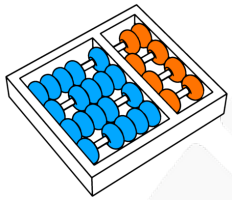
Demonstração

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'.



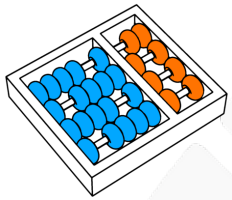
Demonstração

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'.



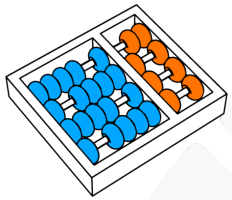
Demonstração

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 $t' = \sum_{s \in S'} s$.



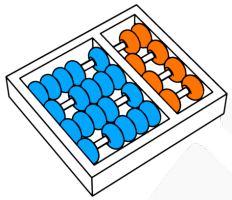
Demonstração

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'.



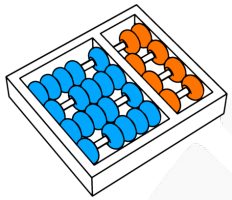
Demonstração

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'.



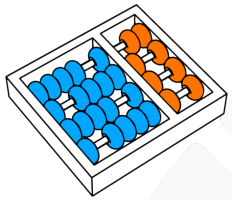
Demonstração

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.



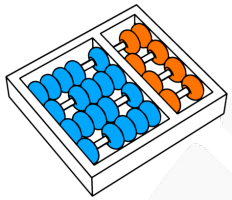
Demonstração

2. Suponha que existe $S' \subseteq S$ tal que $\sum_{s \in S'} s = t$:



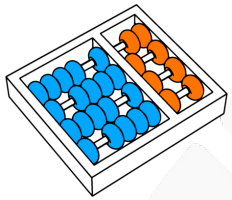
Demonstração

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.



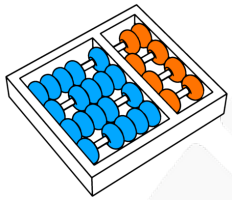
Demonstração

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'.



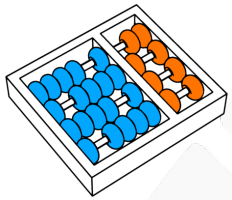
Demonstração

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:



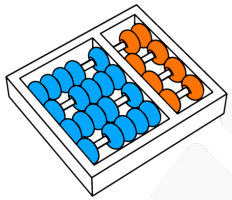
Demonstração

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 $x_i = 1$.



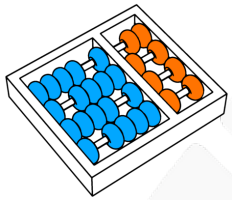
Demonstração

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 $x_i = 0$.



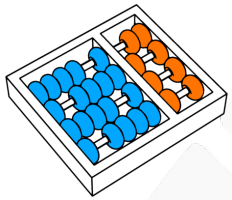
Demonstração

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.



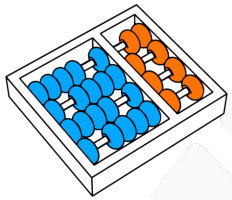
Demonstração

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.



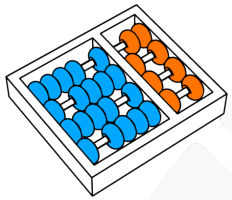
Demonstração

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.



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.



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 e reduções

MC558 - Projeto e Análise de Algoritmos II

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

UNICAMP