

# NP-Completeness

F. K. Miyazawa

Instituto de Computação/Unicamp

2021

# Sumário I

## 1 Visão Geral

- Problemas Computáveis Eficientemente
- Problemas de Decisão
- Problemas Difíceis

## 2 Reduções, Codificação e Linguagens Formais

- Reduções entre Problemas
- Codificação de Problemas
- Linguagens Formais e Classe P

## 3 Certificados, Classes de Complexidade, NP-Completo e Reduções

- Verificação por Certificados em Tempo Polinomial
- Classes NP e co-NP
- NP-Completo e Reduções de Tempo Polinomial

## 4 Reduções de NP-Completo

- CIRCUIT-SAT
- Esquema geral da prova de NP-Completo
- SAT
- 3-CNF-SAT
- CLIQUE
- VERTEX-COVER

## Sumário II

- HAM-CYCLE
- TSP
- SUBSET-SUM
- CIRCUIT-SAT  $\in$  NP-Difícil

### 5 Outros aspectos de complexidade computacional

- Decisão  $\times$  Otimização
- NP-Intermediário
- Complexidade de Espaço e Não-Determinismo
- Problemas Indecidíveis

# Problemas Computáveis Eficientemente

## Projeto e Análise de Algoritmos voltados para computação

- John von Neumann desenvolve o Merge Sort'45
- Análise por Goldstine e von Neumann'48



## Popularizado por Donald Knuth

- The art of Computer Programming'68,69,73,...



# Problemas Computáveis Eficientemente

## Algoritmos rápidos para vários problemas básicos e importantes

- Caminho Mínimo
- Emparelhamento Mínimo
- Programação Linear
- Fluxo Máximo e Corte Mínimo
- Árvore Geradora de Peso Mínimo
- ...

# Problemas Computáveis Eficientemente

- Cobham'64 & Edmonds'65:

Algoritmo *eficiente* = Complexidade de tempo polinomial



Cobham



Edmonds

Algoritmos eficientes foram encontrados para vários problemas

Mas há muitos que – possivelmente – jamais admitirão!

# Problema de Decisão da Satisfatibilidade (D-SAT)

- **Entrada:**

Fórmula lógica  $\phi(x_1, \dots, x_n)$

com variáveis lógicas  $x_1, \dots, x_n$ , operadores lógicos, parênteses sem quantificadores

- **Pergunta:**

Existe atribuição de valores V/F às variáveis  $x_1, \dots, x_n$  que tornam  $\phi$  verdadeira?

**Exemplo:**

$$\phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3)$$

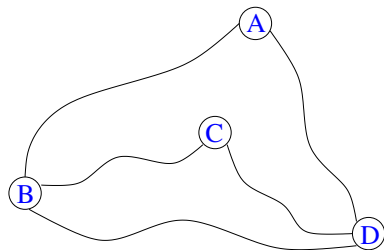
**Resposta:** Sim! Atribuição/Certificado:  $x_1 = \mathbf{V}$ ,  $x_2 = \mathbf{F}$ ,  $x_3 = \mathbf{F}$ .

$$\begin{aligned} \phi(\mathbf{V}, \mathbf{F}, \mathbf{F}) &= (\mathbf{V} \vee \mathbf{F} \vee \mathbf{F}) \wedge (\mathbf{V} \vee \mathbf{V} \vee \mathbf{F}) \wedge (\mathbf{V} \vee \mathbf{V} \vee \mathbf{V}) \\ &= \mathbf{V} \wedge \mathbf{V} \wedge \mathbf{V} \\ &= \mathbf{V} \end{aligned}$$

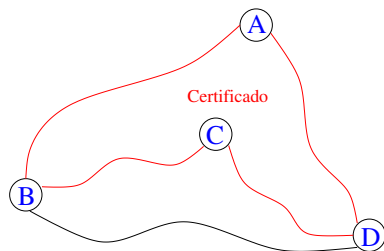
# Problema HAM-CYCLE

- **Entrada:**  
Grafo  $G = (V, E)$
- **Pergunta:**  
Existe ciclo hamiltoniano (passa por cada vértice uma vez) em  $G$ ?

Entrada



Resposta: Sim!





# Problemas Difíceis

Os problemas **D-SAT** e **D-CH**

- Algoritmos (simples):  $O^*(2^n)$  para D-SAT e  $O^*(n!)$  para D-CH
- Possuem um **certificado**/garantia quando a resposta é **Sim**
- **Não são conhecidos algoritmos eficientes para os problemas**
- **Possivelmente jamais terão algoritmos eficientes!**

**Indício:** Há problemas que

- contemplam **toda dificuldade** computacional
- de uma **grande classe**

## Problemas de Decisão e algumas Classes de Complexidade:

Baseada nos modelos básicos de computação, como a Máquina de Turing ou Modelo RAM (Random Access Model).

Segue uma definição informal das classes de problemas P, NP, NP-Completo, NP-Difícil

$X \in P$ :

$X$  pode ser decidido em tempo polinomial.

$X \in NP$ :

$X$  é de decisão e tem certificado curto e verificável em tempo polinomial.

$X \in NP\text{-Completo}$ :

$X \in NP$  e  $\forall Y \in NP$ ,  $Y$  é redutível polinomialmente a  $X$ .

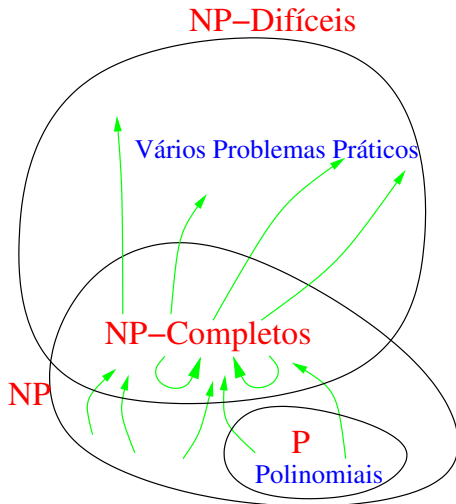
$X \in NP\text{-Difícil}$ :

$\forall Y \in NP$ ,  $Y$  é redutível polinomialmente a  $X$ , onde  $X$  não necessariamente pertence a NP.

## Observações:

- **Todo** problema de NP é redutível polinomialmente a **um** problema NP-completo
- Se existir **um** algoritmo de tempo polinomial para um problema NP-completo então **todos** os problemas de NP se tornam de tempo polinomial
- Se **um** dos problemas de NP só puder ser resolvido em tempo exponencial, então **todos** os problemas NP-completos também só poderão ser resolvidos em tempo exponencial.

## Possível configuração:



# Primeiro Problema NP-Completo: Satisfatibilidade (SAT)

- NP é gigantesca
- Existem problemas NP-completos

Cook'71 & Levin'73: D-SAT é NP-Completo.



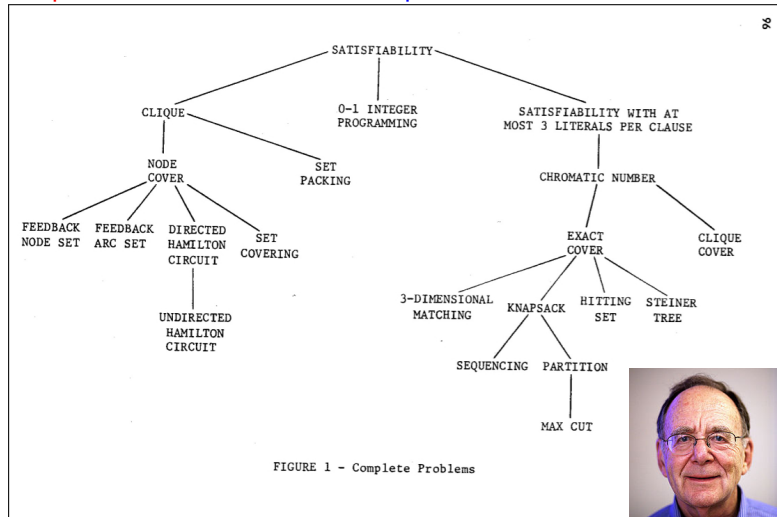
Cook



Levin

# Primeiros Problemas NP-Completo

Karp'72: 21 Problemas NP-Completo.



## 21 Problemas NP-Completos provados por Karp

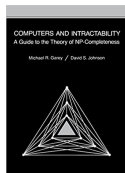
- *Satisfatibilidade* em forma normal conjuntiva
- *Programação Inteira 0-1*
- *Clique*
- *Empacotamento de Conjuntos*
- *Cobertura de Vértices*
- *Cobertura de Conjuntos*
- *Feedback Node Set*
- *Feedback Arc Set*
- *Circuito Hamiltoniano em grafo orientado*
- *Circuito Hamiltoniano em grafo não orientado*
- *3-Satisfatibilidade*
- *Coloração de Grafos*
- *Partição em Cliques*
- *Cobertura Exata*
- *Transversal*
- *Árvore de Steiner*
- *Emparelhamento Tridimensional*
- *Mochila*
- *Sequenciamento de Tarefas*
- *Partição*
- *Corte Máximo*

## O número de problemas NP-completos aumentou rapidamente

- **M.R. Garey & D.S. Johnson'79:**  
Computer and Intractability: A guide to the Theory of NP-Completeness.



“I can't find an efficient algorithm, but neither can all these famous people.”



Garey



Johnson



# Comparando tempos polinomiais e exponenciais

## O quão ruim é ter um algoritmo de tempo exponencial:

- Considere um computador com velocidade de 1 Terahertz (mil vezes mais rápido que um computador de 1 Gigahertz)
- Funções de complexidades de tempo simplificadas

$f(n)$	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
$n$	$2,0 \times 10^{-11}$ seg	$4,0 \times 10^{-11}$ seg	$6,0 \times 10^{-11}$ seg	$8,0 \times 10^{-11}$ seg	$1,0 \times 10^{-10}$ seg
$n^2$	$4,0 \times 10^{-10}$ seg	$1,6 \times 10^{-9}$ seg	$3,6 \times 10^{-9}$ seg	$6,4 \times 10^{-9}$ seg	$1,0 \times 10^{-8}$ seg
$n^3$	$8,0 \times 10^{-9}$ seg	$6,4 \times 10^{-8}$ seg	$2,2 \times 10^{-7}$ seg	$5,1 \times 10^{-7}$ seg	$1,0 \times 10^{-6}$ seg
$n^5$	$2,2 \times 10^{-6}$ seg	$1,0 \times 10^{-4}$ seg	$7,8 \times 10^{-4}$ seg	$3,3 \times 10^{-3}$ seg	$1,0 \times 10^{-2}$ seg
$2^n$	$1,0 \times 10^{-6}$ seg	1,0seg	13,3dias	$1,3 \times 10^5$ séc	$1,4 \times 10^{11}$ séc
$3^n$	$3,4 \times 10^{-3}$ seg	140,7dias	$1,3 \times 10^7$ séc	$1,7 \times 10^{19}$ séc	$5,9 \times 10^{28}$ séc

onde seg=segundos e séc=séculos.

# Comparando tempos polinomiais e exponenciais

E se usarmos computadores muito melhores?

$f(n)$	Computador atual	100×mais rápido	1000×mais rápido
$n$	$N_1$	$100N_1$	$1000N_1$
$n^2$	$N_2$	$10N_2$	$31.6N_2$
$n^3$	$N_3$	$4.64N_3$	$10N_3$
$n^5$	$N_4$	$2.5N_4$	$3.98N_4$
$2^n$	$N_5$	$N_5 + 6.64$	$N_5 + 9.97$
$3^n$	$N_6$	$N_6 + 4.19$	$N_6 + 6.29$

Fixando o tempo de execução

# Problemas NP-difíceis

- Vários problemas práticos são NP-difíceis

## Exemplos:

- ▶ Problema do Caixeiro Viajante
  - ▶ Atribuição de Frequências em Telefonia Celular
  - ▶ Empacotamento de Objetos em Contêineres
  - ▶ Escalonamento de Funcionários em Turnos de Trabalho
  - ▶ Escalonamento de Tarefas em Computadores
  - ▶ Classificação de Objetos
  - ▶ Coloração de Mapas
  - ▶ Projetos de Redes de Computadores
  - ▶ Vários outros...
- 
- $P \neq NP \Rightarrow$  Não existem algoritmos eficientes para problemas NP-difíceis!

## O quão grande é a classe NP-difícil?

Considere o seguinte problema NP-difícil

**Problema MOCHILA:** São dados

- Conjunto de itens  $I = \{1, \dots, n\}$ , cada item  $i$  com tamanho  $t_i > 0$
- Mochila (recipiente) de capacidade  $B > 0$ .

O objetivo é encontrar subconjunto  $S \subseteq I$  tal que

- $\sum_{i \in S} t_i \leq B$
- $\sum_{i \in S} t_i$  é máximo.

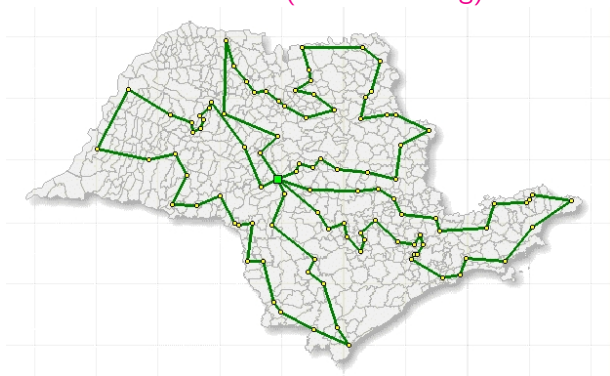
Em problemas práticos:

- É comum termos quantidades limitadas de recursos e o problema da mochila aparecer como subproblema
- Queremos aproveitar o recurso da melhor maneira possível.
- Exemplos: Otimização do número de pessoas, tempo, espaço/ocupação, etc.

Problemas NP-difíceis costumam ser parte de muitos problemas práticos

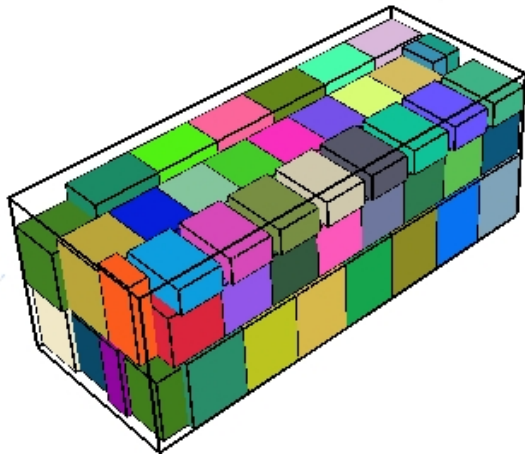
# Dificuldade intrínseca de problemas

Exemplo de problema NP-difícil: Busca por rotas (de custo mínimo) para entrega de produtos em São Paulo (vehicle routing)



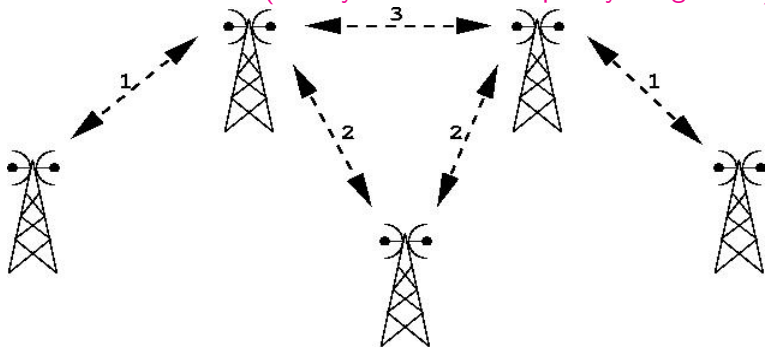
## Dificuldade intrínseca de problemas

Exemplo de problema NP-difícil: calcular o número mínimo de *containers* para transportar um conjunto de caixas com produtos. (bin packing 3D)



# Dificuldade intrínseca de problemas

Exemplo de problema NP-difícil: calcular a localização de antenas para garantir a cobertura de uma certa região e atribuir frequências minimizando interferências (facility location + frequency assignment)



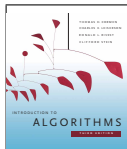
## É importante poder

- **Identificar** se estamos lidando com um problema NP-difícil!
- **Tratar** um problema NP-difícil!
  - ▶ Algoritmos Exatos
  - ▶ Algoritmos de Aproximação
  - ▶ Algoritmos Probabilísticos
  - ▶ Algoritmos Paramétricos
  - ▶ Heurísticas e Metaheurísticas
  - ▶ ...



# Referência para material usados nestes slides

- Livro texto da parte de NP-Completeness [CLRS'09] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms, 3ed. The MIT Press, 2009.



- Consulte outras referências na página do curso, principalmente para fazer exercícios

# Reduções

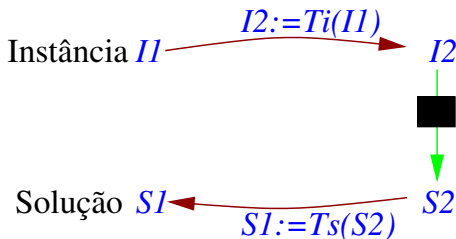
## Reduções de Karp

- Por vezes, conseguimos mapear instâncias de um problema  $P_1$  para instâncias de um problema  $P_2$ ,
- e mapear de volta soluções de  $P_2$  para soluções de  $P_1$ .
- Neste caso, podemos usar  $P_2$  como 'caixa preta' para resolver instâncias de  $P_1$ .
- Este processo é conhecido como redução de  $P_1$  para  $P_2$  e denotaremos por  $P_1 \leq P_2$ .

# Reduções

Reduções de Karp:  $P_1$  é redutível a  $P_2$  ( $P_1 \leq P_2$ ) se

- $\exists Ti$  que transforma instância  $I_1$  de  $P_1$  para instância  $I_2$  de  $P_2$
- $\exists Ts$  que transforma solução  $S_2$  de  $I_2$  para solução  $S_1$  de  $I_1$



Claramente vale transitividade:

Se  $P_1 \leq P_2$  e  $P_2 \leq P_3$  então  $P_1 \leq P_3$

# Reduções

## Exemplo de redução de Karp

### Problema (do Triângulo em um Grafo (D-TG))

*Dado grafo  $G = (V, E)$  não-orientado, decidir se há triângulo (3 vértices com arestas entre si) em  $G$ .*

Algoritmo trivial  $O(n^3)$ : Verificar os  $\binom{n}{3}$  subconjuntos de 3 vértices de  $G$ .

Vamos reduzir D-TG para o problema de Multiplicação de Matrizes e obter algoritmo com complexidade de tempo melhor

### Problema (Multiplicação de Matrizes (MM))

*Dadas matrizes  $A$  e  $B$ , computar a matriz  $C = A \cdot B$ .*

Algoritmos para MM:

- Strassen'69:  $O(n^{2,807})$ .
- Coppersmith-Winograd'90:  $O(n^{2,376})$ .

# Reduções

Seja  $A$  a matriz de adjacência de  $G$ :

$$A[i, j] = \begin{cases} 1 & \text{se } \{i, j\} \in E \\ 0 & \text{caso contrário.} \end{cases}$$

Compute  $A^2$ . Note que

$$A^2[i, j] = \sum_{k=1}^n A[i, k] \cdot A[k, j]$$

I.e.,

$$A^2[i, j] > 0 \iff \text{existe } k \text{ tal que } A[i, k] = 1 \text{ e } A[k, j] = 1.$$

Assim,

$$G \text{ tem triângulo} \iff \text{existem } i, j \text{ tal que } A[i, j] = 1 \text{ e } A^2[i, j] > 0$$

# Reduções

## Proposição

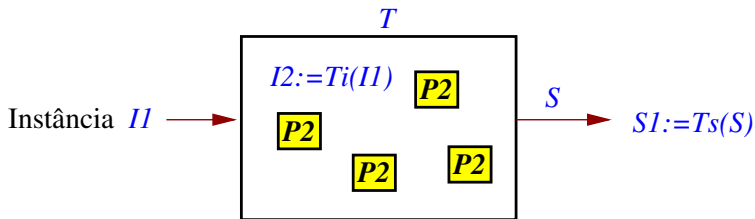
*O Problema D-TG pode ser resolvido em tempo  $O(n^{2,376})$ .*

## Algoritmo

1. Compute matriz de adjacência  $A$ .
2. Compute  $A^2$  (em tempo  $O(n^{2,376})$ ).
3. Para cada par  $1 \leq i, j \leq n$ , com  $i \neq j$ :
4. Se  $A[i, j] = 1$  e  $A^2[i, j] > 0$  devolva 1.
5. Devolva 0.

# Reduções

Reduções de Turing: reduções de  $P_1$  para  $P_2$ , fazendo várias aplicações de  $P_2$ .



Exemplificaremos, mostrando que muitos problemas de busca podem ser resolvidos pelos correspondentes problemas de decisão.

## Redução para Problemas de Decisão

A teoria de NP-Compleitude é desenvolvida sobre problemas de decisão.

- Problemas que possuem resposta sim/não.
- Em geral, problemas práticos não são de decisão.

Muitas vezes, a versão de decisão se refere a existência de uma estrutura (em vez de buscá-la).

### Problema (de Decisão do Ciclo Hamiltoniano)

*Dado grafo  $G$ , decidir se  $G$  possui ciclo que passa por cada vértice exatamente uma vez.*

Na prática, queremos encontrar tal estrutura.

### Problema (do Ciclo Hamiltoniano)

*Dado grafo  $G$ , encontrar ciclo em  $G$  que passa por cada vértice exatamente uma vez, caso exista.*



# Redução para Problemas de Decisão

Apesar da aparente simplicidade, muitos problemas de decisão são tão difíceis quanto as versões de busca.

Vamos reduzir o problema (de busca) do Ciclo Hamiltoniano para sua versão de decisão.

Suponha que temos uma rotina ACH que decide a existência de ciclo hamiltoniano em um grafo:

## Subrotina ACH( $G$ )

ACH( $G$ ) devolve 1, se  $G$  possui ciclo hamiltoniano

ACH( $G$ ) devolve 0, caso contrário.

**Ideia:** Se  $G$  — e possui ciclo hamiltoniano, descarte e

# Redução para Problemas de Decisão

**Algoritmo BuscaCH(G)** # Devolve ciclo hamiltoniano de  $G$ , se existir

1. Se  $ACH(G)=0$  devolva  $\emptyset$
2. Senão
3. Para todo  $e \in E$  faça
4. Se  $ACH(G - e) = 1$  então remova  $e$  de  $G$ .
5. Devolva  $G$

Se um problema  $\Pi_1$  é redutível polinomialmente a um problema  $\Pi_2$  e vice-versa, dizemos que  $\Pi_1$  é **polinomialmente equivalente** a  $\Pi_2$ .

## Teorema

*Os problemas CH e D-CH são polinomialmente equivalentes*

# Problemas de Otimização para Decisão

Problemas de otimização: buscamos por

- estruturas que respeitam certas propriedades e
- possuem o *melhor* valor dentre elas.

Podem ser de minimização ou de maximização:

- Problemas de minimização: buscamos estruturas de valor mínimo.
- Problemas de maximização: buscamos estruturas de valor máximo.

Forma padrão para construir versão de decisão: Inclusão de parâmetro  $K$

- Problemas de minimização: Decidir se há estrutura de valor  $\leq K$
- Problemas de maximização: Decidir se há estrutura de valor  $\geq K$

# Problemas de Otimização para Decisão

## Exemplo

### Problema (do Caminho Mínimo (CM))

*Dado grafo  $G = (V, E)$  e dois vértices  $s, t \in V$ , encontrar um caminho mais curto entre  $s$  e  $t$  em  $G$ .*

### Problema (de Decisão do Caminho Mínimo (D-CM))

*Dado grafo  $G = (V, E)$ , dois vértices  $s, t \in V$  e um inteiro  $K$ , decidir se há caminho de  $s$  a  $t$  em  $G$  de comprimento no máximo  $K$ .*

**Exercício:** Faça uma redução de Turing em tempo polinomial do Problema de Encontrar um Caminho Mais Curto para o Problema de Decisão do Caminho Mínimo.

# Reduções

Redução de  $P_1$  para  $P_2$  é interessante para:

- Resolver  $P_1$  quando já temos a resposta para  $P_2$ .
- Transferir a “dificuldade” de  $P_1$  para  $P_2$ , caso as transformações envolvidas sejam suficientemente rápidas.

# Codificação de Problemas

- **Instâncias** são **codificadas** por strings de bits.
- $|I|$ : Tamanho de  $I$  - número de bits da string que codifica  $I$ .
- **Complexidade** é definida em relação ao tamanho  $|I|$ .

**Exemplo:** Considere **algoritmo**  $A$  e **instância**  $I$  dada por um **inteiro**  $k$ , e  $A$  tem **complexidade**  $O(k)$ .

- Se usarmos a **codificação unária**: o número  $k$  é codificado com  $k$  uns e  $A$  tem complexidade de **tempo linear**
- Se usarmos a **codificação binária**: o número  $k$  é codificado com  $n = \lceil \log_2 k \rceil$  bits e  $A$  tem complexidade de **tempo exponencial** ( $k = O(2^n)$ ).
  
- Na prática não consideramos codificações “caras” como a unária.
- Utilizaremos codificações “baratas” / compactas, como a binária.

# Codificação de Problemas

Codificações com outras bases fixas (como ternária, decimal,...) também funcionam.

## Exemplo:

- Um inteiro  $N$  na base 2 gasta  $n = \lceil \log_2 N \rceil$  bits e na base 3 gasta  $n' = \lceil \log_3 N \rceil$  bits
- Como  $\log_2 N = \frac{\log_3 N}{\log_3 2}$
- Mudança para codificação ternária difere por um fator constante.

Assumiremos base binária.

**Exercício:** Anteriormente, vimos um algoritmo por PD para o Problema da Mochila com dados inteiros e complexidade  $O(nK)$ , onde  $n$  é o número de itens e  $K$  a capacidade da mochila. Comente a complexidade computacional deste algoritmo supondo codificação compacta para representar seus dados.

# Linguagens Formais

Podemos representar os problemas de decisão por linguagens formais.

## Notação:

- Alfabeto  $\Sigma$ : conjunto finito de símbolos
- *Linguagem sobre  $\Sigma$* : Conjunto de strings formadas com símbolos de  $\Sigma$
- $\epsilon$ : String vazia
- $\emptyset$ : Linguagem vazia
- $\Sigma^*$ : Linguagem com todas possíveis strings sobre  $\Sigma$

## Exemplo

Se  $\Sigma = \{0, 1\}$  então  $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$



# Linguagens Formais

- Operações usuais sobre linguagens:  $\in$ ,  $\subset$ ,  $\cup$ ,  $\cap$
- $\bar{L}$ : Complemento de uma linguagem  $L$ , dada por
$$\bar{L} = \Sigma^* - L$$
- $L_1L_2$ : Concatenação de linguagens  $L_1$  e  $L_2$  é dada pela linguagem
$$L_1L_2 = \{x_1x_2 : x_1 \in L_1 \text{ e } x_2 \in L_2\}$$
- $L^i$ : Concatenação da linguagem  $L$ ,  $i$  vezes.
- $L^*$ : Fecho de uma linguagem é dada por
$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup \dots$$

# Linguagens Formais

## Dado problema de decisão $\Pi$

- Assumiremos  $\Sigma = \{0, 1\}$ .
- $\Sigma^*$  corresponde a todas possíveis instâncias de  $\Pi$ .
- $\Pi$  mapeia instâncias de  $\Sigma^*$  para  $\{0, 1\}$  (se resposta *sim* ou *não*).
- Assumiremos que  $\Pi$  é a linguagem:

$$\Pi = \{x \in \Sigma^* : \Pi(x) = 1\}$$

I.e.,  $\Pi$  é a linguagem composta das instâncias cuja solução é *sim*.

# Linguagens Formais

## Problema (PATH)

$\text{PATH} = \{ \langle G, u, v, k \rangle : \begin{array}{l} G = (V, E) \text{ é um grafo} \\ u, v \in V, \\ k \geq 0 \text{ inteiro} \\ \exists \text{ caminho de } u \text{ a } v \text{ em } G \text{ de tamanho } \leq k \end{array} \}$

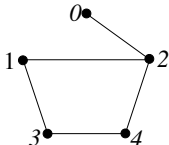
**Notação:**  $\langle l \rangle$  é a string, em bits, que codifica  $l$ .

# Linguagens Formais

Possível codificação para entradas em PATH:

- $T$ : Sequência de 0's terminando em 1 (número de bits dos números)
- Grafo  $G = (V, E)$  codificado como:
  - ▶  $|V|$ : Número de vértices ( $V$  é conjunto de inteiros começando do 0)
  - ▶  $|E|$ : Número de arestas (pares de vértices)
  - ▶  $E$ : Sequência das arestas
- Vértices  $u$  e  $v$  (origem e destino do caminho)
- $k$  (tamanho máximo do caminho)

Exemplo de  $x = \langle G, u, v, k \rangle \in \text{PATH}$ :



$$x = \underbrace{00110110100001000101000101101010000111000000110111}_{\substack{T \\ |V| \quad |E| \quad (0,2) \quad (1,2) \quad (1,3) \quad (2,4) \quad (3,4) \\ G}} \quad \underbrace{0 \quad 3 \quad 3}_{\substack{u \quad v \quad k}}$$

# Linguagens Formais

Dado algoritmo  $A$  e  $x \in \Sigma^*$  escrevemos

- $A(x) = 1$  se  $A$  aceita  $x$
- $A(x) = 0$  se  $A$  rejeita  $x$

A linguagem aceita por  $A$  é

$$L = \{x \in \Sigma^* : A(x) = 1\}$$

Dizemos que  $A$  aceita uma linguagem  $L$  se

- $A(x) = 1$  para todo  $x \in L$ ;
- $A(x) \neq 1$  para todo  $x \notin L$  (o algoritmo pode não parar).

Dizemos que  $A$  decide uma linguagem  $L$  se

- $A(x) = 1$  para todo  $x \in L$
- $A(x) = 0$  para todo  $x \notin L$ .

# Linguagens Formais

*A* aceita linguagem  $L$  em tempo polinomial se

- *A* aceita  $L$
- $A(x)$  executa em tempo polinomial  $\forall x \in L$

*A* decide linguagem  $L$  em tempo polinomial se

- *A* decide  $L$
- $A(x)$  executa em tempo polinomial  $\forall x \in \Sigma^*$

Definição (Classe  $P$ )

$P = \{L \subseteq \Sigma^* : \text{existe um algoritmo } A \text{ que decide } L \text{ em tempo polinomial}\}$

# Linguagens Formais

## Teorema

$$P = \{L \subseteq \Sigma^* : L \text{ é aceita por um algoritmo polinomial}\}$$

*Prova.*

⊆ Seja  $L \in P$ .

Por definição, existe algoritmo  $A$  de tempo polinomial que **decide**  $L$ . Então,  $A$  também **aceita**  $L$ .

⊇ Seja  $L$  uma linguagem **aceita** por um algoritmo  $A$  em tempo polinomial  $p(n)$ .

Construa um algoritmo  $A'$  que **decide**  $L$  em tempo  $O(p(n))$ :  
Seja  $x \in L$  e  $n = |x|$ . Temos que o  $A(x)$  executa em  $p(n)$  passos. O algoritmo  $A'$  emula  $A$  executando os mesmos passos.

Se  $A$  **aceitou**  $x$  dentro do tempo  $p(n)$ ,  $A'$  **aceita**  $x$ .

Caso contrário,  $A'$  **rejeita**  $x$ .

□

# Verificação por Certificados em Tempo Polinomial

**Certificados:** estruturas que *certificam* que uma entrada tem resposta SIM

**Exemplo:** Considere a linguagem (problema de decisão) do Ciclo Hamiltoniano

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

Para todo grafo  $G$  tal que  $\langle G \rangle \in \text{HAM-CYCLE}$

- Existe um ciclo hamiltoniano (certificado)  $S$  em  $G$
- Podemos ter algoritmo que *verifica* que  $S$  é de fato ciclo hamiltoniano
- Esta *verificação* pode ser feita em tempo polinomial

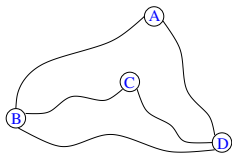


# Verificação por Certificados em Tempo Polinomial

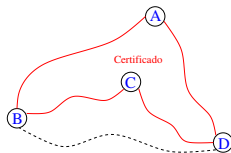
**Exemplo:** Considere

- Grafo  $G = (V, E)$  e  $x = \langle G \rangle$ .
- Permutação de vértices  $S$  que é ciclo hamiltoniano em  $G$  e  $y = \langle S \rangle$ .

Entrada  $x = \langle G \rangle$



Certificado:  $y = \langle A, B, C, D \rangle$



Podemos ter **algoritmo de tempo polinomial  $A$  que verifica** que  $S$  é de fato ciclo hamiltoniano de  $G$ :

- Verifica que  $S$  é permutação de  $V$
- Verifica ter aresta em  $G$  entre cada par de vértices consecutivos de  $S$
- Verifica ter aresta em  $G$  ligando os extremos de  $S$

# Verificação por Certificados em Tempo Polinomial

Algoritmo de verificação  $A$ :

$A$  tem dois argumentos:  $x$  (entrada) e  $y$  (possível certificado).

$A$  verifica a entrada  $x$  se há certificado  $y$  tal que  $A(x, y) = 1$ .

Linguagem verificada por  $A$  é

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

- Se  $x \in L$  então existe certificado  $y$  tal que  $A(x, y) = 1$ .
- Se  $x \notin L$  então para todo  $y$  temos  $A(x, y) \neq 1$ .

Se  $A$  é de tempo polinomial e  $y$  tem tamanho polinomial em relação a  $x$ ,  
 $A$  verifica  $L$  em tempo polinomial

# Verificação por Certificados em Tempo Polinomial

**Exemplo:** Considere a linguagem dos grafos hamiltonianos

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

Considere um algoritmo  $A$  com dois argumentos  $x$  e  $y$ , tal que

- $y$  (certificado) tem tamanho polinomial em relação a  $x$  (entrada)
- $A(x, y)$  devolve 0 se  $x$  não codifica um grafo  $G$
- $A(x, y)$  devolve 0 se  $y$  não codifica um ciclo hamiltoniano de  $G$
- $A(x, y)$  devolve 1 se  $y$  codifica ciclo hamiltoniano de  $G$
- $A(x, y)$  executa em tempo polinomial

Então,  $A$  verifica HAM-CYCLE em tempo polinomial.

# Classe NP

- NP se refere a **Não-determinístico Polinomial** (baseada no modelo computacional não-determinístico)
- Usaremos a definição através de certificados

## Definição (Classe NP)

$NP = \{L \subseteq \Sigma^* : L \text{ é verificada por um algoritmo de tempo polinomial}\}$

Isto é,  $L \in NP$  se e somente se existe algoritmo  $A$  de tempo polinomial e constante  $c$  tal que

$$L = \{x \in \Sigma^* : \text{ existe certificado } y \text{ tal que } |y| = O(|x|^c) \text{ e } A(x, y) = 1\}$$

**A classe NP é gigantesca:**

- Contém muitos problemas de decisão associados a busca/otimização
- Em geral, o certificado é a própria solução do problema de busca
- Para estar em NP, basta verificar o certificado eficientemente

# Classe NP

**Exemplo:** Considere a linguagem dos grafos hamiltonianos CH.

$$\text{CH} = \{\langle G \rangle : G \text{ é um grafo hamiltoniano.}\}$$

Vimos que para todo grafo  $G$

- Se  $G$  é hamiltoniano, existe certificado  $S$  tal que
- $S$  é ciclo hamiltoniano e tem tamanho polinomial
- há algoritmo  $A$  que verifica  $S$  em tempo polinomial:  
 $A(x, y) = 1$ , para  $x = \langle G \rangle$  e  $y = \langle S \rangle$   
 $A(x, y) = 0$ , para todo  $x \notin \text{CH}$  e  $y \in \Sigma^*$ .

## Lema

*A linguagem dos grafos hamiltonianos pertence a NP (CH  $\in$  NP).*

# Classe NP

Lema

$P \subseteq NP$ .

*Prova.* (esboço)

- Seja  $L \in P \Rightarrow$  Existe algoritmo  $A$  que decide  $L$  em tempo polinomial
- Seja  $A'$  o seguinte algoritmo  
Algoritmo  $A'(x, y)$ , para  $x, y \in \Sigma^*$ 
  1. Devolva  $A(x)$  # o algoritmo ignora  $y$ .
- $A'$  executa em tempo polinomial
- $A'(x, y)$ , para  $y = \varepsilon$ , verifica corretamente em tempo polinomial.
- Portanto  $L \in NP$ .

Com isso  $P \subseteq NP$

□

Prêmio de 1 Milhão de dólares:  $P = NP?$

## Classe co-NP

Classe dos problemas com *certificado curto e verificável eficientemente* para a resposta NÃO.

### Definição (Classe co-NP)

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

I.e., para cada  $x \notin L$ , existe

- Algoritmo verificador  $A$  de tempo polinomial;
- Certificado  $y$  de tamanho polinomial em  $x$

tal que  $A(x, y) = 1$ .

**Exemplo:** Considere a linguagem dos grafos não-hamiltonianos (NH)

$$\text{NH} = \{\langle G \rangle : G \text{ não é um grafo hamiltoniano.}\}$$

### Lema

**NH  $\in$  co-NP.**

**Problema em aberto (há várias décadas):** **co-NP = NP?**

## Classe co-NP

**Exemplo:** Considere a linguagem dos números primos:

$$\text{Primo} = \{ \langle n \rangle : n \text{ é um número primo.} \}$$

Lema

**Primo**  $\in$  co-NP.

- Seja  $x \notin \text{Primo}$ .
- Se  $x \notin \text{Primo}$ , então é composto e há  $y$  que divide  $x$  com  $1 < y < x$ .
- $y$  é um certificado que  $x \notin \text{Primo}$  ( $y$  é um divisor de  $x$ ).
- $y$  tem tamanho polinomial (de fato linear, pois  $|\langle y \rangle| \leq |\langle x \rangle|$ ).
- Um algoritmo verificador só teria que dividir  $x$  por  $y$  para verificar que  $x \notin \text{Primo}$
- Tal algoritmo verificador pode ser implementado em tempo polinomial.



# Classes de Complexidade

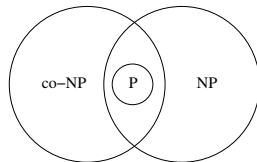
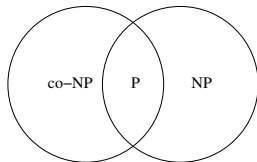
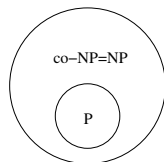
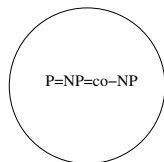
**P** =  $\{L \subseteq \Sigma^* : \text{existe alg. pol. } A \text{ que decide } L\}$

**NP** =  $\{L \subseteq \Sigma^* : \text{existe alg. pol. } A \text{ que verifica } L, \text{ ou seja,}$   
para cada  $x \in L$  existe  $y \in \Sigma^*$  de tam. pol.  
tal que  $A(x,y)=1\}$

**co-NP** =  $\{L \subseteq \Sigma^* : \text{existe alg. pol. } A \text{ que verifica } \bar{L}, \text{ ou seja,}$   
para cada  $x \notin L$  existe  $y \in \Sigma^*$  de tam. pol.  
tal que  $A(x,y)=1\}$

# Classes de Complexidade

Possíveis configurações das classes P, NP e co-NP:



# NP-Completeness and Polynomial Time Reductions

Um dos principais motivos que se acredita que  $P \neq NP$

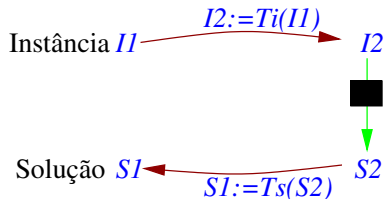
- Sabemos que a classe **NP é gigantesca!**
- Mas um subconjunto de problemas **NP-Completo  $\subseteq$  NP concentra toda a dificuldade** de NP.
- Se **um** problema de **NP-Completo é resolvido eficientemente, todos de NP também são.**

A Classe NP-Completo será construída por reduções (de Karp)

# NP-Completeness and Polynomial Time Reductions

**Reduções de Karp:**  $P_1$  é redutível a  $P_2$  ( $P_1 \leq P_2$ ) se

- $\exists T_i$  que transforma instância  $I_1$  de  $P_1$  para instância  $I_2$  de  $P_2$
- $\exists T_s$  que transforma solução  $S_2$  de  $I_2$  para solução  $S_1$  de  $I_1$



- Se  $T_i$  e  $T_s$  são de tempo polinomial, temos uma redução de tempo polinomial

# NP-Compleitude e Reduções de Tempo Polinomial

Para problemas de decisão / linguagens, simplificamos:

Linguagem  $L_1$  é *reduzível polinomialmente* à linguagem  $L_2$ ,

- denotado por  $L_1 \leq_P L_2$ , se
- existe função  $f$  computável
- em tempo polinomial tal que

$$x \in L_1 \text{ se e somente se } f(x) \in L_2 \quad \forall x \in \Sigma^*$$

- Isto é, ao respondermos se  $f(x) \in L_2$  temos a resposta se  $x \in L_1$ .

# NP-Completeness and Polynomial Time Reductions

## Lema

Se  $L_1$  e  $L_2$  são linguagens tal que  $L_1 \leq_P L_2$  e  $L_2 \in P$ , então  $L_1 \in P$ .

*Prova.* Vamos mostrar que existe algoritmo  $A_1$  que decide  $L_1$ .

- Se  $L_2 \in P$  então existe algoritmo  $A_2$  que decide  $L_2$ .
- Se  $L_1 \leq_P L_2$  então existe função  $f$  computável polinomialmente tal que

$$x \in L_1 \iff f(x) \in L_2$$

- Seja  $A_1$  o algoritmo tal que  $A_1(x)$  devolve  $A_2(f(x))$ .
- Note que  $x \in L_1$  se e somente se  $A_1(x) = 1$ , pois

$$x \in L_1 \iff f(x) \in L_2 \iff A_2(f(x)) = 1 \iff A_1(x) = 1$$

□

# NP-Completeness and Polynomial Time Reductions

## Class NP-Difícil:

### Definição

Uma linguagem  $L \subseteq \{0, 1\}^*$  pertence a NP-Difícil se:

$$L' \leq_P L \text{ para todo } L' \in \text{NP}$$

**Obs.:** Uma linguagem em NP-Difícil não necessariamente está em NP.

## Class NP-Completo:

### Definição

Uma linguagem  $L \subseteq \{0, 1\}^*$  pertence a NP-Completo se:

$$L \in \text{NP} \cap \text{NP-Difícil}$$

# NP-Completeness and Polynomial Time Reductions

## Theorem

*If any language in NP-Complete is decided in polynomial time, then  $P = NP$ .*

*If any language in NP cannot be decided in polynomial time, then no language in NP-Complete is polynomially decidable.*

*Proof.* [Follows from the previous lemma]

Suppose that  $L$  is a language such that  $L \in P \cap \text{NP-Complete}$ .

For every  $L' \in \text{NP}$ , we will show that  $L' \in P$ .

$L' \in \text{NP} \Rightarrow L' \leq_P L$

# pois

$L \in \text{NP-Complete}$

$\Rightarrow \exists f : (x \in L' \iff f(x) \in L)$

#  $f$  eficiente

$\Rightarrow \exists A : (x \in L' \iff A(f(x)) = 1)$

#  $L \in P$  e  $A$  eficiente

$\Rightarrow L' \in P$

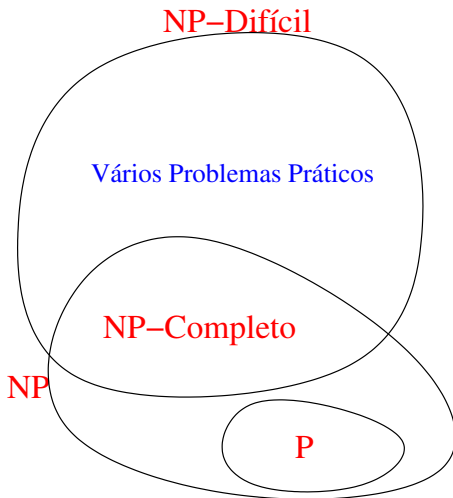
The other case is the contrapositive.

□



# NP-Compleitude e Reduções de Tempo Polinomial

Como acredita-se que seja a relação das classes:



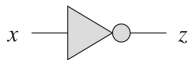
# Problema CIRCUIT-SAT

## Portas Lógicas e Operadores Lógicos:

(a) NOT denotado por  $\neg$ .

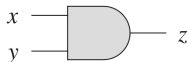
(b) AND denotado por  $\wedge$ .

(c) OR denotado por  $\vee$ .



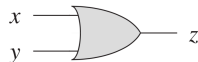
$x$	$\neg x$
0	1
1	0

(a)



$x$	$y$	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

(b)



$x$	$y$	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

(c)

Na figura,  $x$  e  $y$  são fios de entrada e  $z$  o fio de saída da porta.

Resultado de cada porta depende dos fios de entrada conforme tabelas

# Problema CIRCUIT-SAT

**Circuito Combinatorial Lógico:** composto por portas lógicas e fios

- **Fios** podem ser de

**ligação:** conectam a saída de um porta com a entrada de outra (transferem valores da saída de uma porta para a entrada de outra).

**entrada:** tem uma extremidade livre e ligam na entrada de uma porta (recebem valor externo e repassam para a entrada de uma porta)

**saída:** conectam a saída de uma porta e tem uma extremidade livre (resultado do circuito)

## Circuito Combinatorial Lógico

- Possuem pelo menos um fio de entrada
- Possuem exatamente um fio de saída (na prática podem ter mais)
- São acíclicos (uma porta nunca depende do seu próprio resultado)

# Problema CIRCUIT-SAT

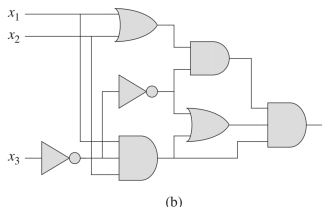
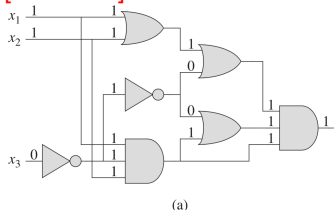
**Definição.** Seja  $C$  um circuito combinatorial lógico.

Uma **atribuição** para  $C$  é uma atribuição 0 ou 1 para seus fios de entrada

Uma **atribuição verdadeira** para  $C$  é uma atribuição que resulta em saída 1

$C$  é **satisfazível** se possui atribuição verdadeira

**Exemplo [CLRS'09]:**



O circuito (a) é satisfazível e o circuito (b) não é.

**Exercício:** Apresente um algoritmo com complexidade de tempo  $O(2^n(n + m))$  que diz se um circuito é satisfazível ou não, onde  $n$  é o número de portas e  $m$  o número de ligações.

## Problema CIRCUIT-SAT

A linguagem que contempla os circuitos satisfatíveis é dado por

**CIRCUIT-SAT** =  $\{\langle C \rangle : C \text{ é um circuito combinatorial lógico satisfazível}\}$

### Lema

CIRCUIT-SAT  $\in$  NP.

*Prova.*

- Se  $x \in \text{CIRCUIT-SAT}$ , então existe atribuição verdadeira  $y$  para  $x$  (o certificado é a própria atribuição verdadeira  $y$ .)

Algoritmo  $A(x, y)$  - esboço

1. Faça ordenação topológica das portas (em tempo linear)
  2. Simule/percorra o circuito  $x$  em tempo linear com a entrada  $y$ .
  3. Devolva o resultado do circuito.
- Se  $x \notin \text{CIRCUIT-SAT}$ , não há atribuição que resulte em saída 1 (neste caso,  $A(x, y) = 0$  para qualquer  $y$ .)



# Problema CIRCUIT-SAT

## Lema

CIRCUIT-SAT  $\in$  NP-Difícil.

- Prova possui detalhes técnicos que fogem ao escopo do curso.
- Daremos um esboço da prova deste lema posteriormente,
- considerando conhecimento do hardware de um computador.

Dos dois lemas anteriores, segue que

## Teorema

CIRCUIT-SAT  $\in$  NP-Completo.

# Problema CIRCUIT-SAT

Temos nosso primeiro problema NP-Completo:

**CIRCUIT-SAT**  $\in$  NP-Completo

Agora, para provar que  $L \in \text{NP}$  é NP-Completo, temos duas opções:

(A) Provar que **CIRCUIT-SAT**  $\leq_P L$  ou

(B) Provar que **para todo**  $L' \in \text{NP}$  temos  $L' \leq_P L$

Seguiremos pela opção (A).

# NP-Completeness

De maneira geral: **Para provar que  $L \in \text{NP-Completo}$**  fazemos

- 1 Prove que  $L \in \text{NP}$
- 2 **Selecione** um problema  $L_{\text{NPC}} \in \text{NP-Completo}$  (mais próximo de  $L$ )
- 3 Prove que  $L_{\text{NPC}} \leq_P L$ 
  - 1 Apresente algoritmo  $f$  que transforma entrada de  $L_{\text{NPC}}$  para  $L$
  - 2 Prove que  $x \in L_{\text{NPC}}$  se e somente se  $f(x) \in L$ , para todo  $x$
  - 3 Prove que  $f$  executa em tempo polinomial



# NP-Compleitude

## Lema

Seja  $L$  uma linguagem. Se

- existe  $L_{NPC} \in NP\text{-Completo}$  e
- $L_{NPC} \leq_P L$

então  $L \in NP\text{-Difícil}$ .

Se além disso,  $L \in NP$  então  $L \in NP\text{-Completo}$ .

*Prova.* Como  $L_{NPC}$  é NP-Completo, então

- para todo  $L' \in NP$  temos  $L' \leq_P L_{NPC}$ .
- Como  $L_{NPC} \leq_P L$  então (exercício)  $L' \leq_P L$  para todo  $L' \in NP$ .
- Portanto  $L \in NP\text{-Difícil}$

Se além disso  $L \in NP$ , por definição, temos  $L \in NP\text{-Completo}$  □

# Problema SAT

## Entrada:

Fórmula lógica  $\phi(x_1, \dots, x_n)$

com variáveis lógicas  $x_1, \dots, x_n$ , operadores lógicos, parênteses sem quantificadores

## Pergunta:

Existe atribuição de valores 0/1 às variáveis  $x_1, \dots, x_n$  de maneira a tornar  $\phi$  verdadeira?

## Operadores

- 1 AND ( $\wedge$ ).
- 2 OR ( $\vee$ ).
- 3 NOT ( $\neg$ ).
- 4 Implicação ( $\rightarrow$ ).
- 5 Se e Somente Se ( $\leftrightarrow$ ).

# Problema SAT

## Operadores Lógicos

$x$	$\neg x$
0	1
1	0

Tabela verdade do operador unário  $\neg$

$x$	$y$	$x \wedge y$	$x \vee y$	$x \rightarrow y$	$x \leftrightarrow y$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

Tabela verdade dos operadores binários  $\wedge$ ,  $\vee$ ,  $\rightarrow$  e  $\leftrightarrow$

# Problema SAT

**Exemplo:**  $f(x_1, x_2, x_3) = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$

Atribuição:  $x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1$

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

## Definição

Dada fórmula  $f(x_1, \dots, x_n)$ , dizemos que  $f$  é *satisfazível* se existe atribuição  $y = (y_1, \dots, y_n) \in \{0, 1\}^n$  tal que  $f(y) = 1$ .

# Problema SAT

Linguagem que contempla fórmulas que admitem atribuição verdadeira

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

## Lema

SAT  $\in$  NP.

*Prova.*

- Seja  $f(x_1, \dots, x_n)$  fórmula do SAT que é satisfazível.
- Então, existe atribuição  $y = (y_1, \dots, y_n)$  para  $(x_1, \dots, x_n)$ , onde  $y_i \in \{0, 1\}$ , tal que  $f(y) = 1$ .
- $y$  é o certificado e
- é possível implementar verificador eficiente que testa se  $f(y) = 1$  (exercício). □

# Problema SAT

## Lema

SAT  $\in$  NP-Difícil.

### *Prova.*

Vamos mostrar que CIRCUIIT-SAT  $\leq_P$  SAT. Para isso

mostraremos algoritmo  $T$  que transforma circuitos em fórmulas tal que

- $C \in \text{CIRCUIIT-SAT}$  se e somente se  $T(C) \in \text{SAT}$ ,  $\forall$  circuito  $C$
- $T$  executa em tempo polinomial.

A seguir, apresentamos alguns passos do algoritmo  $T$ .

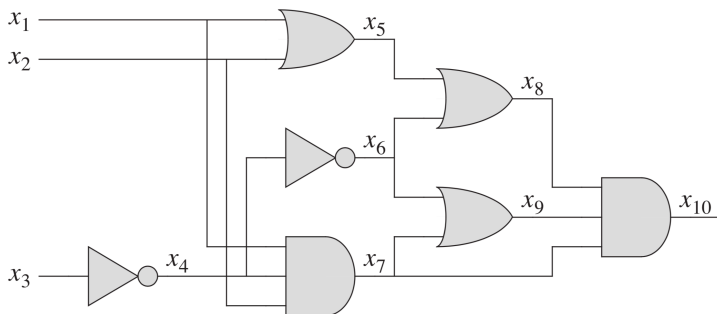
# Problema SAT

## Continuação da Prova.

Seja  $C$  um circuito combinatorial lógico.

Criamos uma variável  $x_i$  para cada fio  $i$  de  $C$

## Exemplo [CLRS'09]:



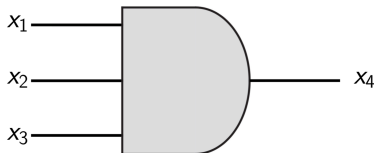
# Problema SAT

## Continuação da Prova.

Seja  $x_i$  a variável/fio resultante da porta lógica  $i$ . Geramos cláusula  $c_i$ :

- obtenha fórmula  $\phi_i$  que computa a saída de  $i$  pelos fios de entrada
- obtenha fórmula  $c_i$  que força equivalência de  $x_i$  com  $\phi_i$

## Exemplo:



$$c_4 = (x_4 \leftrightarrow (x_1 \wedge x_2 \wedge x_3))$$

Com isso,  $x_4$  terá o valor igual a saída da porta lógica correspondente.

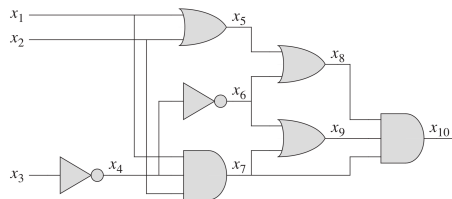


# Problema SAT

Continuação da Prova. A fórmula final é obtida

- Fazendo **conjunção** das fórmulas/cláusulas  $c_i$ 's, para cada porta  $i$ , e
- com a **saída do circuito** (correspondência com circuitos satisfatíveis)

Exemplo [CLRS'09]:



$$\begin{aligned} f = x_{10} &\wedge (x_4 \leftrightarrow (\neg x_3)) \\ &\wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ &\wedge (x_6 \leftrightarrow (\neg x_4)) \\ &\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ &\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\ &\wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ &\wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)) \end{aligned}$$

# Problema SAT

## Continuação da Prova.

- O algoritmo que produz a fórmula é de tempo polinomial (exercício)
- Dado circuito  $C$  seja  $f$  a fórmula obtida pelo algoritmo
- Se  $C$  é satisfazível, existe atribuição dos fios de entrada que resulta em 1:
- Atribua o valor do fio  $i$  para a correspondente variável  $x_i$
- A cláusula  $c_i$  garante valor correto para saída da porta  $i$
- A conjunção com variável do fio resultante garante que  $f$  é verdadeira
- e com isso temos que  $f$  é satisfeita
- Se  $f$  é satisfazível –de maneira análoga– temos  $C$  satisfazível □

## Teorema

SAT  $\in$  NP-Completo.

*Prova.* Segue dos dois lemas anteriores. □

# Problema 3-CNF-SAT

## Definição (fórmula em CNF)

Uma fórmula  $\phi$  está na forma **CNF** (forma normal conjuntiva) se

- $\phi$  pode ser expressa como uma conjunção ( $\wedge$ ) de cláusulas, onde
- uma **cláusula** é uma disjunção ( $\vee$ ) de uma ou mais literais, e
- uma **literal** é uma variável ou sua negação.

**Exemplo:**[Fórmula em CNF]

$$\phi(x_1, x_2, x_3) = (x_1) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3).$$

# Problema 3-CNF-SAT

## Definição (fórmula em 3-CNF-SAT)

Uma fórmula  $\phi$  está em **3-CNF** se

- $\phi$  está na forma CNF e
- cada cláusula possui exatamente 3 literais distintas.

**Exemplo:**[Fórmula em 3-CNF-SAT]

$$\phi(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee \neg x_3).$$

# Problema 3-CNF-SAT

## Problema (3-CNF-SAT)

*Dada fórmula  $\phi(x_1, \dots, x_n)$  em 3-CNF, decidir se existe atribuição para as variáveis de maneira a tornar  $\phi$  verdadeira.*

Em termos de linguagem:

$$\mathbf{3\text{-CNF-SAT}} = \{ \langle f \rangle : f \text{ é fórmula em 3-CNF satisfazível} \}$$

## Lema

3-CNF-SAT  $\in$  NP.

*Prova.* Exercício.

□

# Problema 3-CNF-SAT

## Lema

3-CNF-SAT  $\in$  NP-Difícil.

*Prova.* Vamos mostrar que  $\text{SAT} \leq_P \text{3-CNF-SAT}$ .

Dada fórmula  $f_{\text{SAT}}$  para CNF vamos construir em tempo polinomial fórmula  $f_{\text{3CNF}}$  para 3-CNF-SAT tal que

$f_{\text{SAT}} \in \text{SAT}$  se e somente se  $f_{\text{3CNF}} \in \text{3-CNF-SAT}$

$f_{\text{3CNF}}$  será construída em três etapas de transformação equivalentes.

- T1:**  $f_{\text{SAT}}$  é transformada em fórmula  $f_{\text{T1}}$  que é conjunção de cláusulas, cada cláusula contém no máximo 3 variáveis.
- T2:** Cada cláusula de  $f_{\text{T1}}$  é substituída por conjunção de cláusulas na forma CNF, obtendo  $f_{\text{T2}}$
- T3:** Cada cláusula de  $f_{\text{T2}}$  é substituída por conjunção de cláusulas com exatamente 3 variáveis cada, obtendo  $f_{\text{3CNF}}$ .

# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T1

A transformação  $T1$  irá considerar uma árvore de avaliação:

**Árvore de avaliação:** Representação de fórmula lógica por árvore binária

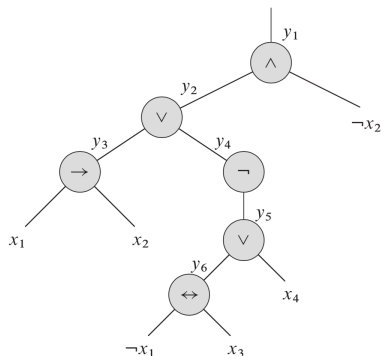
- Se  $N$  é nó folha,  $N$  representa fórmula de um literal dado por  $v_N$
- Cada nó interno  $N$  tem um operador lógico  $f_N$ .
- Se  $f_N = \neg$  então  $N$  tem um filho  $N.dir$  e sua fórmula é dada por  $\neg(expr(N.dir))$ , onde  $expr(N.dir)$  é a fórmula lógica do nó  $N.dir$ .
- Se  $f_N$  é op. binário,  $N$  tem nós filhos  $N.esq$  e  $N.dir$  e sua fórmula é dada por  $((expr(N.esq)) f_N (expr(N.dir)))$ , onde  $expr(N.esq)$  e  $expr(N.dir)$  são as fórmulas lógicas de seus filhos da esquerda e direita, resp.
- A fórmula representada pela árvore é dada pela fórmula da raiz.

# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T1

Dada fórmula  $f_{SAT}$ , construir sua árvore de avaliação  $A(f_{SAT})$ .

Exemplo [CLRS'09]:  $\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$



## Fato

A árvore  $A(f_{SAT})$  de  $f_{SAT}$  pode ser construída em tempo polinomial (exercício).



# Problema 3-CNF-SAT

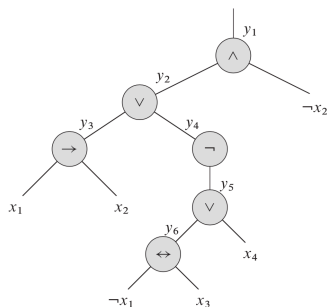
## Continuação da Prova: Transformação T1

Podemos considerar a árvore  $A(f_{SAT})$  como um circuito que transformamos para uma fórmula  $f_{T1}$  (como na redução  $CIRCUIT-SAT \leq_P SAT$ )

- Cada fio que liga uma folha é representado pela respectiva literal
- Cada fio que liga nós internos é o resultado de um operador lógico
- O fio que representa o resultado da raiz é o resultado do circuito.

Exemplo [CLRS'09]:

$$\begin{aligned}f_{T1} = & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\ & \wedge (y_4 \leftrightarrow (\neg y_5)) \\ & \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\end{aligned}$$



# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T1

### Fato

A fórmula  $f_{T1}$  pode ser obtida de  $A(f_{SAT})$  em tempo polinomial (exercício).

Obs.: A fórmula  $f_{T1}$  é uma conjunção de cláusulas:

$$f_{T1} = c_1 \wedge c_2 \wedge \dots \wedge c_m,$$

t.q. cláusula  $c_i$  é a fórmula de equivalência associada ao nó (operador)  $i$  ou ao fio resultante do nó raiz.

Obs.: Cada cláusula  $c_i$  tem no máximo 3 variáveis, mas não necessariamente está na forma CNF.

### Fato

$f_{SAT}$  é satisfazível se e somente se  $f_{T1}$  é satisfazível (exercício).

# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T2

T2: Substituímos cada cláusula  $c_i$  em  $f_{T1}$  por uma equivalente em CNF.

### Algoritmo T2

- 1 Para cada cláusula  $c_i$ , construímos a tabela verdade com todos os valores possíveis de suas variáveis e o resultado para  $c_i$ .
- 2 Construímos a fórmula normal disjuntiva (disjunção de  $\wedge$ 's) das entradas com resultado 0.
- 3 Construímos a negação da fórmula anterior
- 4 Distribuimos a negação, aplicando a Lei DeMorgan e obtendo fórmula  $d_i$  em CNF

## Problema 3-CNF-SAT

Continuação da Prova: Transformação T2

Exemplo [CLRS'09] Seja a cláusula  $c_i = (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$ .

$y_1$	$y_2$	$x_2$	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	<b>0</b>
1	1	0	1
1	0	1	<b>0</b>
1	0	0	<b>0</b>
0	1	1	1
0	1	0	<b>0</b>
0	0	1	1
0	0	0	1

Construindo do jeito tradicional (pelas entradas 1's) obtemos fórmula em **Forma Normal Disjuntiva** (e queremos em **Forma Normal Conjuntiva**).

Então, vamos primeiro obter fórmula de  $\neg c_i$  com entradas **0**'s e negar:

$$\neg c_i = (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$$

## Problema 3-CNF-SAT

### Continuação da Prova: Transformação T2

Para obter a fórmula  $c_i$ , é equivalente obter  $\neg(\neg c_i)$ .

$$\begin{aligned}c_i &= \neg(\neg c_i) \\ &= \neg\left((y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)\right) \\ &= \neg(y_1 \wedge y_2 \wedge x_2) \wedge \neg(y_1 \wedge \neg y_2 \wedge x_2) \wedge \neg(y_1 \wedge \neg y_2 \wedge \neg x_2) \wedge \neg(\neg y_1 \wedge y_2 \wedge \neg x_2) \\ &= (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)\end{aligned}$$

Seja  $d_i$  a fórmula em CNF obtida de  $c_i$ .

Seja  $f_{T2}$  a conjunção das fórmulas  $d_i$ 's.

### Fato

$f_{T1}$  é satisfazível se e somente se  $f_{T2}$  é satisfazível (exercício).

# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T2

### Fato

$f_{T2}$  é obtida em tempo polinomial.

*Prova.* Seja  $c_i$  cláusula de  $f_{T1}$  que foi transformada em fórmula  $d_i$  em  $f_{T2}$ .

Como  $c_i$  tem no máximo 3 variáveis distintas, a tabela verdade usada para obter  $d_i$  tem no máximo 8 linhas (valor constante).

Portanto,  $d_i$  tem no máximo 8 cláusulas, cada uma com 3 literais.

Assim,  $d_i$  tem tamanho linear em relação a  $c_i$ . □

# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T3

$f_{T2}$  é uma fórmula em CNF, dada por uma conjunção de cláusulas:

$$f_{T2} = e_1 \wedge \dots \wedge e_k$$

onde  $e_i$  é uma disjunção de literais.

Obs.:  $e_i$  tem no máximo 3 literais e está na forma CNF.

A transformação  $T3$  troca cada cláusula  $e_i$  com menos que 3 literais para uma correspondente fórmula em CNF com 3 literais em cada cláusula.

# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T3

**Exemplo:** Considere a cláusula com apenas uma literal:  $(x_1)$   
Podemos aumentar o número de literais por cláusula usando uma outra variável, digamos  $p$ :

$(x_1)$  é satisfazível se e somente se  $(x_1 \vee p) \wedge (x_1 \vee \neg p)$  é satisfazível.

**Exemplo:** Considere a cláusula com duas literais:  $(x_1 \vee x_2)$   
Podemos aumentar o número de literais por cláusula usando outra variável, digamos  $q$ :

$(x_1 \vee x_2)$  é satisfazível  $\leftrightarrow (x_1 \vee x_2 \vee q) \wedge (x_1 \vee x_2 \vee \neg q)$  é satisfazível.

### Podemos usar

variável  $p$  para aumentar cada cláusula com uma para duas literais.  
variável  $q$  para aumentar cada cláusula com duas para três literais  
(e garantimos que cada cláusula final tem exatamente 3 literais)



# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T3

### Algoritmo $T3(f_{T2})$

- 1  $f \leftarrow f_{T2}$
- 2 Adicione mais duas variáveis lógicas, dadas por  $p$  e  $q$  em  $f$
- 3 Enquanto  $f$  não é entrada válida para 3-CNF-SAT faça
- 4   Seja  $C$  cláusula com menos que 3 literais
- 5   Se  $C$  tem 1 literal, digamos  $l$ , então
- 6     Troque  $C$ , em  $f$ , pela fórmula  $(l \vee p) \wedge (l \vee \neg p)$
- 7   Se  $C$  tem 2 literais, digamos  $l_1$  e  $l_2$ , então
- 8     Troque  $C$ , em  $f$ , pela fórmula  $(l_1 \vee l_2 \vee q) \wedge (l_1 \vee l_2 \vee \neg q)$
- 9  $f_{3\text{CNF}} \leftarrow f$
- 10 Devolva  $f_{3\text{CNF}}$

# Problema 3-CNF-SAT

## Continuação da Prova: Transformação T3

### Fato

- $f_{3\text{CNF}}$  é obtida em tempo polinomial.
- $f_{T2}$  é satisfazível se e somente se  $f_{3\text{CNF}}$  é satisfazível.

*Prova.* Seja  $C$  uma cláusula em  $f_{T2}$  com menos que 3 literais.

- O algoritmo  $T3$  troca  $C$  por uma expressão equivalente com
- 8 cláusulas se  $C$  tem apenas uma literal, ou
- 4 cláusulas se  $C$  tem duas literais.
- Cada uma destas cláusulas finais tem exatamente 3 literais.

Assim,  $C$  foi trocada por expressão de tamanho linear em relação a  $C$ .

E  $T3$  pode ser implementado para executar em tempo linear. □

Fim da prova do Lema: 3-CNF-SAT  $\in$  NP-Difícil. □

### Teorema

3-CNF-SAT  $\in$  NP-Completo.

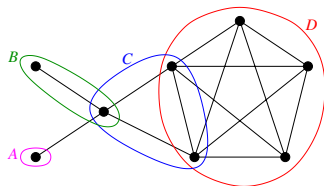
# Problema CLIQUE

## Definição

Dado grafo não-direcionado  $G = (V, E)$ , um conjunto de vértices  $C \subseteq V$  é uma **clique** se qualquer par de vértices distintos de  $C$  estão conectados por aresta em  $E$ .

O **tamanho da clique**  $C$  é sua cardinalidade.

Exemplo:



Cada um dos conjuntos de vértices  $A$ ,  $B$ ,  $C$  e  $D$  forma uma clique.

# Problema CLIQUE

Versão de Otimização:

Problema (MAX-CLIQUE )

Dado grafo não-direcionado  $G = (V, E)$ , encontrar clique  $C \subseteq V$  de  $G$  de cardinalidade máxima.

Versão de Decisão/Linguagem: Usamos parâmetro  $k$

**CLIQUE** =  $\{\langle G, k \rangle : G \text{ é um grafo que possui uma clique de tamanho pelo menos } k\}$

Lema

CLIQUE  $\in$  NP

Prova. Exercício.

□

# Problema CLIQUE

## Lema

CLIQUE  $\in$  NP-Difícil

### Prova.

Vamos fazer a redução  $3\text{-CNF-SAT} \leq_P \text{CLIQUE}$ .

Seja  $f$  uma fórmula qualquer para 3-CNF-SAT.

Vamos construir grafo  $G$  e definir parâmetro  $k$  tal que

$f \in 3\text{-CNF-SAT}$  se e somente se  $\langle G, k \rangle \in \text{CLIQUE}$

# Problema CLIQUE

## Continuação da Prova.

Transformação de fórmula  $f$  para  $\langle G, k \rangle$ , com  $k = m$

Seja  $f = C_1 \wedge C_2 \wedge \dots \wedge C_m$  fórmula em 3-CNF.

Denote por  $l_1^i, l_2^i$  e  $l_3^i$  as três literais da cláusula  $C_i$ .

O parâmetro  $k$  é igual ao número de cláusulas  $m$ .

O grafo  $G = (V, E)$  é definido com:

- conjunto  $V$  de vértices formado por  $3m$  vértices  
Para cada literal  $l_j^i$ , defina um vértice  $v_j^i$ :

$$V = \{v_j^i : i \in \{1, \dots, k\} \text{ e } j \in \{1, 2, 3\}\}$$

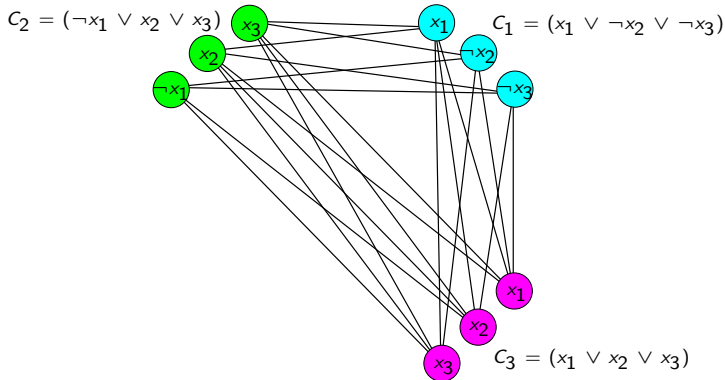
- conjunto  $E$  de arestas definido como

$$E = \{ \{u, v\} : u \text{ e } v \text{ não vieram de literais da mesma cláusula e a literal de } u \text{ não é a negação da literal de } v \}$$

# Problema CLIQUE

Continuação da Prova.

Exemplo da construção do grafo



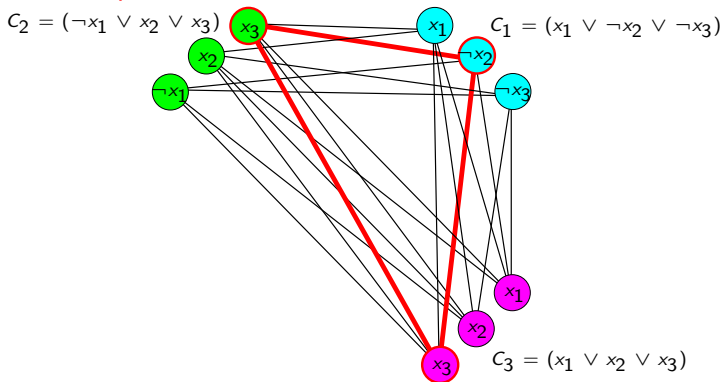
Grafo para fórmula  $f = C_1 \wedge C_2 \wedge C_3$ , onde

$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$ ,  $C_2 = (\neg x_1 \vee x_2 \vee x_3)$  e  $C_3 = (x_1 \vee x_2 \vee x_3)$

# Problema CLIQUE

Continuação da Prova.

Exemplo com clique



Grafo para fórmula  $f = C_1 \wedge C_2 \wedge C_3$ , onde

$C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$ ,  $C_2 = (\neg x_1 \vee x_2 \vee x_3)$  e  $C_3 = (x_1 \vee x_2 \vee x_3)$

Clique em vermelho tem literais:  $\{x_3, \neg x_2\}$



# Problema CLIQUE

## Continuação da Prova.

**Fato.** A construção de  $G$  é feita em tempo polinomial (exercício).

Vamos mostrar

$f \in 3\text{-CNF-SAT}$  se e somente se  $\langle G, m \rangle \in \text{CLIQUE}$

( $\Rightarrow$ ): Considere  $f \in 3\text{-CNF-SAT}$

Se  $f$  é satisfeita, existe atribuição  $y$  tal que  $f(y) = 1$  e  $y$  torna verdadeira pelo menos uma literal  $l_i \in \{l_1^i, l_2^i, l_3^i\}$  de cada cláusula  $C_i$ .

Seja  $v_i$  o vértice correspondente a literal  $l_i$  e  $C = \{v_1, v_2, \dots, v_m\}$ .

Temos que  $C$  é uma clique de tamanho  $m$ , pois

- $l_i$  e  $l_j$  ficam verdadeiras na atribuição  $y$  e
- $v_i$  e  $v_j$ , por construção, possuem aresta ligando-os (só não há arestas entre literais que se complementam)

Portanto  $\langle G, k \rangle \in \text{CLIQUE}$ .

## Problema CLIQUE

**Continuação da Prova.** ( $\Leftarrow$ ): Considere que  $\langle G, m \rangle \in \text{CLIQUE}$

Se  $\langle G, m \rangle \in \text{CLIQUE}$ , existe clique  $K = \{v_1, \dots, v_m\}$  de tamanho  $k$  em  $G$ , onde  $v_i \in \{v_1^i, v_2^i, v_3^i\}$ . Note que  $K$  possui um vértice de cada cláusula (pois não há arestas entre vs da mesma cláusula).

Seja  $l_i \in \{l_1^i, l_2^i, l_3^i\}$  a correspondente literal de  $v_i$ .

Seja  $y$  atribuição das variáveis, fazendo literais  $l_i = 1$ , para  $i = 1, \dots, m$ . (É possível fazer  $l_i = 1$ , pois não há atribuições de literais conflitantes)  
Complete a atribuição  $y$  atribuindo 0 para variáveis ainda sem atribuição.

**Fato.** Vale que  $f(y) = 1$  (cada cláusula tem pelo menos 1 literal verdadeira)

Portanto  $f \in 3\text{-CNF-SAT}$ .

Fim da prova do Lema:  $\text{CLIQUE} \in \text{NP-Difícil}$ . □

### Teorema

$\text{CLIQUE} \in \text{NP-Completo}$ .

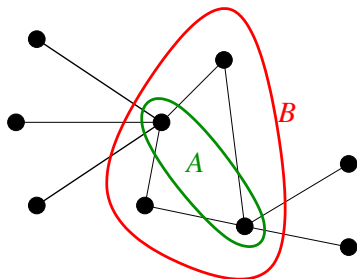
# Problema VERTEX-COVER

## Definição

Dado grafo não-direcionado  $G = (V, E)$ , um conjunto de vértices  $C \subseteq V$  é uma **cobertura por vértices** se qualquer aresta  $e \in E$  tem pelo menos uma das extremidades em  $C$ .

O **tamanho da cobertura por vértices**  $C$  é sua cardinalidade.

Exemplo:



Cada um dos conjuntos de vértices  $A$  e  $B$  forma cobertura por vértices.

# Problema VERTEX-COVER

Versão de Otimização:

Problema (MIN-VERTEX-COVER )

*Dado grafo não-direcionado  $G = (V, E)$ , encontrar cobertura por vértices  $C \subseteq V$  de  $G$  de cardinalidade mínima.*

Versão de Decisão/Linguagem: Usamos parâmetro  $k$

**VERTEX-COVER** =  $\{ \langle G, k \rangle : G \text{ é grafo que possui uma cobertura por vértices de tamanho no máximo } k \}$

Lema

VERTEX-COVER  $\in$  NP

*Prova.* Exercício. □

# Problema VERTEX-COVER

## Lema

VERTEX-COVER  $\in$  NP-Difícil.

## Prova.

Vamos fazer a redução **CLIQUE**  $\leq_P$  VERTEX-COVER.

- Dado um grafo  $G = (V, E)$  compute seu complemento  $\bar{G} = (V, \bar{E})$ , onde  $\bar{E} = \{\{u, v\} : \{u, v\} \notin E\}$ .
- Vamos mostrar que

$G$  possui uma clique de tamanho  $k$

se e somente se

$\bar{G}$  possui um cobertura por vértices de tamanho  $|V| - k$ .

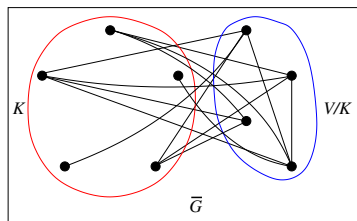
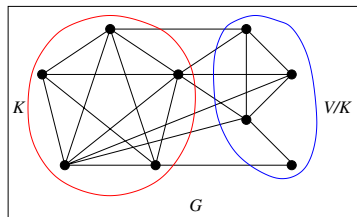
I.e. vamos mostrar que

$\langle G, k \rangle \in \text{CLIQUE}$  se e somente se  $\langle \bar{G}, |V| - k \rangle \in \text{VERTEX-COVER}$

# Problema VERTEX-COVER

Continuação da Prova.

Exemplo: Redução de CLIQUE para VERTEX-COVER



# Problema VERTEX-COVER

Continuação da Prova.

**Fato.** A construção de  $\overline{G}$  é feita em tempo polinomial (exercício).

( $\Rightarrow$ ): Considere  $\langle G, k \rangle \in \text{CLIQUE}$

I.e.,  $G = (V, E)$  possui uma clique  $K \subseteq V$  tal que  $|K| = k$ .

Seja  $C = V \setminus K$  e  $\overline{G} = (V, \overline{E})$  o grafo complementar de  $G$ .

Vamos mostrar que  $C$  é cobertura em  $\overline{G}$ .

$K$  é clique em  $G$

$\Rightarrow$  Não há arestas entre vértices de  $K$  em  $\overline{G}$

$\Rightarrow$  Todas arestas de  $\overline{G}$  tem pelo menos uma extremidade em  $C = V \setminus K$

$\Rightarrow C$  é cobertura em  $\overline{G}$

Como  $K = V \setminus C$  e  $|K| = k$

$\Rightarrow |C| = |V| - |K|$

$\Rightarrow |C| = |V| - k$

Portanto  $\langle \overline{G}, |V| - k \rangle \in \text{VERTEX-COVER}$

# Problema VERTEX-COVER

Continuação da Prova.

( $\Leftarrow$ ): Considere  $\langle \overline{G}, |V| - k \rangle \in \text{VERTEX-COVER}$

Seja  $C \subseteq V$  cobertura de tamanho  $|V| - k$  em  $\overline{G}$ .

Seja  $K = V \setminus C$ . Vamos mostrar que  $K$  é clique de tamanho  $k$  em  $G$ .

Se  $C$  é uma cobertura em  $\overline{G}$

$\Rightarrow$  Toda aresta de  $\overline{G}$  tem pelo menos uma extremidade em  $C$

$\Rightarrow$  Não há arestas ligando vértices de  $K = V \setminus C$  em  $\overline{G}$

$\Rightarrow K$  é uma clique em  $G$

Como  $|C| = |V| - k$  e  $K = V \setminus C$  temos que

$\Rightarrow |K| = |V| - |C| = |V| - (|V| - k)$

$\Rightarrow |K| = k$

Portanto  $\langle G, k \rangle \in \text{CLIQUE}$

Fim da prova do Lema:  $\text{VERTEX-COVER} \in \text{NP-Difícil}$ . □

## Teorema

$\text{VERTEX-COVER} \in \text{NP-Completo}$ .



# Problema HAM-CYCLE

## Definição

Dado grafo  $G = (V, E)$ , um *ciclo hamiltoniano* em  $G$  é um ciclo que passa por cada vértice de  $V$  exatamente uma vez.

## Definição

Um grafo  $G$  é *hamiltoniano* se  $G$  possui um ciclo hamiltoniano.

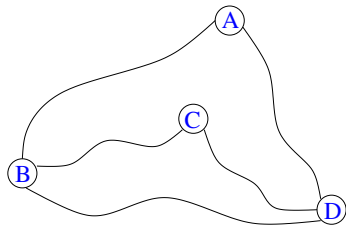
## Problema HAM-CYCLE

- **Entrada:** Grafo  $G = (V, E)$
- **Pergunta:** O grafo  $G$  é hamiltoniano?

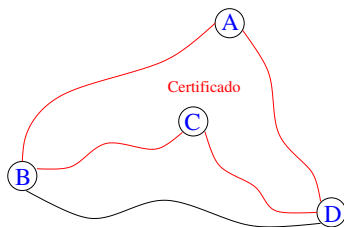
# Problema HAM-CYCLE

Exemplo:

Entrada



Resposta: Sim!



Versão de Decisão/Linguagem:

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

Lema

HAM-CYCLE  $\in$  NP

Prova. Exercício.

□

# Problema HAM-CYCLE

## Lema

HAM-CYCLE  $\in$  NP-Difícil.

## Prova.

Vamos fazer a redução  $\text{VERTEX-COVER} \leq_P \text{HAM-CYCLE}$ .

- Dado entrada  $\langle G, k \rangle$  para VERTEX-COVER vamos construir grafo  $G'$  para HAM-CYCLE tal que

$G$  possui cobertura por vértices de tamanho  $k$

se e somente se

$G'$  é hamiltoniano.

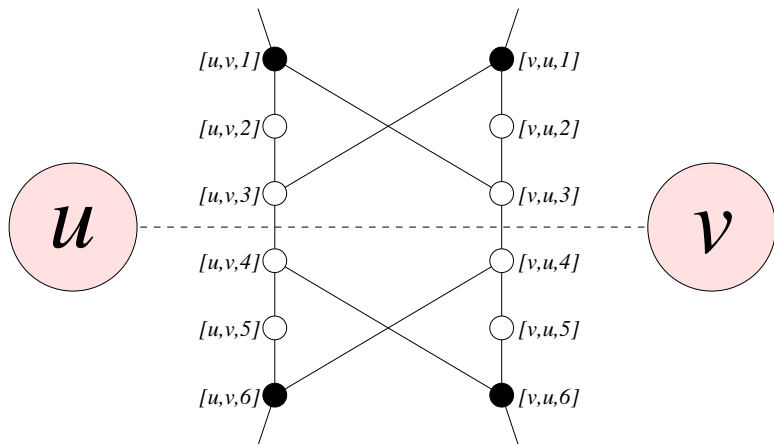
I.e. vamos mostrar que

$\langle G, k \rangle \in \text{VERTEX-COVER}$  se e somente se  $\langle G' \rangle \in \text{HAM-CYCLE}$

# Problema HAM-CYCLE

Continuação da Prova. Widget  $W_{u,v}$

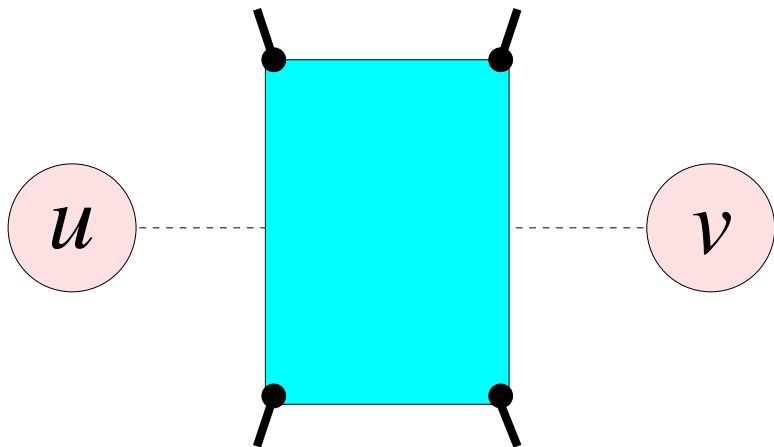
Vamos usar a estrutura (widget)  $W_{uv}$  em  $G'$  para cada  $\{u, v\} \in E$  de  $G$



# Problema HAM-CYCLE

Continuação da Prova. Representação do Widget  $W_{u,v}$

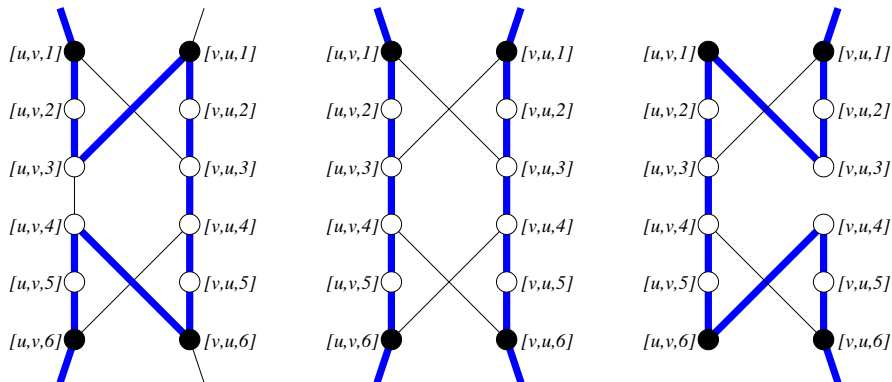
Vamos usar a estrutura (widget)  $W_{uv}$  em  $G'$  para cada  $\{u, v\} \in E$  de  $G$



## Ciclo Hamiltoniano

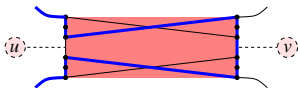
Apenas vértices  $[u,v,1]$ ,  $[u,v,6]$ ,  $[v,u,1]$  e  $[v,u,6]$  tem ligação p/ fora.

**Fato.** As únicas maneiras do circuito passar pelos vértices do widget são:

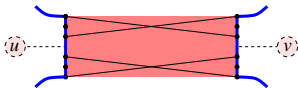


# Problema HAM-CYCLE

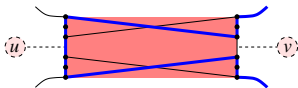
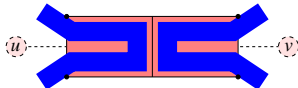
## Continuação da Prova Representação simplificada do widget



(a) Entra por  $u$ , percorre widget e sai por  $u$  (e bloqueia uso do widget por  $v$ )



(b) Entra por  $u$ , percorre metade do widget e sai por  $u$   
entra por  $v$ , percorre metade do widget e sai por  $v$



(c) Entra por  $v$ , percorre widget e sai por  $v$  (e bloqueia uso do widget por  $u$ )



Possíveis configurações do C.H. ao passar por widget da aresta  $\{u, v\}$

**Fato.** Se um caminho/ciclo entra em  $W_{u,v}$  pelo lado de  $u$ , então ele passa por metade ou todos os vértices de  $W_{u,v}$  e sai pelo mesmo lado de  $u$ .

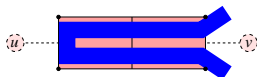
# Problema HAM-CYCLE

**Continuação da Prova.** Como widget  $W_{u,v}$  da aresta  $\{u, v\}$  é percorrido:

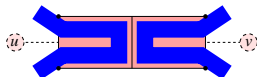
- Apenas  $u$  é selecionado no Vertex Cover: todos vértices do widget são percorridos pelo lado de  $u$ .



- Apenas  $v$  é selecionado no Vertex Cover: todos vértices do widget são percorridos pelo lado de  $v$ .



- $u$  e  $v$  são selecionados no Vertex Cover: Metade dos vértices do widget são percorridos pelo lado de  $u$  e outra metade pelo lado de  $v$ .



Obs.: Nos desenhos acima, os vértices  $u$  e  $v$  não pertencem ao grafo  $G'$ , apenas os que estão nos widgets e os  $k$  seletores.



# Problema HAM-CYCLE

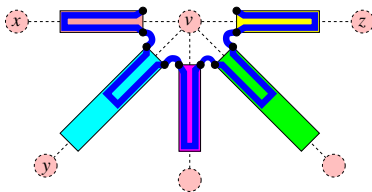
Continuação da Prova. Construção do grafo  $G'$

As pontas dos widgets de um mesmo vértice  $v \in G$  são ligados em série.

Por exemplo, se adjacentes de  $v \in G$  são  $x, y$  e  $z$  então:

Série de  $v$ :  $[v,x,1] \dots [v,x,6] \text{---} [v,y,1] \dots [v,y,6] \text{---} \dots \text{---} [v,z,1] \dots [v,z,6]$

Pontas da (série de)  $v$ :  $[v,x,1]$  e  $[v,z,6]$ .

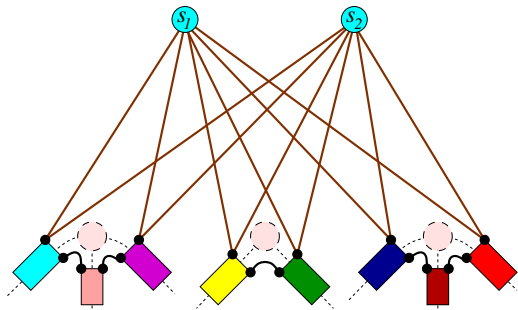


**Fato.** Ao entrar por uma ponta de  $v$ , o circuito percorre os widgets de todas arestas incidentes a  $v$  (para cada widget, passa por todos os vértices do widget ou metade deles) e sai pela outra ponta de  $v$ .

**Fato.** Se um caminho/ciclo passa pelos vértices de  $W_{u,v}$ , ele deve ter entrado por uma das pontas de  $u$  ou de  $v$  (ou ambos).

# Problema HAM-CYCLE

Continuação da Prova. Construção do grafo  $G'$

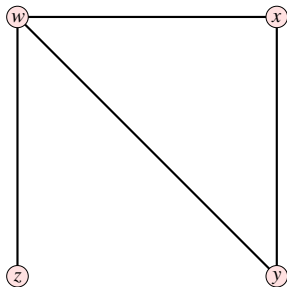


- São inseridos  $k$  seletores  $s_j$ .
- Cada seletor é ligado às duas pontas de cada **série**.

**Fato.** Em um **circuito hamiltoniano** de  $G'$ , as séries (uma para cada vértice de  $G$ ) se alternam com vértices **seletores**. Portanto, número de **séries** no circuito é igual ao número de **seletores**.

# Problema HAM-CYCLE

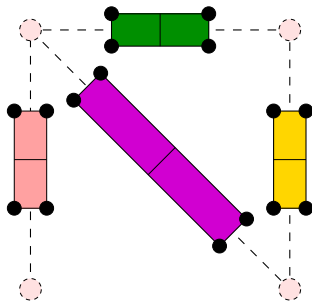
Continuação da Prova. Exemplo:



Grafo de entrada  $G$  para VERTEX-COVER

# Problema HAM-CYCLE

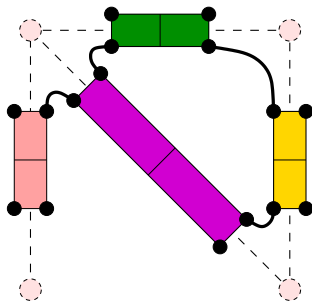
Continuação da Prova. Exemplo:



Inserção dos widgets  $W$ 's, uma para cada aresta do grafo original

# Problema HAM-CYCLE

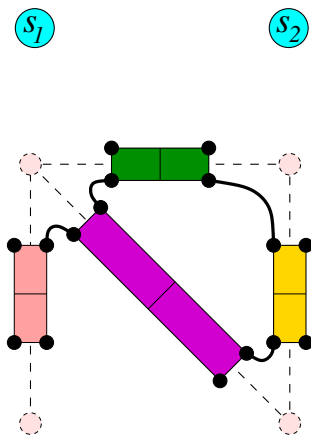
Continuação da Prova. Exemplo:



Ligação em série das pontas de widgets (voltadas aos mesmos vértices de  $G$ )

# Problema HAM-CYCLE

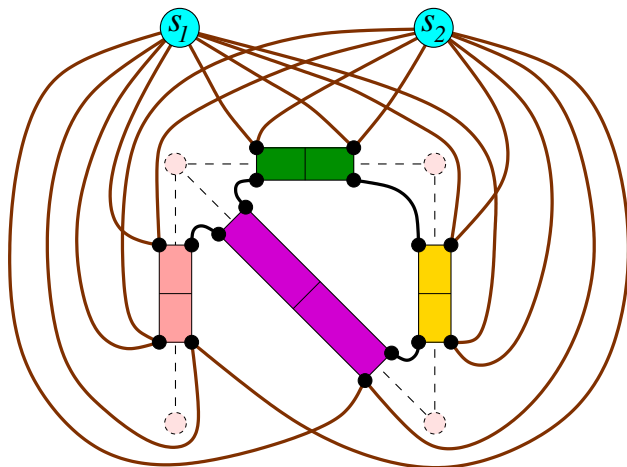
Continuação da Prova. Exemplo:



Inserção de  $k$  vértices seletores

# Problema HAM-CYCLE

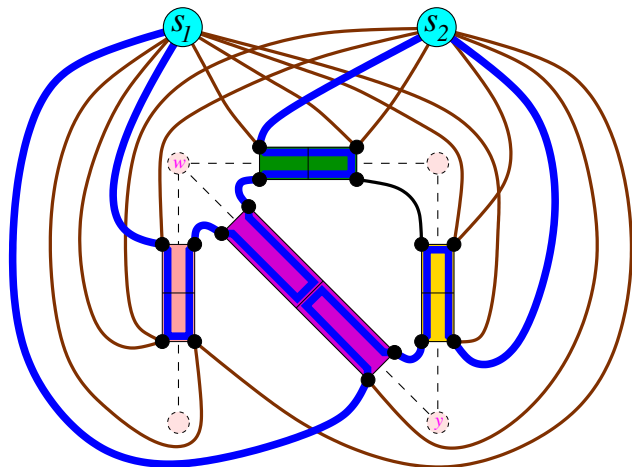
Continuação da Prova. Exemplo:



Ligação dos seletores para as pontas dos widgets e grafo final  $G'$

# Problema HAM-CYCLE

Continuação da Prova. Exemplo:



Cobertura por vértices  $\{w, y\}$  em  $G$  e Circuito hamiltoniano em  $G'$



# Problema HAM-CYCLE

Continuação da Prova.

**Fato.** A construção de  $G'$  é feita em tempo polinomial (exercício).

Vamos mostrar que

$\langle G, k \rangle \in \text{VERTEX-COVER}$  se e somente se  $\langle G' \rangle \in \text{HAM-CYCLE}$

( $\Rightarrow$ ): Seja  $\langle G, k \rangle \in \text{VERTEX-COVER}$  e  $C = \{v_1, \dots, v_k\}$  cobertura de  $G$

Seja  $A_v$  os vértices adjacentes de  $v \in V$  que pertencem a  $C$ .

Seja  $S_v$  a série de  $v$ , que para cada adjacente  $w$ , passa por *todos* vértices de  $W_{v,w}$ , se  $w \notin A_v$  ou por *metade dos* vértices de  $W_{v,w}$ , se  $w \in A_v$ .

Seja  $H$  o circuito formado pela seguinte sequência de vértices:

$$H = (s_1, S_{v_1}, s_2, S_{v_2}, \dots, s_k, S_{v_k}, s_1)$$

**Fato.**  $H$  é um ciclo.

**Fato.** Dada aresta  $\{u, v\} \in E$ , os vértices de  $W_{u,v}$  pertencem a  $H$ .

Como  $H$  contém todos os widgets e todos os seletores,  $H$  é hamiltoniano.

# Problema HAM-CYCLE

## Continuação da Prova.

Seja  $H$  um circuito hamiltoniano em  $G'$ .

**Fato.**  $H$  tem a forma:

$$H = (s_1, S_{v_1}, s_2, S_{v_2}, \dots, s_k, S_{v_k}, s_1)$$

Como  $H$  é hamilt., todos os vértices de widgets pertencem a alguma série.

Seja  $C = \{v_1, \dots, v_k\} \subseteq V$ . Cada  $S_{v_j}$  é uma série relativa a um vértice  $v_j$ .

**Fato.** Se  $\{u, v\} \in E$ , então  $u \in C$  ou  $v \in C$  (ou ambos).

(note que o widget  $W_{u,v}$  pertence a  $G'$  e foi percorrido por  $H$ ).

Portanto  $\langle G, k \rangle \in \text{VERTEX-COVER}$ .

Fim da prova do Lema: HAM-CYCLE  $\in$  NP-Difícil. □

## Teorema

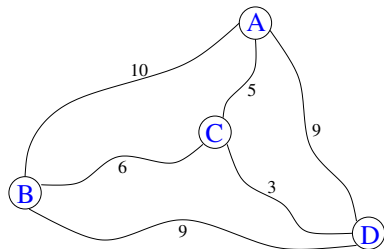
HAM-CYCLE  $\in$  NP-Completo.

# Problema TSP

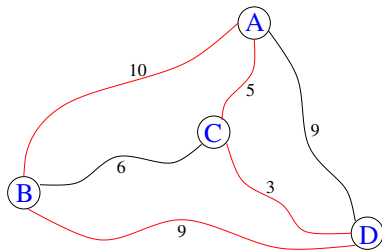
## Problema (de Otimização) do Caixeiro Viajante (TSP)

- **Entrada:**  
Grafo completo não-orientado com custos nas arestas
- **Objetivo:**  
Encontrar um *tour* (circuito hamiltoniano) de custo mínimo que visita cada vértice exatamente uma vez.

Entrada



Solução Ótima de custo 27



# Problema TSP

Definição como linguagem: inserimos parâmetro  $k$

$\text{TSP} = \{ \langle G, c, k \rangle : G = (V, E) \text{ é um grafo completo, } c \text{ é função nas arestas, } c : E \rightarrow \mathbb{Z}, k \in \mathbb{Z}, e G \text{ tem tour de custo máximo } k \}$ .

Lema

$\text{TSP} \in \text{NP}$ .

# Problema TSP

## Lema

TSP  $\in$  NP-Difícil.

### Prova.

Vamos fazer a redução **HAM-CYCLE**  $\leq_P$  **TSP**.

Dado entrada  $G = (V, E)$  para HAM-CYCLE, vamos construir entrada  $(G', c, k)$  para TSP tal que

$G$  é hamiltoniano  $\iff G'$  tem tour de custo no máximo  $k$ .

Construção de  $(G', c, k)$ : Seja  $G' = (V', E')$ ,  $c$  e  $k$  tal que:

$V' = V$  (o conjunto de vértices é o mesmo de  $G$ ),

$E' = \{\{u, v\} : u, v \in V \text{ e } u \neq v\}$

$c(u, v) = \begin{cases} 0 & \text{se } \{u, v\} \in E \\ 1 & \text{se } \{u, v\} \notin E \end{cases}$  para todo  $\{u, v\} \in V \times V$  e  $u \neq v$ ,

$k = 0$

# Problema TSP

## Continuação da Prova.

Fato.  $G$  é hamiltoniano se e somente se  $G'$  tem tour de custo 0.

( $\Rightarrow$ ): Seja  $C$  um ciclo hamiltoniano em  $G$ .

Então, todas as arestas de  $C$  pertencem a  $G'$  com custo 0.

Então,  $G'$  possui tour de custo 0.

( $\Leftarrow$ ): Seja  $C'$  um tour em  $G'$  de custo 0.

Então, todas as arestas de  $C'$  possuem custo 0.

Então, Todas arestas de  $C'$  estão em  $G$ .

Então,  $G$  é hamiltoniano.

Fim da prova do Lema:  $TSP \in NP$ -Difícil. □

## Teorema

$TSP \in NP$ -Completo.

# Problema SUBSET-SUM

## Problema SUBSET-SUM

- **Entrada:** Conjunto  $S$  de inteiros positivos e um inteiro  $t > 0$ .
- **Pergunta:** Existe subconjunto  $S' \subseteq S$  tal que  $t = \sum_{s \in S'} s$  ?

## Versão de Decisão/Linguagem:

$$\text{SUBSET-SUM} = \{ \langle S, t \rangle : \text{existe } S' \subseteq S \text{ tq. } t = \sum_{s \in S'} s \}$$

# Problema SUBSET-SUM

**Exemplo [CLRS'09]:** Considere a entrada  $\langle S, t \rangle$  tal que

$$S = \{1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993\}$$

$$t = 138457.$$

**Pergunta:**  $\langle S, t \rangle \in \text{SUBSET-SUM}$ ?

**Resposta:** Sim!

$S' = \{1, 2, 7, 98, 343, 686, 2409, 17206, 117705\}$  é uma solução.

Lema

SUBSET-SUM  $\in$  NP.

*Prova.* Exercício. □



# Problema SUBSET-SUM

## Lema

SUBSET-SUM  $\in$  NP-Difícil.

### Prova.

Vamos fazer a redução  $3\text{-CNF-SAT} \leq_P \text{SUBSET-SUM}$ .

Seja  $\phi(x_1, \dots, x_n)$  uma fórmula em 3-CNF.

Vamos construir uma entrada  $\langle S, t \rangle$  tal que

$\phi \in 3\text{-CNF-SAT}$  se e somente se  $\langle S, t \rangle \in \text{SUBSET-SUM}$

Vamos considerar, s.p.g., que  $\phi$  está na forma 3-CNF e

- não possui cláusulas com uma literal e sua negação
- não possui variáveis que não são utilizadas nas cláusulas

# Problema SUBSET-SUM

Continuação da Prova: Formato dos números de  $S$  e  $t$

Seja

- $x_1, \dots, x_n$  as variáveis de  $\phi$
- $C_1, \dots, C_k$  as cláusulas de  $\phi$

Os números de  $S$  e o inteiro  $t$  terão  $n + k$  dígitos (na base 10):

- Os  $n$  primeiros dígitos estão associados às variáveis
- Os  $k$  últimos dígitos estão associados às cláusulas

Se  $s$  é um inteiro tal que  $s \in S \cup \{t\}$ , vamos denotar por

- $s[x_j]$  o  $i$ -ésimo dígito de  $s$ , associado à variável  $x_j$ .
- $s[C_j]$  o  $n + j$ -ésimo dígito de  $s$ , associado à cláusula  $C_j$ .

**Exemplo:** Se  $\phi$  tem duas variáveis  $x_1$  e  $x_2$  e três cláusulas  $C_1$ ,  $C_2$  e  $C_3$ , então um número  $s \in S$  possui 5 dígitos na seguinte sequência:

	$x_1$	$x_2$	$C_1$	$C_2$	$C_3$
$s =$	$s[x_1]$	$s[x_2]$	$s[C_1]$	$s[C_2]$	$s[C_3]$

# Problema SUBSET-SUM

Continuação da Prova: Construção de  $v_i$  e  $v'_i$

O conjunto  $S$  tem  $2n + 2k$  inteiros:

$$S = \{v_1, v'_1, v_2, v'_2, \dots, v_n, v'_n, s_1, s'_1, s_2, s'_2, \dots, s_k, s'_k\}$$

onde

- $v_i$  e  $v'_i$  são inteiros associados à variável  $x_i$ , onde  $i = 1, \dots, n$
- $s_j$  e  $s'_j$  são inteiros associados à cláusula  $C_j$ , onde  $j = 1, \dots, k$

Os dígitos de  $v_i$  são dados por

$$v_i[x_{i'}] = \begin{cases} 1 & \text{se } i = i' \\ 0 & \text{se } i \neq i'. \end{cases} \quad v_i[C_j] = \begin{cases} 1 & \text{se } C_j \text{ contém a literal } x_i \\ 0 & \text{se } C_j \text{ caso contrário.} \end{cases}$$

Os dígitos de  $v'_i$  são dados por

$$v'_i[x_{i'}] = \begin{cases} 1 & \text{se } i = i' \\ 0 & \text{se } i \neq i'. \end{cases} \quad v'_i[C_j] = \begin{cases} 1 & \text{se } C_j \text{ contém a literal } \neg x_i \\ 0 & \text{se } C_j \text{ caso contrário.} \end{cases}$$

## Problema SUBSET-SUM

Continuação da Prova: Construção de  $s_j$ ,  $s'_j$  e  $t$ .

Os dígitos de  $s_j$  são dados por

$$s_j[x_{i'}] = 0 \text{ para todo } i' \qquad s_j[C_{j'}] = \begin{cases} 1 & \text{se } j = j' \\ 0 & \text{se } j \neq j'. \end{cases}$$

Os dígitos de  $s'_j$  são dados por

$$s'_j[x_{i'}] = 0 \text{ para todo } i' \qquad s'_j[C_{j'}] = \begin{cases} 2 & \text{se } j = j' \\ 0 & \text{se } j \neq j'. \end{cases}$$

Os dígitos de  $t$  são dados por

$$t[x_{i'}] = 1 \text{ para todo } i' \qquad t[C_{j'}] = 4 \text{ para todo } j'.$$

Fato

*A entrada  $\langle S, t \rangle$  é construída em tempo polinomial.*

# Problema SUBSET-SUM

Continuação da Prova: Exemplo [CLRS'09]: Construção de  $\langle S, t \rangle$

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

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$v_1$	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
$t$	1	1	1	4	4	4	4

# Problema SUBSET-SUM

Continuação da Prova: Exemplo [CLRS'09]: Construção de  $\langle S, t \rangle$

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

Fato

Todos os números em  $S$  são distintos.

Fato (Não ocorre 'vai um' nas somas)

A soma de todos dígitos na coluna  $x_i$  é no max. 2:

$$\sum_{s \in S} s[x_i] \leq 2.$$

A soma de todos dígitos na coluna  $C_j$  é no max. 6:

$$\sum_{s \in S} s[C_j] \leq 6.$$

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$v_1$	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

# Problema SUBSET-SUM

Continuação da Prova: ( $\Rightarrow$ ): Considere  $\langle \phi \rangle \in 3\text{-CNF-SAT}$

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

Atribuição Satisfazível		$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$x_1=0$	$v_1$	1	0	0	1	0	0	1
	$v'_1$	1	0	0	0	1	1	0
$x_2=0$	$v_2$	0	1	0	0	0	0	1
	$v'_2$	0	1	0	1	1	1	0
$x_3=1$	$v_3$	0	0	1	0	0	1	1
	$v'_3$	0	0	1	1	1	0	0
	Soma	1	1	1	1	2	3	1

Fato (Coluna das variáveis soma 1)

Seja  $x$  atribuição satisfazível para  $\phi$  e  $S^x$  o conjunto

$$S^x = \{v_i : x_i = 1, i \in [1, n]\} \cup \{v'_i : x_i = 0, i \in [1, n]\}$$

Então, para toda variável  $x_i$  temos

$$\sum_{s \in S^x} s[x_i] = 1$$

# Problema SUBSET-SUM

Continuação da Prova:

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

Atribuição Satisfazível		$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$x_1=0$	$v_1$	1	0	0	1	0	0	1
	$v'_1$	1	0	0	0	<b>1</b>	<b>1</b>	0
$x_2=0$	$v_2$	0	1	0	0	0	0	1
	$v'_2$	0	1	0	<b>1</b>	<b>1</b>	<b>1</b>	0
$x_3=1$	$v_3$	0	0	1	0	0	<b>1</b>	<b>1</b>
	$v'_3$	0	0	1	1	1	0	0
	Soma	1	1	1	<b>1</b>	<b>2</b>	<b>3</b>	<b>1</b>

Fato (Coluna de cláusula soma entre 1 e 3)

Seja  $x$  atribuição satisfazível para  $\phi$  e  $S^x$  o conjunto

$$S^x = \{v_i : x_i = 1, i \in [1, n]\} \cup \{v'_i : x_i = 0, i \in [1, n]\}$$

Então, para toda cláusula  $C$  temos

$$1 \leq \sum_{s \in S^x} s[C] \leq 3$$



# Problema SUBSET-SUM

Continuação da Prova: Complementar  $S^x$  para coluna de cláusula somar 4

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

Atribuição Satisfazível		$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$x_1=0$	$v_1$	1	0	0	1	0	0	1
	$v'_1$	1	0	0	0	<b>1</b>	<b>1</b>	0
$x_2=0$	$v_2$	0	1	0	0	0	0	1
	$v'_2$	0	1	0	<b>1</b>	<b>1</b>	<b>1</b>	0
$x_3=1$	$v_3$	0	0	1	0	0	<b>1</b>	<b>1</b>
	$v'_3$	0	0	1	1	1	0	0
Complementa coluna $C_1$ com 3 (=1+2)	$s_1$	0	0	0	<b>1</b>	0	0	0
	$s'_1$	0	0	0	<b>2</b>	0	0	0
Complementa coluna $C_2$ com 2	$s_2$	0	0	0	0	1	0	0
	$s'_2$	0	0	0	0	<b>2</b>	0	0
Complementa coluna $C_3$ com 1	$s_3$	0	0	0	0	0	<b>1</b>	0
	$s'_3$	0	0	0	0	0	2	0
Complementa coluna $C_4$ com 3 (=1+2)	$s_4$	0	0	0	0	0	0	<b>1</b>
	$s'_4$	0	0	0	0	0	0	<b>2</b>
	$t$	1	1	1	4	4	4	4

# Problema SUBSET-SUM

## Continuação da Prova:

### Fato

Seja  $x$  atribuição satisfazível para  $\phi$ . Então, existe  $S' \subseteq S$  tal que

$$\sum_{s \in S'} s[x_i] = 1 \quad \text{e} \quad \sum_{s \in S'} s[C_j] = 4$$

para toda variável  $x_i$  e cláusula  $C_j$ .

Para verificar este fato, considere

$$\begin{aligned} S^x &= \{v_i : x_i = 1, i \in [1, n]\} \cup \{v'_i : x_i = 0, i \in [1, n]\} \\ S^{C_j} &= \begin{cases} \{s_j, s'_j\} & \text{se } \sum_{s \in S^x} s[C_j] = 1 \\ \{s'_j\} & \text{se } \sum_{s \in S^x} s[C_j] = 2 \\ \{s_j\} & \text{se } \sum_{s \in S^x} s[C_j] = 3 \end{cases} \\ &\text{e} \\ S^C &= S^{C_1} \cup S^{C_2} \cup \dots \cup S^{C_k}. \end{aligned}$$

O fato vale usando  $S' = S^x \cup S^C$ . Portanto  $\langle S, t \rangle \in \text{SUBSET-SUM}$ .

# Problema SUBSET-SUM

Continuação da Prova: ( $\Leftarrow$ ): Considere  $\langle S, t \rangle \in \text{SUBSET-SUM}$

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

Seja  $S' \subseteq S$  tal que  $t = \sum_{s \in S'} s$ .

Fato ( $S'$  tem exatamente um em  $\{v_i, v'_i\}$  para cada  $x_i$ )

Se  $S' \subseteq S$  tal que  $t = \sum_{s \in S'} s$ , então exatamente um em  $\{v_i, v'_i\}$  é escolhido em  $S'$ , para cada  $x_i$ .

Fato (Soma dos  $v$  e  $v'$  em  $S'$  é  $\geq 1$  em cada cláusula)

Se  $S' \subseteq S$  tal que  $t = \sum_{s \in S'} s$ , então soma  $\sum_{i=1}^n \sum_{u \in \{v_i, v'_i\} \cap S'} u[C] \geq 1$  para cada  $C$ .

	$x_1$	$x_2$	$x_3$	$C_1$	$C_2$	$C_3$	$C_4$
$v_1$	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
$t$	<b>1</b>	<b>1</b>	<b>1</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>

# Problema SUBSET-SUM

Continuação da Prova:

Seja  $S' \subseteq S$  tal que  $t = \sum_{s \in S'} s$ .

Seja atribuição  $x = (x_1, \dots, x_n)$  tal que

$$x_i = \begin{cases} 1 & \text{se } v_i \in S' \\ 0 & \text{se } v'_i \in S'. \end{cases}$$

Fato

A atribuição  $x$  satisfaz  $\phi$ .

Para toda cláusula  $C_j$  temos que  $\sum_{r \in S'} r[C_j] = 4$ .

Como os elementos de  $s_j[C_j]$  e  $s'_j[C_j]$  somam no máximo 3, existe número em  $S'$  vindo das variáveis  $v_i$  ou  $v'_i$ , para algum  $i$ , com 1 na coluna  $C_j$ .

# Problema SUBSET-SUM

## Continuação da Prova:

Temos dois casos: (i)  $v_i \in S'$  e  $v_i[C_j]=1$  ou (ii)  $v'_i \in S'$  e  $v'_i[C_j]=1$

**(i)** Se  $v_i \in S'$  e  $v_i[C_j]=1$ , por construção, temos que a atribuição produzida com  $x_i = 1$  torna  $C_j$  verdadeira

**(ii)** Se  $v'_i \in S'$  e  $v'_i[C_j]=1$ , por construção, temos que a atribuição produzida com  $x_i = 0$  torna  $C_j$  verdadeira

Em ambos os casos,  $C_j$  fica verdadeira na atribuição de  $x$ . Como isso ocorre para todas as cláusulas, temos que  $x$  satisfaz  $\phi$  e portanto  $\langle \phi \rangle \in 3\text{-CNF-SAT}$ .

Fim da prova do Lema: SUBSET-SUM  $\in$  NP-Difícil. □

## Teorema

SUBSET-SUM  $\in$  NP-Completo.

# Problema CIRCUIT-SAT

## Lema

CIRCUIT-SAT  $\in$  NP-Difícil.

**Prova.** Apresenta detalhes técnicos fora do escopo do curso.  
Note a importância de se considerar o modelo computacional.

## Esboço:

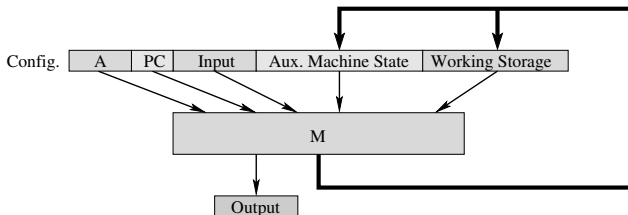
- 1 Seja  $\Pi$  um problema qualquer de NP.  
( $\Pi$  é verificável em tempo polinomial para resposta SIM):  
Existe algoritmo  $A$ , de tempo polinomial tal que  
Se  $x \in \Pi$ , existe certificado  $y$  tal que  $A(x, y) = 1$  e  
Se  $x \notin \Pi$ , temos  $A(x, y) = 0$ , para todo  $y$ .
- 2 Vamos reduzir  $\Pi$  a CIRCUIT-SAT.
- 3 **Ideia:** Simular a execução de  $A$  através um circuito combinatorial  
O circuito combinatorial é baseado no computador  
Em vez fazer a execução de  $A$  com o mesmo circuito,  
Repetimos o circuito do computador para cada iteração

# Problema CIRCUIT-SAT

## Continuação da Prova.

### Computador:

- Acesso a memória, dispositivos de entrada e de saída
- Apresenta circuito para realizar suas operações



A cada passo de execução do algoritmo  $A$ :

- a máquina recebe e atualiza configurações (memória) do computador (**Configuração**: todas as memórias utilizadas, como para armazenar  $A$ ,  $PC$ , dados de entrada, memória de trabalho e saída)
- Atualiza a configuração atual para ser usada no próximo passo
- Reaproveita o mesmo circuito  $M$  da máquina

# Problema CIRCUIT-SAT

## Continuação da Prova.

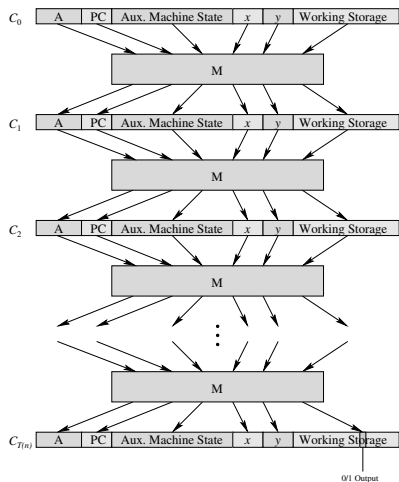
### A execução de $A(x, y)$ :

- Fará no máximo  $T(n) \in O(n^k)$  passos, onde  $n = |x|$  e  $k$  é constante. Vamos supor que  $A$  sempre gasta exatamente  $T(n)$  passos (se terminar antes, faz passos “dummy”)
- $y$  tem tamanho polinomial ( $|y| = O(n^{k'})$ , para algum  $k'$  constante.)
- Em vez de aproveitar a mesma máquina  $M$  fazemos  $T(n)$  cópias de  $M$ , que em vez de atualizar a configuração, a gera como entrada para a próxima cópia de  $M$ .
- Alguns bits (fios) são fixados para definir configuração inicial (algoritmo  $A$ , PC inicial,  $x$ , etc.)
- Fios de entrada são definidos apenas para os que definem  $y$ .
- No último passo, a saída é apenas um bit (fio) do circuito.



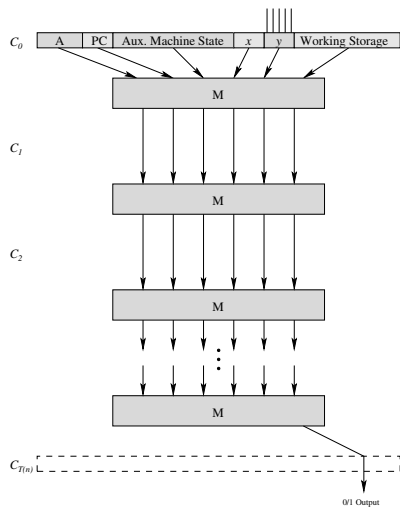
# Problema CIRCUIT-SAT

Continuação da Prova.  $T(n)$  cópias de  $M$  para simular execução de  $A$



# Problema CIRCUIT-SAT

Continuação da Prova. Entrada, saída e ligação direta das máquinas.



# Problema CIRCUIT-SAT

## Continuação da Prova.

Seja  $C$  o circuito final e  $C(y')$  a saída de  $C$  para entrada  $y'$ .

### Fato

*Alguns fios de  $C$  começam com valores de 0/1 atribuídos (codificação de  $A$ ,  $PC$  e  $x$ ).*

Note que para um fio  $f$  receber 1, basta pegar um fio qualquer  $q$  e fazer o fio  $f$  receber  $q \vee \neg q$ . Para  $f$  receber 0, basta  $f$  receber  $q \wedge \neg q$ .

### Fato

*Foram removidos todos os fios da configuração final, exceto pelo fio de saída.*

Para isto, note que o grafo associado é acíclico. Basta remover as portas e fios que não são necessários para o fio de saída (em tempo linear).

# Problema CIRCUIT-SAT

## Continuação da Prova.

### Fato

*O número de fios/bits em cada configuração é de tamanho polinomial.*

Para isto, note que:

- Codificações de  $A$  e  $PC$  usam quantidade constante de bits (objetos acessados por  $A$  podem ter tamanho polinomial).
- $y$  tem tamanho polinomial em relação a  $x$ .
- Memória de trabalho, em qualquer momento, é de tamanho polinomial (pois  $A$  é de tempo polinomial).

# Problema CIRCUIT-SAT

## Continuação da Prova.

### Fato

*A construção de  $C$  é feita em tempo polinomial.*

- A máquina  $M$  tem comportamento específico, mas de tamanho polinomial, uma vez que  $M$  depende da configuração de entrada (visto ser polinomial).
- Foram feitas  $T(n)$  cópias de  $M$  (cada uma de tamanho polinomial)
- Se a maior configuração+máquina de um passo tem tamanho  $Q(n)$ , a construção de  $C$  gasta  $O(T(n) \cdot Q(n))$ .

### Fato

*$A(x, y) = C(y)$ , para todo  $y$ .*

Note que o circuito  $C$  simplesmente simula a execução de  $A$ .

# Problema CIRCUIT-SAT

## Continuação da Prova.

### Fato

Se  $C$  é o circuito construído para  $x$ , então

$x \in \Pi$  se e somente se  $C \in \text{CIRCUIT-SAT}$ .

$\Rightarrow$  Seja  $x \in \Pi$ .

Então, existe certificado  $y$  tal que  $A(x, y) = 1$ .

Como  $C(y) = A(x, y) = 1$ , temos que  $C$  é satisfazível.

$\Leftarrow$  Suponha  $C$  satisfazível.

Então, existe entrada  $y$  tal que  $C(y) = 1$ .

Como  $A(x, y) = C(y) = 1$ , temos que

$A$  verifica  $x$  pelo certificado  $y$ .

Fim da prova do Lema:  $\text{CIRCUIT-SAT} \in \text{NP-Difícil}$ . □

### Teorema

$\text{CIRCUIT-SAT} \in \text{NP-Completo}$

## Outros aspectos de complexidade computacional

No próximo conjunto de slides, apenas citaremos, de maneira superficial, alguns resultados diferentes na área de complexidade. O leitor deve buscar mais informação sobre os tópicos de interesse.

## Decisão $\times$ Otimização

Nem sempre a versão de decisão tem a mesma complexidade do de otimização.

### Definição (fórmula em 2-SAT)

Uma fórmula  $\phi$  está em **2-SAT** se

- $\phi$  está na forma CNF e
- cada cláusula possui no máximo 2 literais distintas.

### Problema (2-CNF-SAT)

Dada fórmula  $\phi(x_1, \dots, x_n)$  em 2-SAT, decidir se existe atribuição para as variáveis de maneira a tornar  $\phi$  verdadeira.

### Lema

O problema 2-SAT pode ser resolvido em tempo polinomial.

Prova. Exercício 34.4-7 de [CLRS'09],

□



# Decisão × Otimização

## Problema (MAX-2-SAT)

*Dada fórmula  $\phi(x_1, \dots, x_n)$  em 2-SAT e inteiro  $k$ , decidir se existe atribuição para as variáveis de maneira a satisfazer pelo menos  $k$  cláusulas de  $\phi$ .*

## Teorema (Garey, Johnson, and Stockmeyer'76)

*O problema MAX-2-SAT é NP-Completo.*

# NP-Intermediário

Vamos supor que  $P \neq NP$ .

Neste caso, há problemas “entre”  $P$  e NP-Completo: NP-Intermediário.

## Teorema (Ladner'75)

*Se  $P \neq NP$ , existem problemas de NP que não estão nem em  $P$  nem em NP-Completo.*

Neste caso, acredita-se que os seguintes problemas estão em NP-Intermediário:

- Fatoração de inteiros e
- Logaritmo discreto
- Isomorfismo em grafos

# Complexidade de Espaço e Não-Determinismo

Originalmente, NP foi definido através dos algoritmos *Nondeterministic Polynomial*.

## Definição

*Um algoritmo não-determinístico é como um algoritmo determinístico, mas que também pode fazer, a cada passo, escolhas entre várias alternativas de maneira não-determinística.*

## Definição

*O espaço (memória) utilizado por um algoritmo determinístico, é o número de células (bits) distintas acessado pelo algoritmo.*

## Definição

*O espaço (memória) utilizado por um algoritmo não-determinístico, é o número de células (bits) distintas acessado por um ramo de execução mais curto pelo algoritmo.*

# Complexidade de Espaço e Não-Determinismo

A Classe NP é a classe dos problemas que podem ser decididos por um algoritmo não-determinístico em tempo polinomial.

Em termos da complexidade de tempo, a inclusão do não-determinismo parece ter elevado em muito a dificuldade dos problemas. E quanto a complexidade de espaço?

O seguinte teorema diz que o conjunto de problemas decididos por algoritmos determinísticos e não-determinísticos com espaço polinomial é o mesmo.

Teorema (Savitch'70)

$$\text{NPSPACE} = \text{PSPACE}.$$

# Problemas Indecidíveis

A maioria dos problemas computacionais pode ser resolvida, possivelmente com um grande esforço computacional. Mas existem alguns que são indecidíveis, independente do tempo de execução.

## Definição

*O problema da parada, consiste em: Dado algoritmo  $A$  e entrada  $x$ , decidir se  $A(x)$  para ou entra em loop.*

## Teorema (Turing'36)

*O problema da parada é indecidível.*

## Problemas Indecidíveis

*Prova.* [Esboço] Suponha que exista função  $H(A, x)$  que devolve **1** se  $A(x)$  **para** e **0** se  $A(x)$  entra em **loop**.

Considere a seguinte função  $F(A)$ :

*Função  $F(A)$*

1. Se  $H(A, A) = 1$  entre em **loop**;
2. senão, **pare**.

O que acontece com  $F(F)$ ?

(Sabemos que  $F(F)$  só pode entrar em loop, ou parar.)

**Caso 1:** Suponha que  $F(F)$  entra em loop.

Neste caso, o algoritmo entrou na condição da linha 1.

Para isso, a condição  $H(F, F) = 1$  foi verdadeira.

**Portanto  $F(F)$  para.** O que nos leva a uma contradição.

**Caso 2:** Suponha que  $F(F)$  para.

Neste caso, o algoritmo executou a linha 2.

Para isso, deve ter ocorrido  $H(F, F) = 0$ .

**Portanto  $F(F)$  entrou em loop.** O que nos leva a uma contradição.  $\square$