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 intratável (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 $f:N \rightarrow N$, 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

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

Analisando algoritmos

- Uma MT para a linguagem A = $\{0^k1^k \mid k >= 0\}$:

M1 = “Sobre a cadeia de entrada w:

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:
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*.”

$2n = O(n)$

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

$2n = O(n)$

Total: $O(n) + O(n^2) + O(n) = O(n^2)$

Classe de complexidade

- **Definição**: Seja uma função $t:N \rightarrow R^+$ (tamanho de entrada → tempo, ex: $t(n) = n^2$). A classe de complexidade de tempo **TIME(t(n))** é a coleção de todas as linguagens decidíveis por uma máquina de Turing de tempo O(t(n)).

- $A = \{0^k1^k \mid k >= 0\} \in TIME(n^2)$

Classe de complexidade

- $A = \{0^k1^k \mid k >= 0\} \in TIME(n^2)$
- Pergunta: será que $A \in TIME(t(n))$ onde $t(n) = o(n^2)$, ou seja, A pode ser decidida por uma MT que rode em menos tempo (mais rápida)?
- Sim, $A \in TIME(n \log n)$ (MT descrita no livro do Sipser)
- 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
  - $A \in TIME(n)$ se a máquina tiver duas fitas (MT descrita no livro do Sipser)

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 computação 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

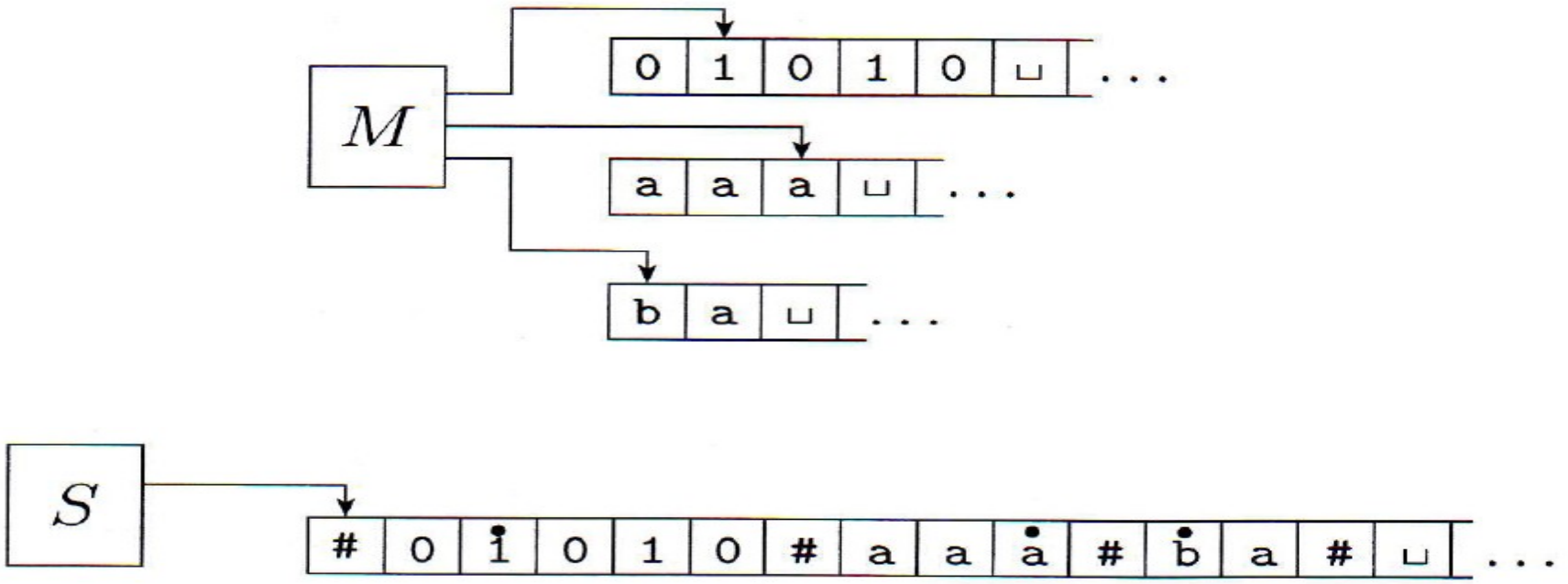
- Teorema: Seja t(n) uma função, onde t(n) >= n. Toda máquina de Turing multifita de tempo t(n) tem uma máquina de Turing fita-única equivalente de tempo $O(t^2(n))$

Relacionamentos de Complexidade entre modelos

- Teorema: Seja t(n) uma função, onde t(n) >= 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 importa 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,

$$P = U_k \, TIME(n^k).$$

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

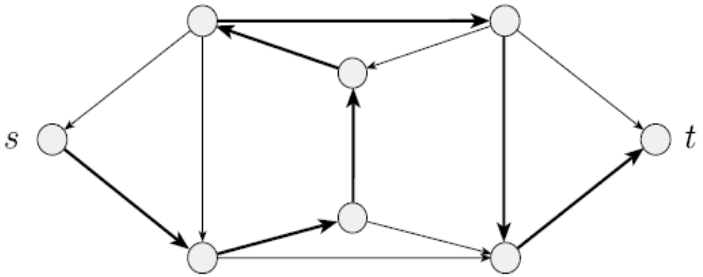
Analisando se um problema está em P

- Descreveremos ALGORITMOS (não considerando um modelo computacional específico, apenas assumindo que é determinístico, ie, sem as várias ramificações)
  - 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 ?

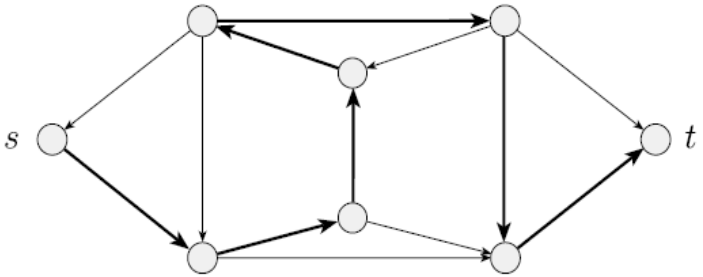
- CAM = {<G,s,t> | G é um grafo direcionado que tem um caminho direcionado do nó s para o nó t}
- Codificações possíveis de G:
  - lista de nós e arestas
  - matriz de adjacência



- Uma solução força-bruta (verifica todas as possibilidades):
  - Verifica todos os caminhos em potencial de comprimento no máximo m (m = número de nós)
  - Número de caminhos em potencial:

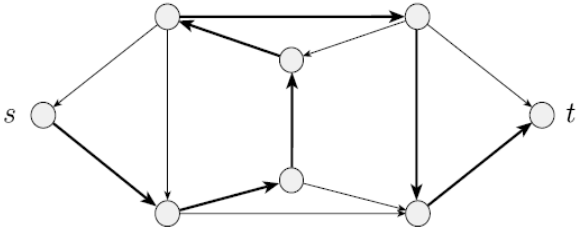
CAM pertence a P ?

- CAM = {<G,s,t> | G é um grafo direcionado que tem um caminho direcionado do nó s 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 = 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 <G,s,t> 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 <G,s,t> 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 <G,s,t> 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 <G,s,t> 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 <G,s,t> 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 <G, s, t>
  - Polinomial

Teorema: CAM pertence a P

- Prova: O algoritmo M é polinomial:

M = “Sobre a entrada <G,s,t> 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 <G, s, t>

– Polinomial

LOGO CAM Є 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 nxn:
  - i <= j : varíaveis que geram a subcadeia $w_i...w_j$
  - 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 $S \to \varepsilon$ 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 $A \to \mathtt{b}$ é uma regra, onde $\mathtt{b} = w_i$.
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 $A \to BC$:
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 P

- Um problema está na classe **P** se existe uma MT **determinística** que o resolva e que rode em tempo **polinomial**.
  - O que corresponde a decidir a linguagem

A classe NP

A classe NP

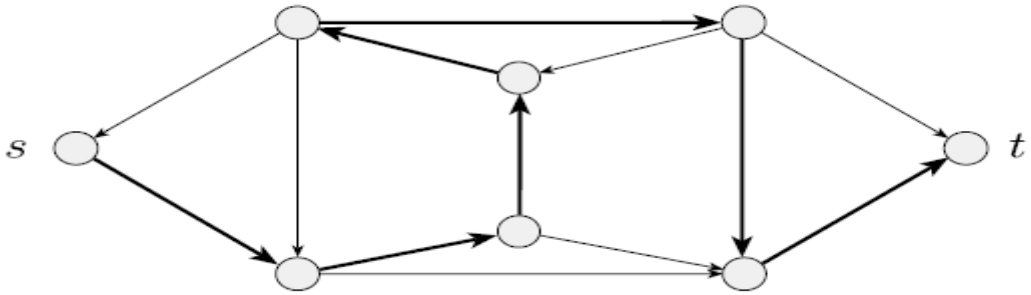
- E se um algoritmo de tempo polinomial não é conhecido?
  - Ou o problema é intrinsicamente difícil
  - Ou um algoritmo polinomial existe mas não é conhecido ainda
- Há vários problemas nesta situação
- Não sabemos distinguir entre os dois casos
- Para certos problemas, embora para DECIDIR conheça-se apenas algoritmos exponenciais, VERIFICAR se uma solução candidata é mesmo uma solução pode ser feito em tempo polinomial mesmo em uma máquina de Turing determinística

A classe NP

- E se um algoritmo de tempo polinomial não é conhecido?
  - Ou o problema é intrinsicamente difícil
  - Ou um algoritmo polinomial existe mas não é conhecido ainda
- 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 mesmo em uma máquina de Turing determinística

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 = { <G, s, t>: 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)

- Algoritmo exponencial que decide CAMHAM:

  Gere todos os caminhos possíveis (parte exponencial)

  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 | V aceita <w,c> 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, que deveria ser hamiltoniano)
- Tempo do verificador medido em termos apenas do comprimento de w
- 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** (em máquinas de Turing determinísticas)
- 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 | x = pq, 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 | x = pq, para inteiros p,q >1}
  - Certificado? Um dos divisores
  - Há um algoritmo polinomial que o resolve (está em P)

A classe NP

- NP vem de tempo polinomial não-determinístico

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

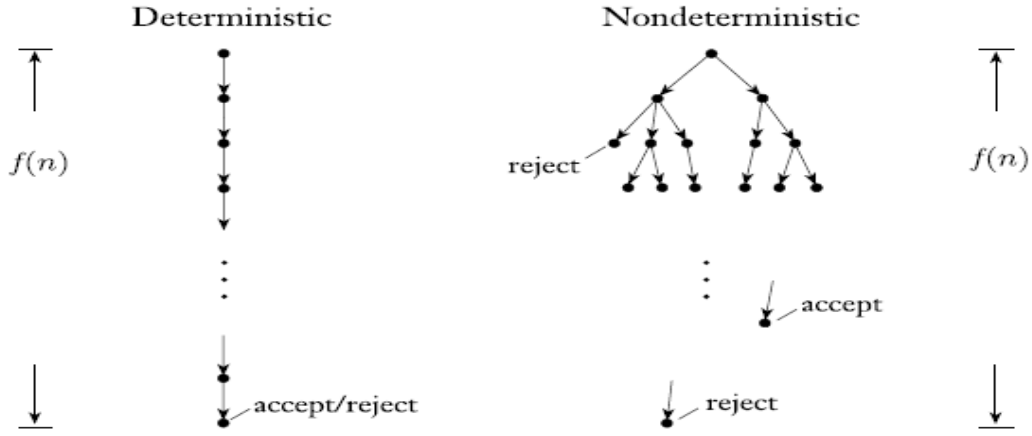


FIGURA **7.10**
Medindo tempo determinístico e não-determinístico

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

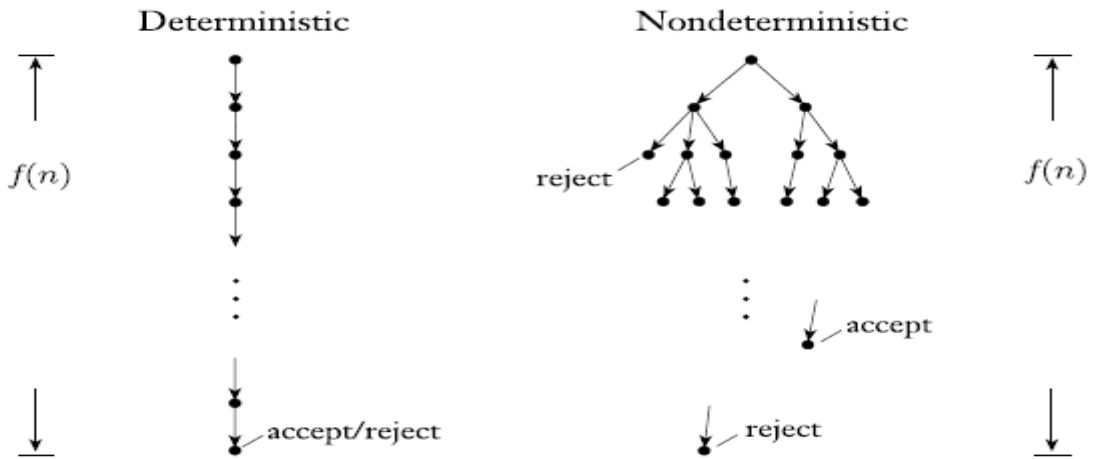


FIGURA **7.10**
Medindo tempo determinístico e não-determinístico

Prova

Uma linguagem está em NP => ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial

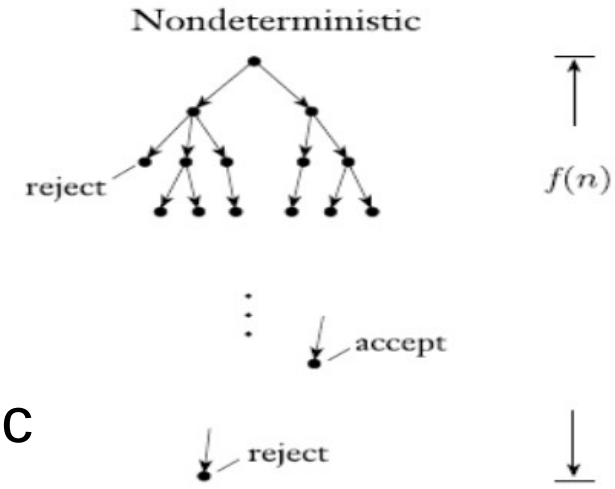
- Seja V o verificador de tempo polinomial (n^k) de uma linguagem A em NP. A MTN N será:

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 <w,c>

3. Se V aceita, *aceite*; caso contrário *rejeite*.”



Prova

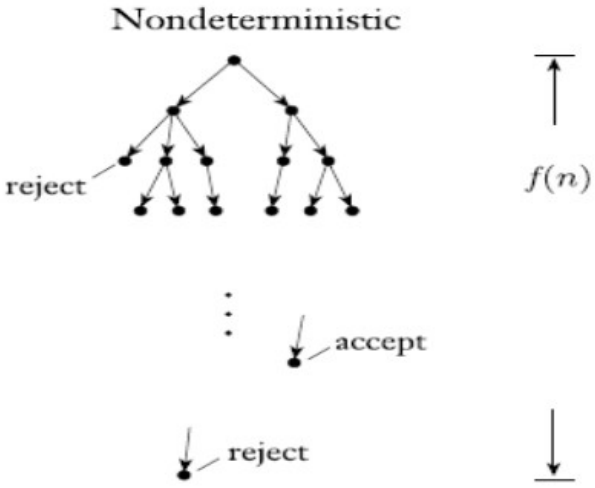
Uma linguagem está em NP <= ela é decidida por alguma máquina de Turing não-determinística de tempo polinomial

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

V = “Sobre a entrada <w,c> 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*.”



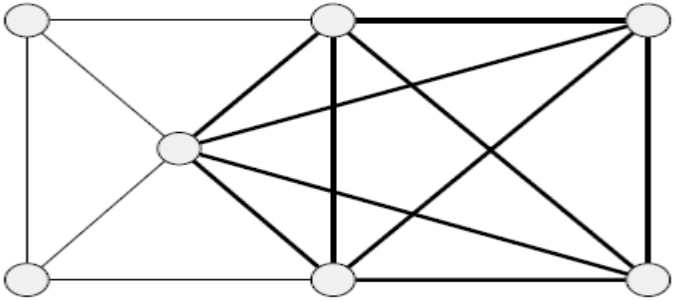
Classe de complexidade de tempo não-determinístico

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

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:)

CLIQUE = {<G, k> | G é um grafo não-direcionado com um k-clique}

Clique está em NP

- Ideia da prova: quem é o certificado?

Clique está em NP

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

V = “Sobre a entrada <<G,k>,c>:

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 <G,k>, 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*.”

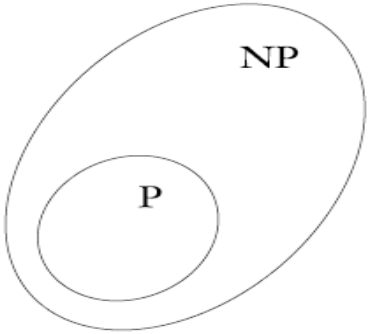
coNP

- O complemento de CLIQUE está em NP?

coNP

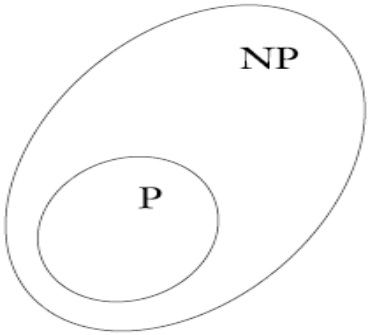
- O complemento de CLIQUE está em NP?
- Não se sabe...
- Verificar que algo NÃO está presente parece ser mais difícil...
- CoNP = { L | L é uma linguagem que é o complemento de uma linguagem que está em NP}

P versus NP



- P é um subconjunto de NP
- Questão: P é igual ou diferente (subconjunto próprio) a NP?
  - –
-
-
-
-
-

P versus NP



- P é um subconjunto de NP
- Questão: P é igual ou diferente (subconjunto próprio) 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 ≠ NP também é complicado (provar que não existe um tal algoritmo...)
- NP subconjunto de EXPTIME = U_k TIME(2^{nk}) (exponenciais)
- Mas não sabemos se NP é subconjunto de uma classe de complexidade menor

NP-Completude

NP-Completude

- 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-Completude

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

NP-Completude – 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-Completude – 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

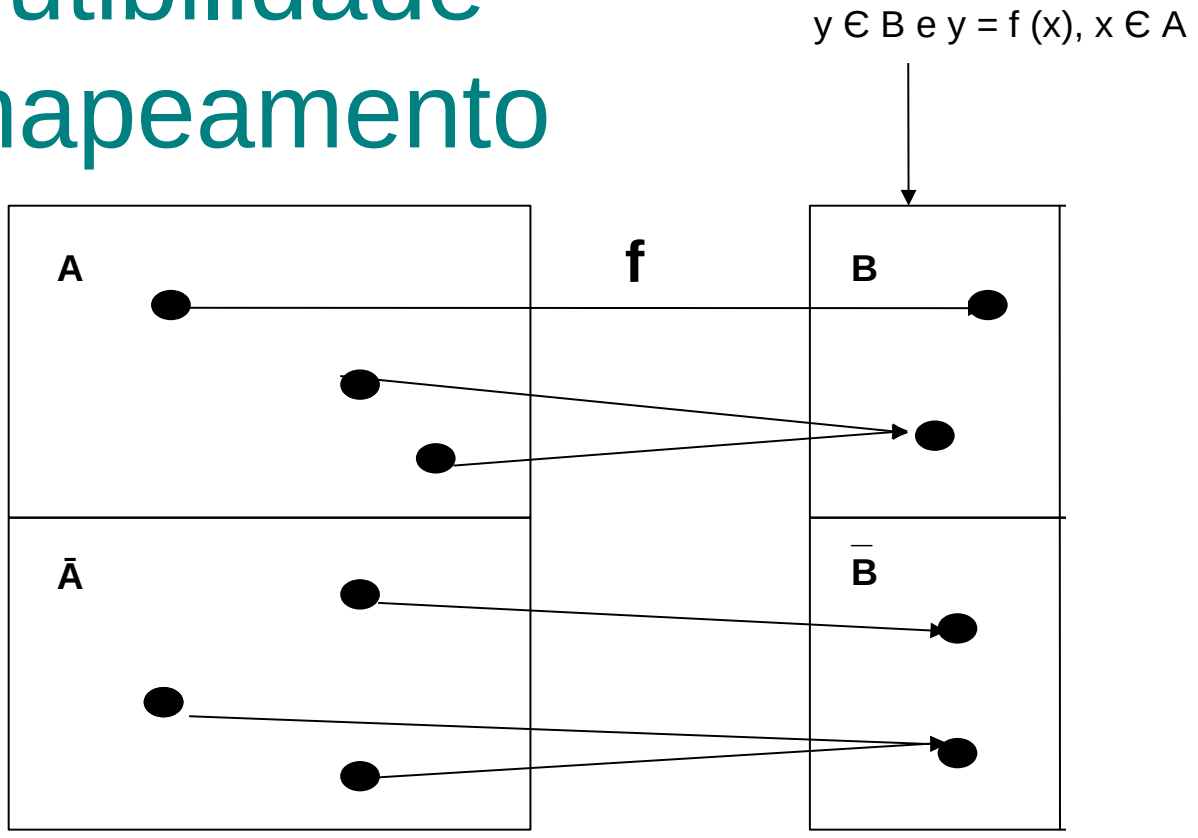
NP-Completude – Definição FORMAL

- 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

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 determinística 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 <=> 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-Completude – 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 (**NP-difícil**)

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

$$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$$

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

$(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$

- Variáveis boolenas: valores falso ou verdadeiro (0 ou 1)
- Operações boolenas: E (^), OU (v), NÃO (¬ ou - ou ¯ ou !)
  - Usaremos ! No lugar de ¬
- Fórmula booleana: expressão contendo variáveis e operações booleanas
  - Ex: Φ = (!x ^ y) v (x ^ !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 Φ
- SAT = { <Φ>: Φ é uma fórmula booleana satisfazível}

3SAT – O problema da satisfazibilidade

- Literal: uma variável booleana ou sua negação
  - ex: x ou !x
- Cláusula: fórmula contendo apenas “OUs” de literais (disjunção)
  - ex: x1 v !x2 v !x3 v x4
- Forma normal conjuntiva (fnc-fórmula): cláusulas conectadas por “E”s (conjunção de disjunções)
  - Ex: (x1 v !x2 v !x3 v x4) ^ (x3 v !x5 v x6) ^ (x3 v !x6)
- 3fnc-fórmula: todas as cláusulas têm exatamente 3 literais
  - Ex: (x1 v !x2 v !x3) ^ (x3 v !x5 v x6) ^ (x3 v !x6 v x4) ^ (x4 v x5 v x6)
- 3SAT = { <Φ>: Φ é uma 3fnc-fórmula satisfazível}
  - Isto é, cada cláusula de Φ deve ter pelo menos um literal valendo 1

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 $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_3 \vee \overline{x_4})$
- 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

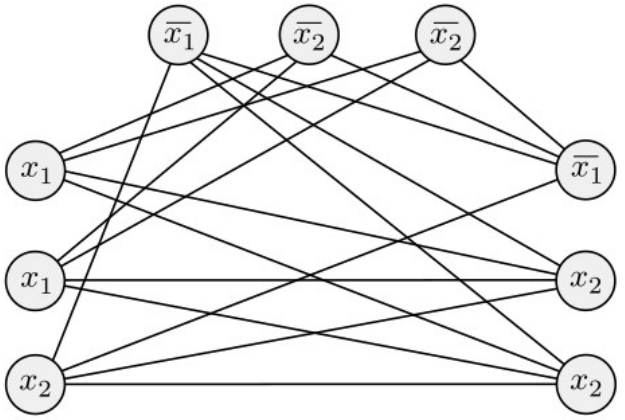


FIGURA **7.33**
O grafo que a redução produz de
$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

- 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

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

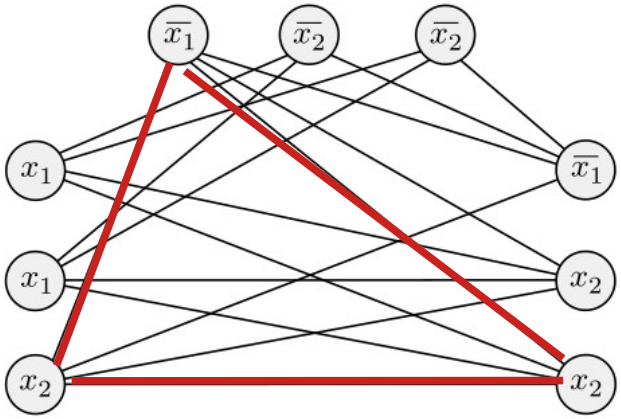


FIGURA **7.33**
O grafo que a redução produz de
$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

- 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

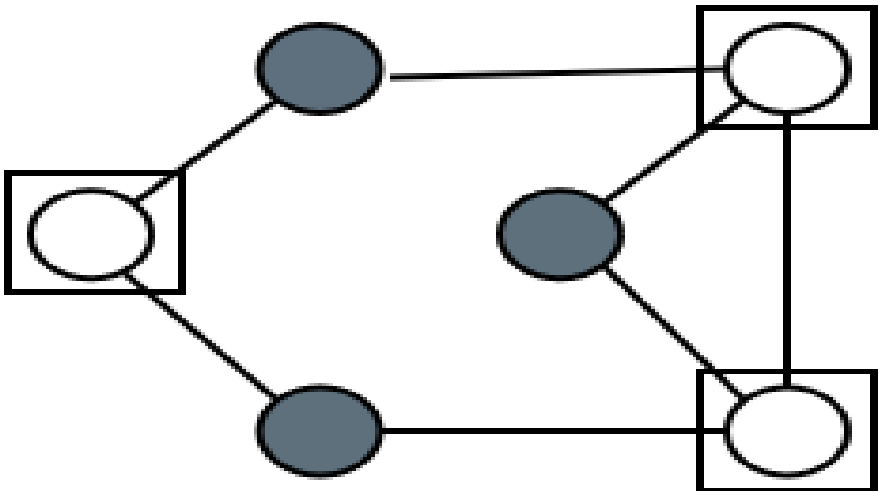
Teorema

Se B for NP-completa e B Є P, então P = NP

Ideia da Prova: pela redutibilidade em tempo polinomial

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

- COB-VERT = { <G, k> | G 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 = {<G,s,t> : G é um grafo direcionado com um caminho hamiltoniano de s a t}

Já vimos que CAMHAM ϵ NP

e 3SAT $\leq_p$ CAMHAM

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

Vídeo

What is complexity theory? (P vs. NP explained visually)

https://www.youtube.com/watch?v=u2DLlNQiPB4

ACH2043
INTRODUÇÃO À TEORIA DA COMPUTAÇÃO

Aula 25

Cap 7 – Complexidade de Tempo

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