

# ACH2043 INTRODUÇÃO À TEORIA DA COMPUTAÇÃO

## Aula 25

### Cap 7 – Complexidade de Tempo

Profa. Ariane Machado Lima  
ariane.machado@usp.br

# Decidibilidade e complexidade

- Um problema pode ser decidível, mas na **prática** ser “insolúvel” (talvez por enquanto) devido à demanda de tempo e/ou de memória
- Analisar complexidade nos ajuda a identificar
  - Quais problemas são **tratáveis**
  - Estimativa de tempo (ou memória) necessários
- Qual é a complexidade dos modelos que vimos até agora (AF, AP, ALL, MT)

# Medindo complexidade de tempo

- Complexidade de tempo relacionada com o **número de passos** necessários para um algoritmo dar uma resposta
- Número de passos depende de parâmetros específicos do problema (ex: número de nós de um grafo)
- Ex: tamanho da cadeia de entrada (natural no caso de linguagens) –  $n$

# Medindo complexidade de tempo

- Análise de **pior caso**: maior tempo (número de passos) considerando todas as possíveis entradas de comprimento  $n$
- Análise de **caso médio**: média dos tempos considerando todas as entradas de tamanho  $n$
- **Definição**: Seja  $M$  uma Máquina de Turing determinística que **pára sobre todas** as entradas. O **tempo de execução** ou **complexidade de tempo** de  $M$  é a função  $f:\mathbb{N}\rightarrow\mathbb{N}$ , onde  $f(n)$  é o número **máximo** de passos que  $M$  usa sobre entradas de comprimento  $n$

# Medindo complexidade de tempo

- As seguintes frases são equivalentes:
  - $f(n)$  é o tempo de execução de  $M$
  - $M$  roda em tempo  $f(n)$
  - $M$  é uma máquina de Turing de tempo  $f(n)$
  - $M$  tem complexidade de tempo  $f(n)$

# Como calcular f?

- Calcular o número EXATO de passos é complicado
- Alternativa: estimar um valor aproximado
- Como? Análise assintótica (comportamento nos limites)
- Ex:  $f(n) = 5n^8 + 6n^3 + 8$

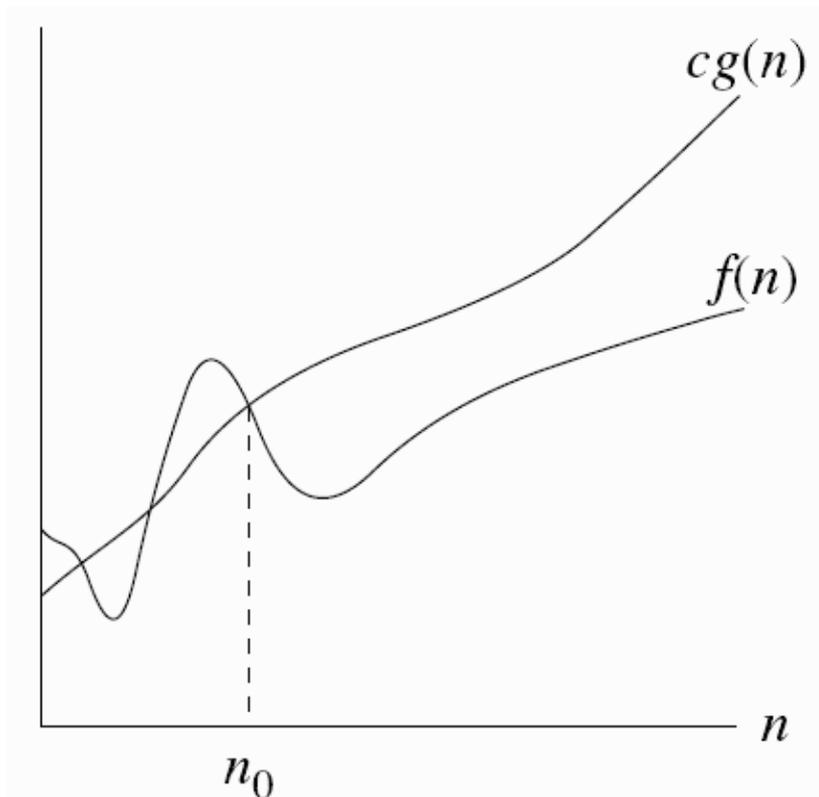
o que realmente está “ditando o crescimento” de  $f(n)$ ?  $n^8$

$$O(f(n)) = n^8$$

# Notação assintótica (O-grande)

- Sejam  $f$  e  $g$  funções  $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$ . Dizemos que  $f(n) = O(g(n))$  se existem inteiros positivos  $c$  e  $n_0$  tais que, para todo  $n \geq n_0$

$$f(n) \leq cg(n)$$



- $g(n)$  é um **limitante superior assintótico** para  $f(n)$

# Analizando algoritmos

- Uma MT para a linguagem  $A = \{0^k 1^k \mid k \geq 0\}$ :

M1 = “Sobre a cadeia de entrada  $w$ :

$$2n = O(n)$$

1. Faça uma varredura na fita e *rejeite* se for encontrado algum 0 à direita de algum 1

2. Repita se existem ambos, 0s e 1s, na fita:

$$n/2 * O(n) = O(n^2)$$

3. Faça uma varredura na fita, cortando um único 0 e um único 1

4. Se ainda restarem 0s após todos os 1s tiverem sido cortados ou se ainda restarem 1s após todos os 0s terem sido cortados, *rejeite*. Caso contrário, *aceite*.”

$$\text{Total: } O(n) + O(n^2) + O(n) = O(n^2)$$

$$2n = O(n)$$

# Classe de complexidade

- **Definição:** Seja uma função  $t:\mathbb{N}\rightarrow\mathbb{R}^+$  (tamanho de entrada  $\rightarrow$  tempo, ex:  $t(n) = n^2$ ). A **classe de complexidade de tempo  $\text{TIME}(t(n))$**  é a coleção de todas as linguagens decidíveis por uma máquina de Turing de tempo  $O(t(n))$ .
- $A = \{0^k1^k \mid k \geq 0\} \in \text{TIME}(n^2)$

# Classe de complexidade

- $A = \{0^k1^k \mid k \geq 0\} \in \text{TIME}(n^2)$
- Pergunta: será que  $A \in \text{TIME}(t(n))$  onde  $t(n) = o(n^2)$ , ou seja,  $A$  pode ser decidida por uma MT que rode em menos tempo (mais rápida)?
- Sim,  $A \in \text{TIME}(n \log n)$
- Pode ser ainda mais rápido?
  - Não usando uma MT de fita única
  - Na verdade, só linguagens regulares podem ser reconhecidas em tempo  $o(n \log n)$  em uma MT de fita única
- $A \in \text{TIME}(n)$  se a máquina tiver duas fitas

# Classe de complexidade

- Dá para diminuir ainda mais a complexidade?
- Não, pois só para ler a cadeia demora  $O(n)$

# Classe de complexidade - Conclusões

- A classe de complexidade de uma linguagem depende do modelo de computação escolhido
- Diferentemente, em computabilidade (ser decidível, reconhecível ou não) o modelo não importa
  - Tese de Church-Turing implica que todos os modelos razoáveis de computáveis são equivalentes
- Pergunta: para classificar um problema segundo sua complexidade, que modelos escolhemos?
- Requisitos de tempo não diferem  **muito**  para os modelos  **determinísticos**  típicos, e portanto a escolha não terá muito impacto

# Relacionamentos de Complexidade entre modelos

- Como a escolha do modelo computacional pode afetar a complexidade de tempo de um problema?
- Consideramos 3 modelos
  - Máquina de Turing fita-única (determinística)
  - Máquina de Turing multifita (determinística)
  - Máquina de Turing não-determinística (fita-única)

# Relacionamentos de Complexidade entre modelos

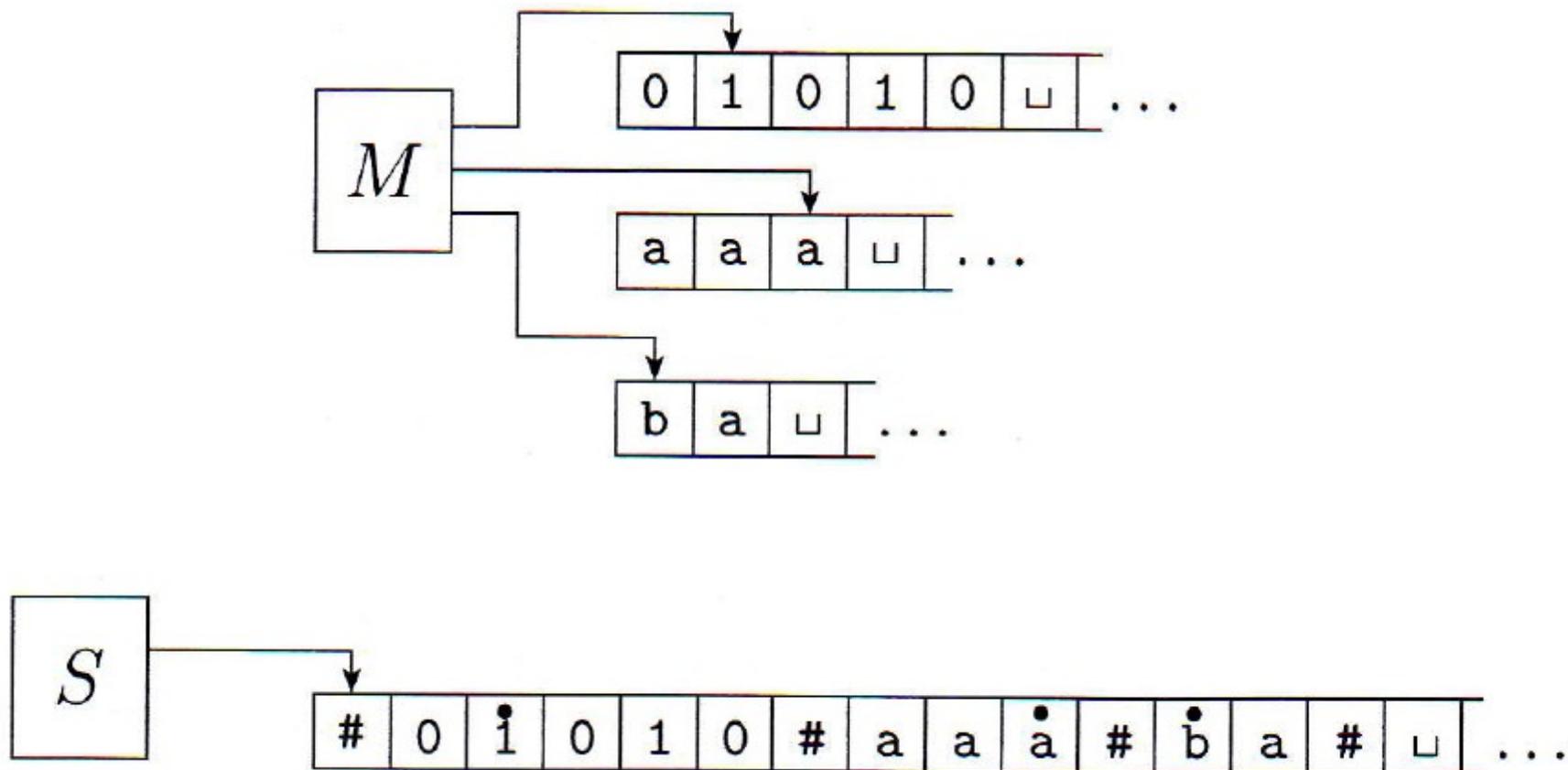
- **Teorema:** Seja  $t(n)$  uma função, onde  $t(n) \geq n$ .  
Toda máquina de Turing **multifita de tempo  $t(n)$**   
tem uma máquina de Turing **fita-única**  
**equivalente de tempo  $O(t^2(n))$**

# Relacionamentos de Complexidade entre modelos

- **Teorema:** Seja  $t(n)$  uma função, onde  $t(n) \geq n$ . Toda máquina de Turing multifita de tempo  $t(n)$  tem uma máquina de Turing fita-única equivalente de tempo  $O(t^2(n))$
- **Ideia da Prova:** Analisar a simulação de uma MT multifita  $M$  por uma MT fita-única  $S$

# MT multifita $M$ e MT fita-única $S$

## IDEIA DA PROVA



# MT multifita M e MT fita-única S

## IDEIA DA PROVA

- Cada passo em M, exige em S
  - Uma varredura na fita identificando as posições das cabeças de fita
  - Outra varredura na fita para atualizar conteúdo e posições de cabeça
  - Se uma cabeça se move para a direita “além do limite” (para uma posição ainda não usada na fita virtual), todo o conteúdo da fita a partir daquela posição deve ser deslocado para a direita.

# Tempo polinomial e exponencial

- Ex:
  - Máquina de tempo  $n^3$  (tempo polinomial)
  - Máquina de tempo  $2^n$  (tempo exponencial)
  - $n = 1000$ :
  - $n^3 = 1$  bilhão
  - $2^n =$  número maior que o número de átomos do universo (aproximadamente 300 dígitos)

# Tempo polinomial

- Todos os modelos computacionais **determinísticos** são **polinomialmente equivalentes** (um simula o outro com uma diferença de tempo polinomial)
- Problemas tratáveis na prática são polinomiais
- Em complexidade, muitas vezes só se quer saber se um problema é polinomial ou exponencial
  - Não importante o grau do polinômio
  - Reduzir complexidade de exponencial para polinomial muitas vezes permite atingir uma complexidade razoável para fins práticos

# Tempo polinomial

- **Definição:** **P** é a classe de linguagens que são decidíveis em tempo **polinomial** sobre uma máquina de Turing determinística de uma fita. Em outras palavras,

$$P = \bigcup_k \text{TIME}(n^k).$$

- **Importância:**
  - P é invariante para todos os modelos de computação polinomialmente equivalentes à MT determinística de fita-única
  - P corresponde aproximadamente à classe de problemas que são realisticamente solúveis em um computador.

# Analizando se um problema está em P

- Descreveremos ALGORITMOS (não considerando um modelo computacional específico)
  - Sem descrever fitas e movimentações de cabeça
- Um algoritmo é polinomial quando:
  - Cada estágio roda em tempo polinomial
  - Cada estágio é executado um número polinomial de vezes
  - Codificação e decodificação da entrada ocorrem em tempo polinomial

# Exemplos de problemas em P

# CAM pertence a P ?

- $CAM = \{ \langle G, s, t \rangle \mid G \text{ é um grafo direcionado que tem um caminho direcionado do nó } s \text{ para o nó } t \}$
- Codificações possíveis de  $G$ :
  - lista de nós e arestas
  - Matriz de adjacência
- Uma solução força-bruta (verifica todas as possibilidades):
  - Verifica todos os caminhos em potencial de comprimento no máximo  $m$  ( $m = \text{número de nós}$ )
  - Número de caminhos em potencial:

# CAM pertence a P ?

- $CAM = \{ \langle G, s, t \rangle \mid G \text{ é um grafo direcionado que tem um caminho direcionado do nó } s \text{ para o nó } t \}$
- Codificações possíveis de  $G$ :
  - lista de nós e arestas
  - Matriz de adjacência
- Uma solução força-bruta (verifica todas as possibilidades):
  - Verifica todos os caminhos em potencial de comprimento no máximo  $m$  ( $m = \text{número de nós}$ )
  - Número de caminhos em potencial:  
**aproximadamente  $m^m$  (exponencial!)**

# CAM pertence a P

- Alternativa: busca em grafo (ex: busca em largura):
  - Marcamos sucessivamente todos os nós em  $G$  que são atingíveis a partir de  $s$  por caminhos direcionados de comprimento 1, depois 2, depois 3, ... até  $m$ .
  - Se existe um caminho direcionado de  $s$  a  $t$ ,  $t$  ficará marcado

# Teorema: CAM pertence a P

- Prova: O algoritmo M é polinomial:

M = “Sobre a entrada  $\langle G,s,t \rangle$  onde  $G$  é um grafo direcionado com nós  $s$  e  $t$ :

1. Marque o nó  $s$ .
2. Repita até que nenhum nó adicional seja marcado:
  3. Faça uma varredura em todas as arestas de  $G$ . Se uma aresta  $(a,b)$  for encontrada indo de um nó marcado  $a$  para um nó não marcado  $b$ , marque o nó  $b$
4. Se  $t$  estiver marcado, *aceite*. Caso contrário, *rejeite*.

- 

—

—

# Teorema: CAM pertence a P

- Prova: O algoritmo M é polinomial:

M = “Sobre a entrada  $\langle G, s, t \rangle$  onde  $G$  é um grafo direcionado com nós  $s$  e  $t$ :

1. Marque o nó  $s$ .

2. Repita até que nenhum nó adicional seja marcado:

3. Faça uma varredura em todas as arestas de  $G$ . Se uma aresta  $(a, b)$  for encontrada indo de um nó marcado  $a$  para um nó não marcado  $b$ , marque o nó  $b$

4. Se  $t$  estiver marcado, *aceite*. Caso contrário, *rejeite*.

- Quantas vezes cada estágio é executado?

- 1 e 4 são executados uma vez

- 3 é executado no máximo  $m$  vezes (se cada vez marcar somente um nó)

# Teorema: CAM pertence a P

- Prova: O algoritmo M é polinomial:

M = “Sobre a entrada  $\langle G, s, t \rangle$  onde  $G$  é um grafo direcionado com nós  $s$  e  $t$ :

1. Marque o nó  $s$ .

2. Repita até que nenhum nó adicional seja marcado:

3. Faça uma varredura em todas as arestas de  $G$ . Se uma aresta  $(a, b)$  for encontrada indo de um nó marcado  $a$  para um nó não marcado  $b$ , marque o nó  $b$

4. Se  $t$  estiver marcado, *aceite*. Caso contrário, *rejeite*.

- Complexidade de cada estágio?

—

# Teorema: CAM pertence a P

- Prova: O algoritmo M é polinomial:

M = “Sobre a entrada  $\langle G, s, t \rangle$  onde  $G$  é um grafo direcionado com nós  $s$  e  $t$ :

1. Marque o nó  $s$ .

2. Repita até que nenhum nó adicional seja marcado:

3. Faça uma varredura em todas as arestas de  $G$ . Se uma aresta  $(a, b)$  for encontrada indo de um nó marcado  $a$  para um nó não marcado  $b$ , marque o nó  $b$

4. Se  $t$  estiver marcado, *aceite*. Caso contrário, *rejeite*.

- Complexidade de cada estágio?

- Polinomial

# Teorema: CAM pertence a P

- Prova: O algoritmo M é polinomial:

M = “Sobre a entrada  $\langle G, s, t \rangle$  onde  $G$  é um grafo direcionado com nós  $s$  e  $t$ :

1. Marque o nó  $s$ .

2. Repita até que nenhum nó adicional seja marcado:

3. Faça uma varredura em todas as arestas de  $G$ . Se uma aresta  $(a, b)$  for encontrada indo de um nó marcado  $a$  para um nó não marcado  $b$ , marque o nó  $b$

4. Se  $t$  estiver marcado, *aceite*. Caso contrário, *rejeite*.

- Codificação / decodificação de  $\langle G, s, t \rangle$

- Polinomial

# Teorema: CAM pertence a P

- Prova: O algoritmo M é polinomial:

M = “Sobre a entrada  $\langle G, s, t \rangle$  onde  $G$  é um grafo direcionado com nós  $s$  e  $t$ :

1. Marque o nó  $s$ .
2. Repita até que nenhum nó adicional seja marcado:
  3. Faça uma varredura em todas as arestas de  $G$ . Se uma aresta  $(a, b)$  for encontrada indo de um nó marcado  $a$  para um nó não marcado  $b$ , marque o nó  $b$
4. Se  $t$  estiver marcado, *aceite*. Caso contrário, *rejeite*.

Codificação / decodificação de  $\langle G, s, t \rangle$

- Polinomial

LOGO CAM  $\in$  P

# Toda Linguagem Livre de Contexto pertence a P

- Testar todas as possíveis derivações: exponencial
- Algoritmo **CYK** (para gramáticas na **forma normal de Chomsky!**)
- **Programação dinâmica**: uso de soluções de subproblemas menores para resolver subproblemas maiores (até chegar à solução do problema original)
- Tabela  $n \times n$ :
  - $i \leq j$  : variáveis que geram a subcadeia  $w_i \dots w_j$
  - Tratam-se tamanhos crescentes (começando de 1)

# Toda Linguagem Livre de Contexto pertence a P

$D =$  “Sobre a entrada  $w = w_1 \cdots w_n$ :

1. Se  $w = \varepsilon$  e  $S \rightarrow \varepsilon$  for uma regra, *aceite*.    [[ trata o caso  $w = \varepsilon$  ]]
2. Para  $i = 1$  até  $n$ :    [[ examina cada subcadeia de comprimento 1 ]]
3.    Para cada variável  $A$ :
4.        Teste se  $A \rightarrow b$  é uma regra, onde  $b = w_i$ .
5.        Se for, coloque  $A$  em  $tabela(i, i)$ .
6. Para  $l = 2$  até  $n$ :    [[  $l$  é o comprimento da subcadeia ]]
7.    Para  $i = 1$  até  $n - l + 1$ :    [[  $i$  é a posição inicial da subcadeia ]]
8.        Faça  $j = i + l - 1$ ,    [[  $j$  é a posição final da subcadeia ]]
9.        Para  $k = i$  até  $j - 1$ :    [[  $k$  é a posição em que ocorre a divisão ]]
10.        Para cada regra  $A \rightarrow BC$ :
11.            Se  $tabela(i, k)$  contém  $B$  e  $tabela(k + 1, j)$  contém  $C$ ,  
              ponha  $A$  em  $tabela(i, j)$ .
12. Se  $S$  estiver em  $tabela(1, n)$ , *aceite*. Caso contrário, *rejeite*.”

# A classe NP

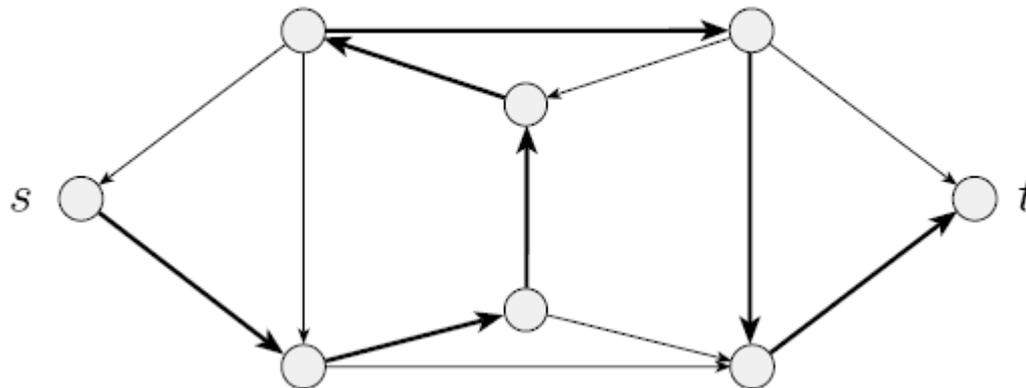
- Um problema está na classe P se existe uma MT determinística que o resolva e que rode em tempo polinomial.
- E se um algoritmo de tempo polinomial não é conhecido?
  - 
  - 
  - 
  - 
  -

# A classe NP

- Um problema está na classe P se existe uma MT determinística que o resolva e que rode em tempo polinomial.
- E se um algoritmo de tempo polinomial não é conhecido?
  - Ou o problema é intrinsecamente difícil
  - Ou um algoritmo polinomial não é conhecido
- Há vários problemas nesta situação
- Não sabemos distinguir entre os dois casos
- Para certos problemas, embora para DECIDIR conheça-se apenas algoritmos exponenciais, VERIFICAR se uma solução candidata é mesmo uma solução pode ser feito em tempo polinomial

# Ex: Caminho Hamiltoniano (em um grafo direcionado)

- Dado um grafo direcionado  $G$ , um caminho Hamiltoniano do nó  $s$  ao nó  $t$  é um caminho que, partindo do nó  $s$ , chega ao nó  $t$  após passar por todos os nós do grafo exatamente uma vez.



# Ex: Caminho Hamiltoniano (em um grafo direcionado)

- CAMHAM = {  $\langle G, s, t \rangle$ :  $G$  é um grafo direcionado com um caminho Hamiltoniano do nó  $s$  ao nó  $t$  }
- Aplicações?

# Ex: Caminho Hamiltoniano (em um grafo direcionado)

- CAMHAM = {  $\langle G, s, t \rangle$ :  $G$  é um grafo direcionado com um caminho Hamiltoniano do nó  $s$  ao nó  $t$  }
- Aplicações?
  - Rotas de distribuição
  - Itinerários de ônibus
  - Etc.

# Ex: Caminho Hamiltoniano (em um grafo direcionado)

- Algoritmo exponencial que decide CAMHAM:  
Gere todos os caminhos possíveis  
Verifique se um deles é:
  - de  $s$  a  $t$
  - passa por cada nó (todos eles) apenas uma vez

# Ex: Caminho Hamiltoniano (em um grafo direcionado)

- Algoritmo exponencial que decide CAMHAM:  
Gere todos os caminhos possíveis  
Verifique se um deles é:
  - de  $s$  a  $t$
  - passa por cada nó (todos eles) apenas uma vez
- Não se conhece uma solução polinomial
- NINGUÉM SABE SE ESSA SOLUÇÃO EXISTE
- Mas dado um caminho, é possível VERIFICAR se ele é hamiltoniano em tempo polinomial.

# Ex: Caminho Hamiltoniano (em um grafo direcionado)

- Para alguns problemas, nem para a VERIFICAÇÃO se conhece um algoritmo polinomial
- Ex: o complemento de CAMHAM (grafos que não possuem nenhum caminho hamiltoniano entre  $s$  e  $t$ )

# Verificador

- Um **verificador** para uma linguagem  $A$  é um algoritmo  $V$ , sendo

$$A = \{ w \mid V \text{ aceita } \langle w, c \rangle \text{ para alguma cadeia } c \}$$

- O  $c$  é o **certificado** ou **prova** de que  $w$  pertence a  $A$  (no exemplo de CAMHAM,  $c$  é um caminho)
- Tempo do verificador medido em termos apenas do comprimento de  $w$  (não de  $c$ )
- Uma linguagem  $A$  é **polinomialmente verificável** se ela tem um verificador de tempo polinomial
- Para verificadores polinomiais,  $c$  deve ter comprimento polinomial (no tamanho de  $w$ )

# Verificador - Exemplo

- O que seria um certificado para CAMHAM?
  - Um caminho hamiltoniano de  $s$  a  $t$
- Como seria o verificador?
  - Verifica se não há repetição de nós no caminho
  - Verifica se começa com  $s$  e termina com  $t$
  - Verifica se entre dois nós há uma aresta no grafo

# A classe NP

- NP é a classe de todas as linguagens **polinomialmente verificáveis**
- Um problema NP também pode ser P?

—

—

—

—

—

# A classe NP

- NP é a classe de todas as linguagens **polinomialmente verificáveis**
- Um problema NP também pode ser P?
  - Sim!
  - Ex: COMPOSTOS =  $\{ x \mid x = pq, \text{ para inteiros } p, q > 1 \}$
  - Certificado?
  - 
  -

# A classe NP

- NP é a classe de todas as linguagens **polinomialmente verificáveis**
- Um problema NP também pode ser P?
  - Sim!
  - Ex: COMPOSTOS =  $\{ x \mid x = pq, \text{ para inteiros } p, q > 1 \}$
  - Certificado? Um dos divisores
  - Há um algoritmo polinomial que o resolve (está em P)
  - Há um algoritmo polinomial que o verifique (está em NP)

# A classe NP

- NP vem de tempo **polinomial não-determinístico**
- Teorema: Uma linguagem está em **NP** se e somente se ela é decidida por alguma **máquina de Turing não-determinística (MTN)** de tempo **polinomial**
  - Lembrando que o tempo de uma MTN é o tempo do ramo mais longo...
  - O certificado é uma solução aceita em um dos ramos

# A classe NP

- Teorema: Uma linguagem está em NP se e somente se ela é decidida por alguma máquina de Turing não-determinística (MTN) de tempo polinomial
  - Lembrando que o tempo de uma MTN é o tempo do ramo mais longo...
  - O certificado é uma solução aceita em um dos ramos

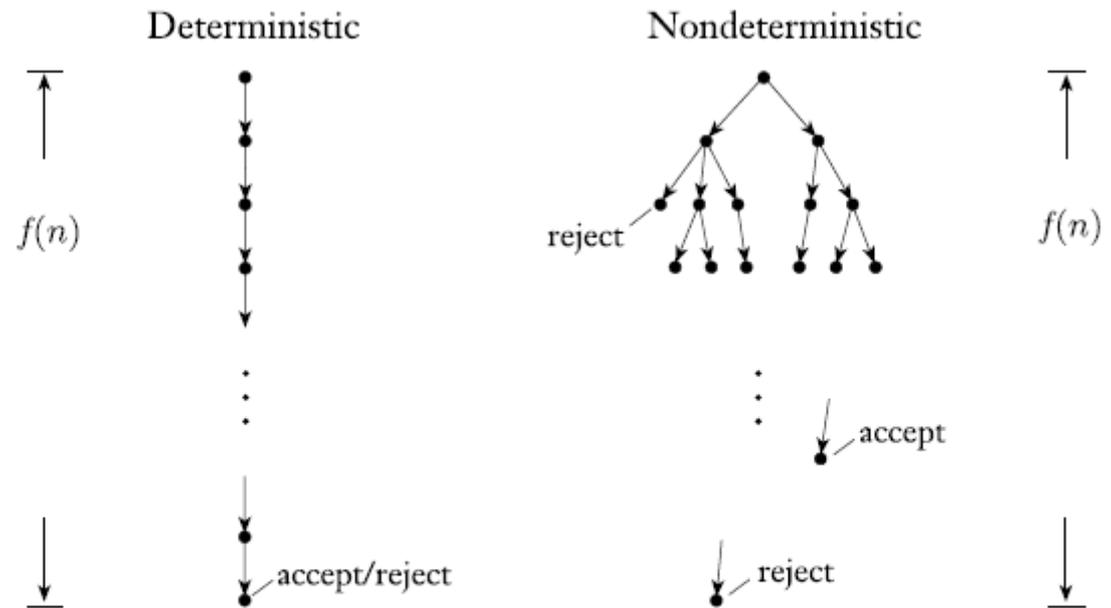
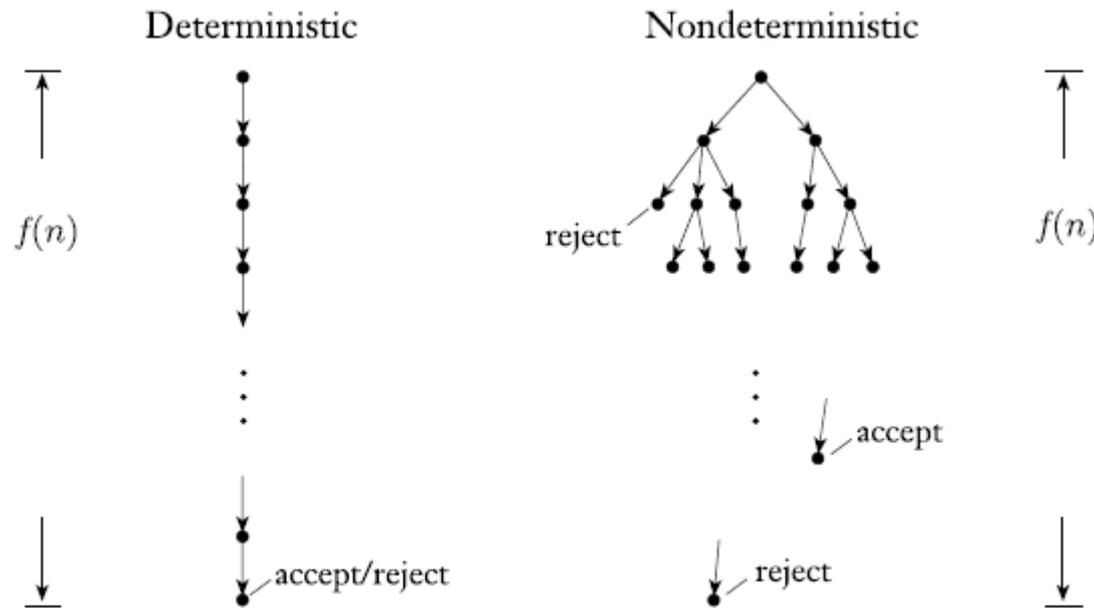


FIGURA 7.10  
Medindo tempo determinístico e não-determinístico

# A classe NP

- Ideia da prova: mostrar como converter um **verificador** de tempo polinomial para uma MTN **decisora** de tempo polinomial equivalente e vice-versa



**FIGURA 7.10**  
Medindo tempo determinístico e não-determinístico

# Prova

Uma linguagem está em NP  $\Rightarrow$  ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial

- Seja  $V$  o verificador de tempo polinomial ( $n^k$ ) de uma linguagem  $A$  em NP. A MTN  $N$  será:

$N =$  “Sobre a entrada  $w$  de comprimento  $n$ :

1. Não-deterministicamente selecione uma cadeia  $c$  de comprimento no máximo  $n^k$ .
2. Rode  $V$  sobre a entrada  $\langle w, c \rangle$
3. Se  $V$  aceita, *aceite*; caso contrário *rejeite*.”

# Prova

Uma linguagem está em NP  $\leq$  ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial

- Seja  $N$  uma MTN que decide uma linguagem  $A$  em NP.  $V$  o verificador de tempo polinomial de  $A$  será:

$V =$  “Sobre a entrada  $\langle w, c \rangle$  onde  $w$  e  $c$  são cadeias:

1. Simule  $N$  sobre a entrada  $w$ , tratando cada símbolo de  $c$  como uma descrição da escolha não-determinística a fazer a cada passo.
2. Se esse ramo da computação de  $N$  aceita, *aceite*; caso contrário *rejeite*.”

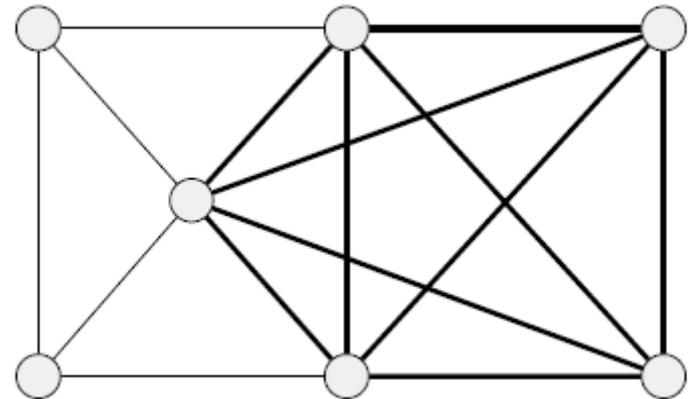
# Classe de complexidade de tempo não-determinístico

- $\text{NTIME}(t(n)) = \{ L \mid L \text{ é uma linguagem decidida por uma máquina de Turing não-determinística de tempo } O(t(n)) \}$
- $\text{NP} = \bigcup_k \text{NTIME}(n^k)$ .
- $\text{P} = \bigcup_k \text{TIME}(n^k)$

# Exemplos de problemas em NP

Dado um grafo não direcionado  $G$ :

- **Clique**: subgrafo no qual todo par de nós está conectado por uma aresta (subgrafo completo)
- **K-clique**: clique de  $k$  nós
- Exemplo de 5-clique:



# Exemplos de problemas em NP

Linguagem de um grafo com um clique (de qualquer tamanho:)

$\text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ é um grafo não-direcionado com um } k\text{-clique} \}$

# Clique está em NP

- Ideia da prova: quem é o certificado? O clique
- Prova: **verificador V:**

V = “Sobre a entrada  $\langle\langle G, k \rangle, c \rangle$ :

1. Teste se  $c$  é um conjunto de  $k$  nós em  $G$
2. Teste se  $G$  contém todas as arestas conectando cada par de nós em  $c$
3. Se ambos os testes forem positivos, *aceite*; caso contrário, *rejeite*.”

# Clique está em NP

- Prova alternativa: a MTN  $N$  que **decide** CLIQUE:

$N$  = “Sobre a entrada  $\langle G, k \rangle$ , onde  $G$  é um grafo:

1. Não-deterministicamente, selecione um subconjunto  $c$  de  $k$  nós de  $G$
2. Teste se  $G$  contém todas as arestas conectando cada par de nós de  $c$
3. Se sim, *aceite*; caso contrário, *rejeite*.”

# coNP

- O complemento de CLIQUE está em NP?

# coNP

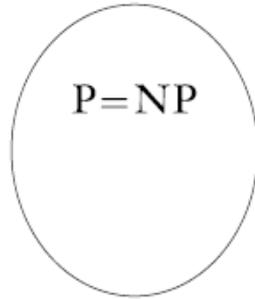
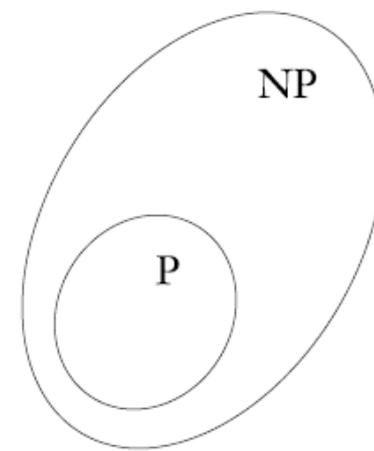
- O complemento de CLIQUE está em NP?
- Não se sabe...
- Verificar que algo NÃO está presente parece ser mais difícil...
- **CoNP** = { L | L é uma linguagem que é o complemento de uma linguagem que está em NP }

# P versus NP

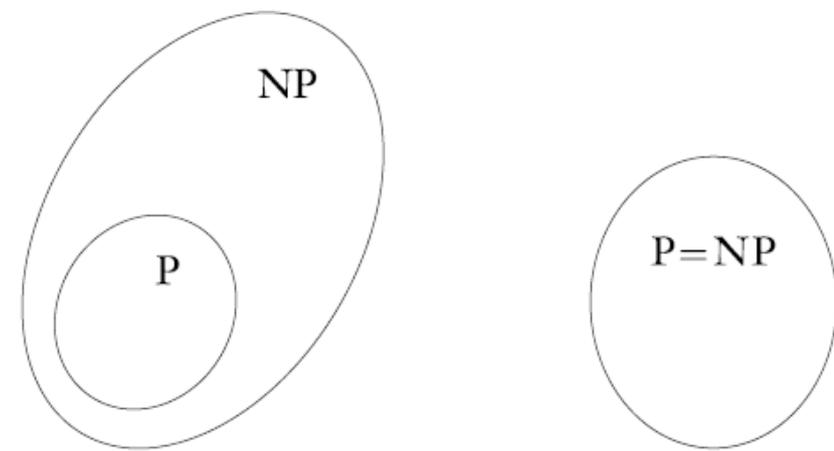
- P é um subconjunto de NP
- Questão: P é igual ou diferente a NP?

—

- 
- 
- 
- 
- 



# P versus NP



- P é um subconjunto de NP
- Questão: P é igual ou diferente a NP?
  - Um dos maiores problemas não-resolvidos da computação
- Acredita-se que sejam diferentes
- Muitos esforços para encontrar algoritmos polinomiais para certos problemas em NP
- Mas provar que  $P \neq NP$  também é complicado (provar que não existe um tal algoritmo...)
- NP subconjunto de  $EXPTIME = \bigcup_k TIME(2^{nk})$  (exponenciais)
- Mas não sabemos se NP é subconjunto de uma classe de complexidade menor

# NP-Compleitude

- Início dos anos 70:
  - Classe de problemas NP para os quais não se conhece uma solução polinomial (determinística)... mas se existir, poderá ser usada para solucionar em tempo polinomial todos os problemas em NP!
  - Problemas **NP-completos**

# NP-Compleitude

- Benefícios:
  - A prova de que UM problema NP-completo tem uma solução polinomial prova que  $P = NP$
  - A prova de que UM problema em NP exige tempo no mínimo exponencial, prova que problemas NP-completos também exigem
  - Se um problema é NP-completo, simplifique-o!

# NP-Completeness – Definição informal

- Uma linguagem B é **NP-completa** se satisfaz duas condições:
  - B está em NP
  - Toda linguagem A em NP pode ser decidida usando a solução de B usando “adaptações” polinomiais (para usar a solução de B em A)

# NP-Completeness – Definição informal

- Uma linguagem B é NP-completa se satisfaz duas condições:
  - B está em NP
  - Toda linguagem A em NP pode ser decidida usando a solução de B usando “adaptações” polinomiais (para usar a solução de B em A)

Função de redução

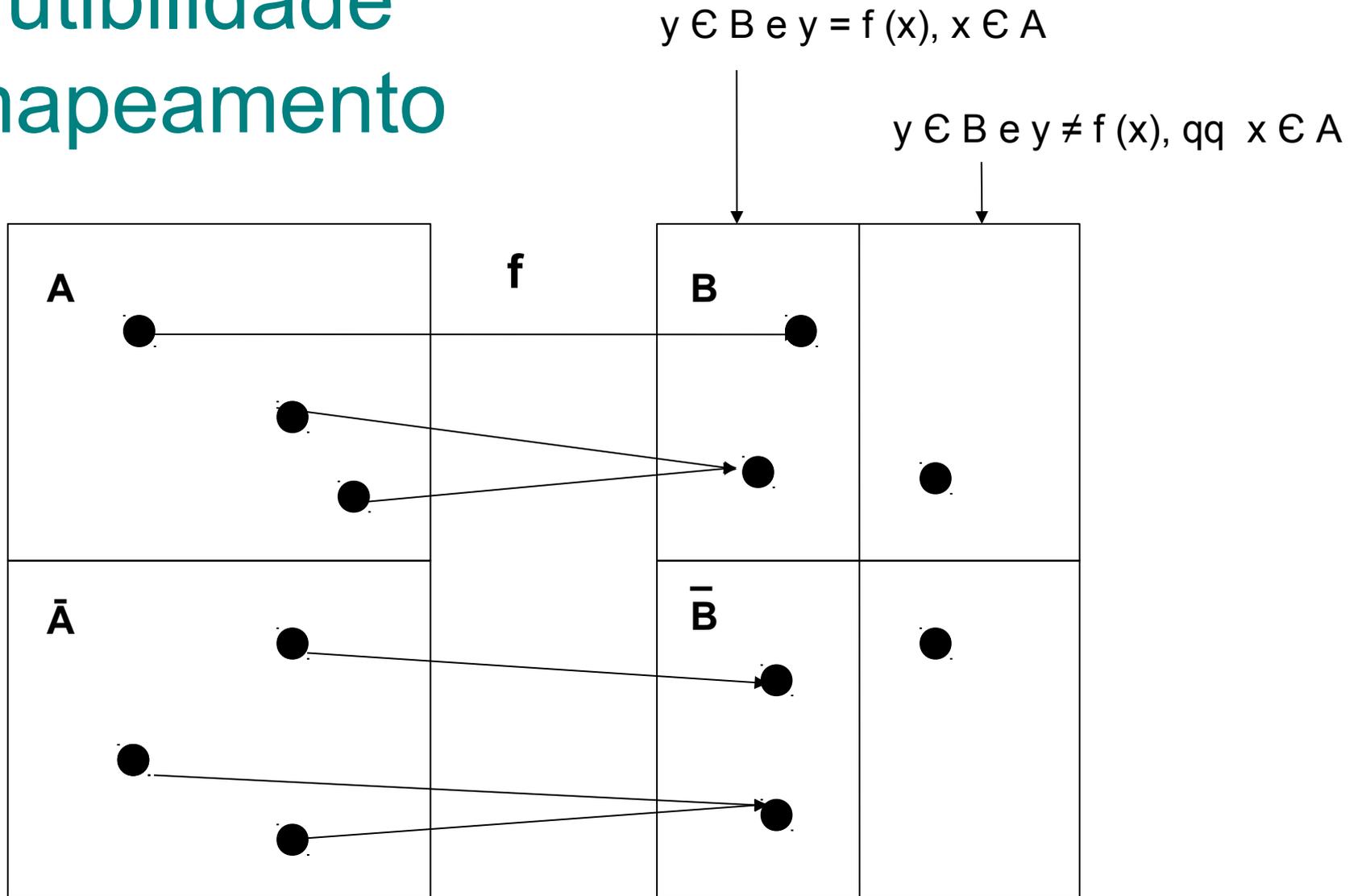
# NP-Completeness – Definição **FORMAL**

- Uma linguagem  $B$  é **NP-completa** se satisfaz duas condições:
  - $B$  está em NP
  - Toda linguagem  $A$  em NP é **reduzível em tempo polinomial a  $B$**

# Redutibilidade em tempo polinomial

- Já vimos redução por mapeamento para provarmos se uma linguagem é ou não decidível
- Como era?

# Redutibilidade por mapeamento



# Redutibilidade em tempo polinomial

- Agora usaremos a mesma ideia para provar que a decibilidade ocorre em tempo polinomial

# Redutibilidade em tempo polinomial

- Uma função  $f: \Sigma^* \rightarrow \Sigma^*$  é uma **função computável em tempo polinomial** se alguma máquina de Turing  $M$  de **tempo polinomial**, sobre toda entrada  $w$ , pára com exatamente  $f(w)$  sobre sua fita

# Redutibilidade em tempo polinomial

- A linguagem  $A$  é **redutível por mapeamento em tempo polinomial** à linguagem  $B$  ( $A \leq_P B$ ), se existe uma função computável em tempo polinomial  $f: \Sigma^* \rightarrow \Sigma^*$  onde para toda  $w$ ,

$w$  pertence a  $A \iff f(w)$  pertence a  $B$ .

A função  $f$  é denominada a **redução de tempo polinomial** de  $A$  para  $B$ .

# Teorema

- Se  $A \leq_p B$  e  $B \in P$ , então  $A \in P$
- Prova: Seja  $M$  o algoritmo de tempo polinomial que decide  $B$ . O seguinte algoritmo  $N$  de tempo polinomial decide  $A$ :

$N =$  “Sobre a entrada  $w$ :

1. Compute  $f(w)$
  2. Rode  $M$  sobre a entrada  $f(w)$  e dê como saída o que  $M$  der como saída ”
- Por que  $f$  deve ser polinomial?
    - Para que  $N$  também seja (não adiantaria  $M$  ser polinomial se  $f$  não fosse)

# NP-Completeness – Definição FORMAL

- Uma linguagem B é NP-completa se satisfaz duas condições:
  - B está em NP
  - **Toda** linguagem A em NP é redutível em tempo polinomial a B

# Como provar que um problema B é NP-completo

- A princípio, teríamos que provar que
  - B está em NP
  - **Todo** problema em NP é redutível em tempo polinomial a B
- É o que foi feito com os problemas SAT e 3SAT
- Mas o **"TODO"** pode ser complicado na prova para muitos problemas
- Estratégia:
  - partir de um problema A que já se sabe que é NP-completo (ex: SAT ou 3SAT)
  - achar uma redução em tempo polinomial de A para B

# Como provar que um problema B é NP-completo

- Se B pertence a NP e  
se  $A \leq_p B$  e A é NP-completo  
então B é NP-completo
- Ex: CLIQUE é NP-completo:  
CLIQUE pertence a NP e  $3SAT \leq_p CLIQUE$
- Precisamos identificar estruturas no problema alvo que simulem as variáveis e cláusulas booleanas
- Podemos fazer a redução a partir de qualquer problema NP-completo (embora 3SAT seja bastante usado)
  - Por isso é importante conhecer alguns

# SAT é NP-completo

- Teorema de Cook-Levin: SAT é NP-completo
- Precisamos provar que
  - SAT está em NP
  - Qualquer problema em NP é redutível em tempo polinomial a SAT (SAT é NP-difícil)

# SAT – O problema da satisfazibilidade

- **Variáveis booleanas:** valores falso ou verdadeiro (0 ou 1)
- **Operações booleanas:** E ( $\wedge$ ), OU ( $\vee$ ), NÃO ( $\neg$ )
  - Usaremos ! No lugar de  $\neg$
- **Fórmula booleana:** expressão contendo variáveis e operações booleanas
  - Ex:  $\Phi = (!x \wedge y) \vee (x \wedge !z)$
- **Fórmula booleana satisfazível:** existem valores das variáveis para os quais a fórmula é igual a 1
  - Ex:  $x = 0, y = 1, z = 0$  satisfaz  $\Phi$
- **SAT** =  $\{ \langle \Phi \rangle : \Phi \text{ é uma fórmula booleana satisfazível} \}$

# 3SAT – O problema da satisfazibilidade

- **Literal**: uma variável booleana ou sua negação
  - ex:  $x$  ou  $\neg x$
- **Cláusula**: fórmula contendo apenas “OUs” de literais
  - ex:  $x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4$
- Forma normal conjuntiva (**fnc-fórmula**): cláusulas conectadas por “E”s (conjunção de disjunções)
  - Ex:  $(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee \neg x_5 \vee x_6) \wedge (x_3 \vee \neg x_6)$
- **3fnc-fórmula**: todas as cláusulas têm exatamente 3 literais
  - Ex:  $(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_3 \vee \neg x_5 \vee x_6) \wedge (x_3 \vee \neg x_6 \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$
- **3SAT** =  $\{ \langle \Phi \rangle : \Phi \text{ é uma 3fnc-fórmula satisfazível} \}$ 
  - Isto é, cada cláusula de  $\Phi$  deve ter pelo menos um literal valendo 1

# Teorema

Se  $B$  for NP-completa e  $B \in P$ , então  $P = NP$

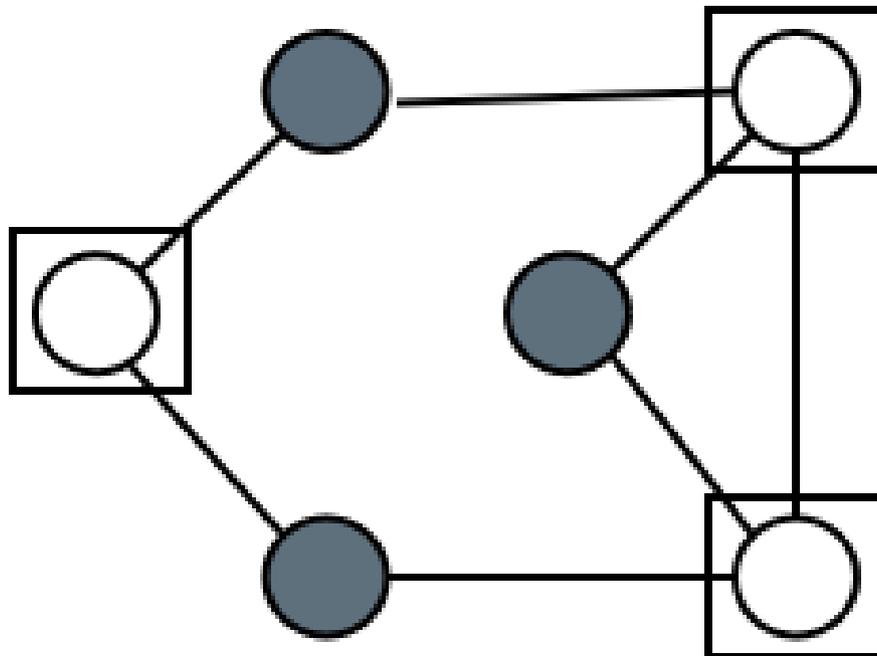
Ideia da Prova: pela redutibilidade em tempo polinomial

# Problemas NP-completos adicionais

- Há vários problemas NP-completos
- A maioria dos problemas NP está em P ou é NP-completo
- Se você está procurando um algoritmo polinomial para um novo problema NP, primeiro tente provar que ele é NP-completo

# O problema da cobertura de vértices

Uma **cobertura** de um grafo é qualquer conjunto de vértices que contenha pelo menos uma das pontas de cada aresta. Em outras palavras, um conjunto  $X$  de vértices é uma cobertura se toda aresta do grafo tem pelo menos uma de suas pontas em  $X$ .



# O problema da cobertura de vértices

- O tamanho de uma cobertura de vértices é igual ao número de vértices da cobertura
- $\text{COB-VERT} = \{ \langle G, k \rangle \mid G \text{ é um grafo não-direcionado que tem uma cobertura de vértices de tamanho } k \}$
- Teorema: COB-VERT é NP-completa

# CAMHAM é NP-Completo

$CAMHAM = \{ \langle G, s, t \rangle : G \text{ é um grafo direcionado com um caminho hamiltoniano de } s \text{ a } t \}$

Já vimos que  $CAMHAM \in NP$

e  $3SAT \leq_p CAMHAM$