MAC 414
Autômatos, Computabilidade e Complexidade
aula 19 — 25/11/2020

Oráculos

Oráculos

Termo usado pelo Turing.

Oráculos

Termo usado pelo Turing.

Idéia: computação com uma caixa preta.
Encapsulamento.

Oráculos

Termo usado pelo Turing.

Idéia: computação com uma caixa preta.
Encapsulamento.

Uma função de biblioteca com implementação não especificada.

Oráculos

Termo usado pelo Turing.

Idéia: computação com uma caixa preta.
Encapsulamento.

Uma função de biblioteca com implementação não especificada.

Isso aumenta o poder da linguagem de programação, é possível computar mais coisas.

Oráculos em computabilidade

Oráculos em computabilidade

Um oráculo é simplesmente uma função $\Sigma^* \to \Sigma^*$.

Oráculos em computabilidade

Um oráculo é simplesmente uma função $\Sigma^* \to \Sigma^*$.

Uma MT com oráculo f é uma MT com uma fita extra e dois estados especiais, q_c e q_r. A computação é a normal de uma máquina com duas fitas, exceto no estado q_c.

Oráculos em computabilidade

Um oráculo é simplesmente uma função $\Sigma^* \to \Sigma^*$.

Uma MT com oráculo f é uma MT com uma fita extra e dois estados especiais, q_c e q_r. A computação é a normal de uma máquina com duas fitas, exceto no estado q_c.

Nesse caso, em um único passo, se x é o conteúdo da fita extra, ele é substituído por $f(x)$ e a MT passa ao estado q_r.

Oráculos em computabilidade

Um oráculo é simplesmente uma função $\Sigma^* \to \Sigma^*$.

Uma MT com oráculo f é uma MT com uma fita extra e dois estados especiais, q_c e q_r. A computação é a normal de uma máquina com duas fitas, exceto no estado q_c.

Nesse caso, em um único passo, se x é o conteúdo da fita extra, ele é substituído por $f(x)$ e a MT passa ao estado q_r.

Uma função g é Turing-redutível a f se existe uma MT com oráculo f que computa g. Notação $g <_T f$.

O que se ganha?

O que se ganha?

Claro que se o oráculo é uma função recursiva, então o que se computa ainda é recursiva.

O que se ganha?

Claro que se o oráculo é uma função recursiva, então o que se computa ainda é recursiva.

O interessante é usar oráculo não computável: por exemplo, a função do problema da parada.

O que se ganha?

Claro que se o oráculo é uma função recursiva, então o que se computa ainda é recursiva.

O interessante é usar oráculo não computável: por exemplo, a função do problema da parada.

Se $\mathscr{R}$ são as funções recursivas, e $\mathscr{R}_1$ são as Turing-redutíveis ao problema da parada, então $\mathscr{R} \subset \mathscr{R}_1$.

$f(x) = \begin{cases} 1 & \text{se } x \text{ é "M" "w" e } M(w) \text{ para} \\ 0 & \text{caso contrário} \end{cases}$

Dada "M", existe x t.q. M(x) para?

toda "M", constrói "M"

1) Tenta não det. M(x) para todo $x \in \Sigma^*$

2) Se para algum x M(x) para, aceite "m"

3) Veja saída de f("m" "a")

devolva essa saída

O que se ganha?

Claro que se o oráculo é uma função recursiva, então o que se computa ainda é recursiva.

O interessante é usar oráculo não computável: por exemplo, a função do problema da parada.

Se $\mathscr{R}$ são as funções recursivas, e $\mathscr{R}_1$ são as Turing-redutíveis ao problema da parada, então $\mathscr{R} \subset \mathscr{R}_1$.

Diagonalizando $\mathscr{R}_1$ se produz uma nova f, e uma nova classe $\mathscr{R}_2$; e uma hierarquia infinita: $\mathscr{R} \subset \mathscr{R}_1 \subset \mathscr{R}_2 \subset \mathscr{R}_3 \subset \cdots$.

Turing-redução polinomial

Turing-redução polinomial

Lembrando: a chamada do oráculo conta como um passo.

Turing-redução polinomial

Lembrando: a chamada do oráculo conta como um passo.

Se $\mathscr{M}^f$ roda em tempo polinomial, ela faz um número polinomial de chamadas de f.

Turing-redução polinomial

Lembrando: a chamada do oráculo conta como um passo.

Se $\mathscr{M}^f$ roda em tempo polinomial, ela faz um número polinomial de chamadas de f.

Definimos $\mathbf{P}^f$ e $\mathbf{NP}^f$ como as classes de linguagens computáveis MTs (determinísticas ou não-determinísticas) com oráculo f em tempo polinomial.

Turing-redução polinomial

Lembrando: a chamada do oráculo conta como um passo.

Se $\mathscr{M}^f$ roda em tempo polinomial, ela faz um número polinomial de chamadas de f.

Definimos $\mathbf{P}^f$ e $\mathbf{NP}^f$ como as classes de linguagens computáveis MTs (determinísticas ou não-determinísticas) com oráculo f em tempo polinomial.

Claro que que se $f \in \mathbf{P}$, então $\mathbf{P}^f = \mathbf{P}$ e se $f \in \mathbf{NP}$ então $\mathbf{NP}^f = \mathbf{NP}$.

NP-difícil

Um problema P é **NP**-dificil se existe uma linguagem **NP**-completa L tal que $L \prec_T P$.

NP-difícil

Um problema P é **NP**-dificil se existe uma linguagem **NP**-completa L tal que $L <_T P$.

Se uma linguagem é **NP**-completa, então ela é **NP**-difícil.

NP-difícil

Um problema P é **NP**-dificil se existe uma linguagem **NP**-completa L tal que $L \prec_T P$.

Se uma linguagem é **NP**-completa, então ela é **NP**-difícil.

Prop: se P é **NP**-difícil e $P \in \mathbf{P}$, então $\mathbf{P} = \mathbf{NP}$.

Exemplos

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Dado: objeto $A \Rightarrow n$
Define uma linguagem de "soluções viáveis"
e um alg. polinomial p/ decidir se
~~x~~ x é viável.
Uma função objetivo $c: \Sigma^* \to \mathbb{R}$
achar $\max\{c(x) \mid x \text{ viável}\}$
Probl de decisão: Dado $K \in \mathbb{N}$, existe sol viável
x com $c(x) \geq K$?

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Problemas de decisão muitas vezes estão associados a um problema de busca:

- DECISÃO: dado um objeto x, existe um objeto w?

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Problemas de decisão muitas vezes estão associados a um problema de busca:

- Decisão: dado um objeto x, existe um objeto w?
- Busca: dado um objeto x, se existe um objeto w, mostre.

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Problemas de decisão muitas vezes estão associados a um problema de busca:

- DECISÃO: dado um objeto x, existe um objeto w?
- BUSCA: dado um objeto x, se existe um objeto w, mostre.

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Problemas de decisão muitas vezes estão associados a um problema de busca:

- Decisão: dado um objeto x, existe um objeto w?
- Busca: dado um objeto x, se existe um objeto w, mostre.

Decisão $\prec_T$ Busca quase que por definição.

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Problemas de decisão muitas vezes estão associados a um problema de busca:

- Decisão: dado um objeto x, existe um objeto w?
- Busca: dado um objeto x, se existe um objeto w, mostre.

Decisão $<_T$ Busca quase que por definição.

Em muitos casos Busca $<_T$ Decisão.

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Problemas de decisão muitas vezes estão associados a um problema de busca:

- Decisão: dado um objeto x, existe um objeto w?
- Busca: dado um objeto x, se existe um objeto w, mostre.

Decisão $<_T$ Busca quase que por definição.

Em muitos casos Busca $<_T$ Decisão.

Ex: Hamiltoniano

Exemplos

Já vimos como muitos problemas de otimização são Turing-redutíveis a problemas de decisão.

Problemas de decisão muitas vezes estão associados a um problema de busca:

- Decisão: dado um objeto x, existe um objeto w?
- Busca: dado um objeto x, se existe um objeto w, mostre.

Decisão $<_T$ Busca quase que por definição.

Em muitos casos Busca $<_T$ Decisão.

Ex: Hamiltoniano

Ex: Conjunto independente

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Composto: dado um natural, ele é composto?

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Composto: dado um natural, ele é composto?

Primo: dado um natural, ele é primo?

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Composto: dado um natural, ele é composto?

Primo: dado um natural, ele é primo?

Ambos estão em **NP**.

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Composto: dado um natural, ele é composto?

Primo: dado um natural, ele é primo?

Ambos estão em **NP**.

RSA

Fatoração: dado um natural, devolver um fator próprio, se houver.

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Composto: dado um natural, ele é composto?

Primo: dado um natural, ele é primo?

Ambos estão em **NP**.

Fatoração: dado um natural, devolver um fator próprio, se houver.

Fator: dado um natural N e um inteiro K, N tem um fator próprio $\leq K$?

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Composto: dado um natural, ele é composto?

Primo: dado um natural, ele é primo?

Ambos estão em **NP**.

Fatoração: dado um natural, devolver um fator próprio, se houver.

Fator: dado um natural N e um inteiro K, N tem um fator próprio $\leq K$?

Óbvio: Fator $\prec_T$ Fatoração.

Fatoração

Lembrando: dado um natural N, $n = \lg N$.

Composto: dado um natural, ele é composto?

Primo: dado um natural, ele é primo?

Ambos estão em **NP**.

Fatoração: dado um natural, devolver um fator próprio, se houver.

Fator: dado um natural N e um inteiro K, N tem um fator próprio $\leq K$?

Óbvio: Fator $\prec_T$ Fatoração.

Nem tanto: Fatoração $\prec_T$ Fator.

co-NP

co-NP

Def. 1: $L \subseteq \Sigma^*$ está em co-**NP** se $\Sigma^* \setminus L$ está em **NP**.

co-NP

Def. 1: $L \subseteq \Sigma^*$ está em co-**NP** se $\Sigma^* \setminus L$ está em **NP**.

Def. 2: um problema de decisão está em co-**NP** se existe um certificado sucinto sempre que a resposta for **não**.

co-NP

Def. 1: $L \subseteq \Sigma^*$ está em co-**NP** se $\Sigma^* \setminus L$ está em **NP**.

Def. 2: um problema de decisão está em co-**NP** se existe um certificado sucinto sempre que a resposta for **não**.

Equivalência restrita de expressões regulares está em co-**NP**.

co-NP

Def. 1: $L \subseteq \Sigma^*$ está em co-**NP** se $\Sigma^* \setminus L$ está em **NP**.

Def. 2: um problema de decisão está em co-**NP** se existe um certificado sucinto sempre que a resposta for **não**.

Equivalência restrita de expressões regulares está em co-**NP**.

Por um bom tempo, alguns problemas importantes estavam em $\mathbf{P} \cap$ co-**NP**.

- Programação linear.

co-NP

Def. 1: $L \subseteq \Sigma^*$ está em co-**NP** se $\Sigma^* \setminus L$ está em **NP**.

Def. 2: um problema de decisão está em co-**NP** se existe um certificado sucinto sempre que a resposta for **não**.

Equivalência restrita de expressões regulares está em co-**NP**.

Por um bom tempo, alguns problemas importantes estavam em **P** $\cap$ co-**NP**.

- Programação linear.
- Primo

co-NP

Def. 1: $L \subseteq \Sigma^*$ está em co-**NP** se $\Sigma^* \setminus L$ está em **NP**.

Def. 2: um problema de decisão está em co-**NP** se existe um certificado sucinto sempre que a resposta for **não**.

Equivalência restrita de expressões regulares está em co-**NP**.

Por um bom tempo, alguns problemas importantes estavam em $\mathbf{P} \cap$ co-**NP**.

- Programação linear.
- Primo

co-NP

Def. 1: $L \subseteq \Sigma^*$ está em co-**NP** se $\Sigma^* \setminus L$ está em **NP**.

Def. 2: um problema de decisão está em co-**NP** se existe um certificado sucinto sempre que a resposta for **não**.

Equivalência restrita de expressões regulares está em co-**NP**.

Por um bom tempo, alguns problemas importantes estavam em **P** $\cap$ co-**NP**.

- Programação linear.
- Primo

Foram provados estar em **P**.

P ∩ co-NP

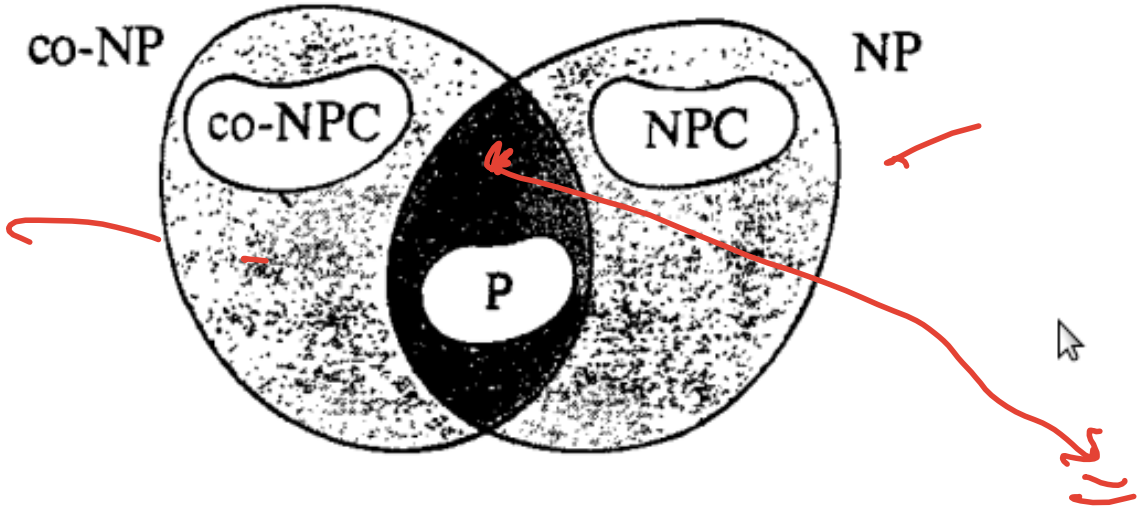


Figure 7.2 The world of NP and vicinity (assuming $P \neq NP$ and $NP \neq$ co-NP). It may or may not be the case that $P = NP \cap$ co-NP.

P ∩ co-NP

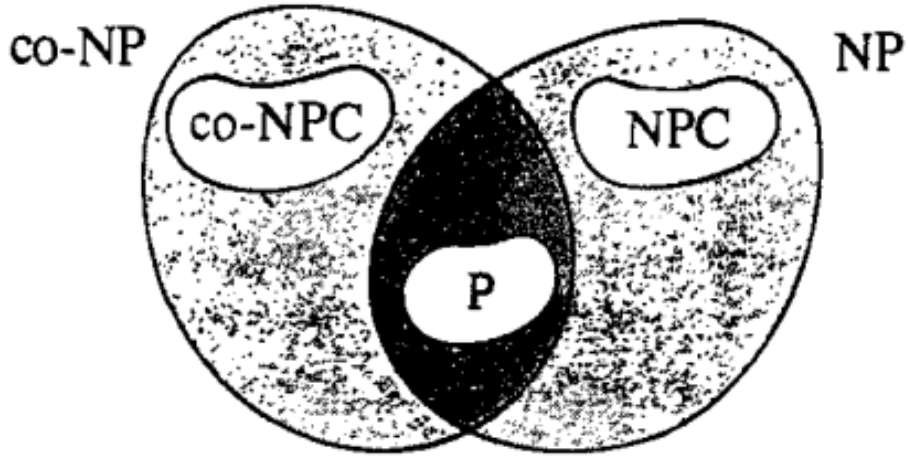


Figure 7.2 The world of NP and vicinity (assuming $P \neq NP$ and $NP \neq$ co-NP). It may or may not be the case that $P = NP \cap$ co-NP.

Garey & Johnson, 1979. A figura não mudou.

P ∩ co-NP

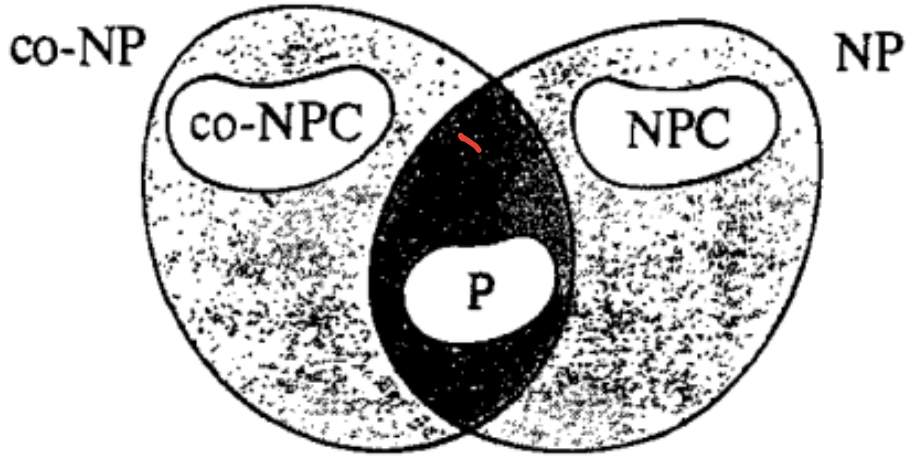


Figure 7.2 The world of NP and vicinity (assuming $P \neq NP$ and $NP \neq$ co-NP). It may or may not be the case that $P = NP \cap$ co-NP.

Garey & Johnson, 1979. A figura não mudou.

Prop: Se existe *L* **NP**-completa que está em co-**NP**, então **NP** = co-**NP**.

Problemas indecididos

Problemas indecididos

Estão em **NP**, mas não se sabe se são **NP** completos.

Problemas indecididos

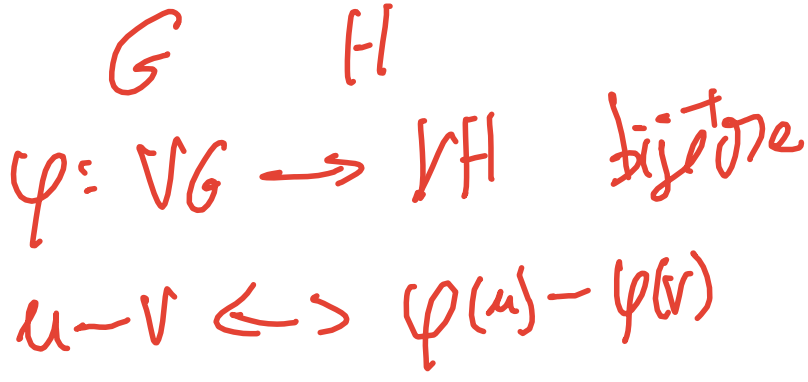
Estão em **NP**, mas não se sabe se são **NP** completos.

FATOR

Problemas indecididos

Estão em **NP**, mas não se sabe se são **NP** completos.

FATOR



GRAPH ISOMORPHISM