ITC: Introdução à Teoria da Computação

Marcos Castilho

DInf/UFPR

4 de agosto de 2021

Complexidade computacional

- $\mathcal{P} = \mathcal{NP}$? Provavelmente, não.
- Mas, se um algoritmo polinomial determinístico for encontrado?
- Ex. Se encontra um para circuito Hamiltoniano. Qual a consequência?

Reduções polinomiais

- Sejam Q e L duas linguagens com alfabetos Σ_1 e Σ_2, repectivamente;
- Dizemos que Q é redutível para L em tempo polinomial se existe uma função computável em tempo polinomial $f : \Sigma_1 \rightarrow \Sigma_2$ tal que $u \in Q$ se, e somente se, $f(u) \in L$.

Reduções polinomiais

- Uma redução de Q para L transforma o problema de aceitar Q para o de aceitar L.
- Seja F uma MT que computa a função f. Se L é aceita por uma MT M, então Q é aceita por uma MT que:
  - Roda F sobre uma entrada $u \in \Sigma_1^*$
  - Roda M sobre $f(u)$.
- A palavra $f(u)$ é aceita por M se, e somente se, $u \in Q$.

Teorema

Seja Q redutível para L em tempo polinomial e seja $L \in \mathcal{P}$. Então $Q \in \mathcal{P}$.

Definição: NP-difícil

Uma linguagem L é dita ser NP-difícil se para todo $Q \in \mathcal{NP}$, Q é redutível para L em tempo polinomial.

Definição: NP-completo

Uma linguagem $\mathcal{NP}$-difícil que também está em $\mathcal{NP}$ é chamada de $\mathcal{NP}$-completa.

Teorema

Se existe uma linguagem $\mathcal{NP}$-completa que também está em $\mathcal{P}$, então $\mathcal{P} = \mathcal{NP}$.

Prova

- Assuma que L é uma linguagem $\mathcal{NP}$-completa que é aceita em tempo polinomial por uma MT determinística. Seja Q uma linguagem de $\mathcal{NP}$.
- Como L é $\mathcal{NP}$-difícil, então existe uma redução polinomial de Q para L.
- Pelo teorema anterior, Q também está em $\mathcal{P}$.

O problema da Satisfazibilidade (SAT)

- Foi o primeiro problema provado ser $\mathcal{NP}$-completo.
- Consiste em saber se existe uma valoração de proposições na lógica proposicional que torne a fórmula verdadeira.

O problema da Satisfazibilidade (SAT)

- Uma variável booleana é uma variável que pode assumir os valores 0 ou 1.
- Uma variável (ou proposição) x é verdadeira quando recebe o valor 1 e é falsa caso contrário.
- Uma função de atribuição é uma função que associa um valor 0 ou 1 a cada variável booleana em uma fórmula.

Fórmula bem formada (FBF)

Seja V um conjunto de variáveis booleanas:

- Se $x \in V$, então x é uma FBF;
- Se u, v são FBF's então $(u), (\neg u), (u \wedge v)$ e $(u \vee v)$ são FBF's;
- Uma expressão é uma FBF sobre V somente se puder ser obtida a partir das variáveis booleanas em V por um número finito de aplicações so passo anterior.

Exemplos

- $((\neg(x \vee y)) \wedge z)$
- $(((x \wedge y) \vee z) \vee \neg(x))$
- $(((\neg x_{\vee} y) \wedge (x \vee z))$

Convenções

Usando-se precedência de operadores (negação, conjunção e disjunção), além de associatividade, podemos escrever assim:

- $\neg(x \vee y) \wedge z$
- $x \wedge y \vee z \vee \neg x$
- $(\neg x \vee y) \wedge (x \vee z)$

Valores verdade

- Os valores verdade de variáveis são obtidos diretamente do seu valor dado pela função de atribuição f.
- Os valores das FBF's podem ser obtidos assim:
  - $f(\neg u) = 1 - f(u)$
  - $f(u \wedge v) = min(f(u), f(v))$
  - $f(u \vee v) = max(f(u), f(v))$

Definições

- Uma cláusula é uma FBF que consiste de uma disjunção de variáveis ou de negações de variáveis.
- Uma variável sem a negação é chamada de literal positivo.
- Uma variável com a negação é chamada de literal negativo.
- Assim, uma cláusula é uma disjunção de literais.
- Exemplos: $x \vee \neg y$, $\neg x \vee \neg y \vee z$.

Forma normal conjuntiva (FNC)

- Uma fórmula está em Forma normal conjuntiva se puder ser escrita assim:
- $u_1 \wedge u_2 \wedge \ldots \wedge u_n$,
- onde cada u_i é uma cláusula.
- Isto, uma fórmula está em FNC se for uma conjunção de cláusulas.
- Toda fórmula pode ser transformada para FNC.

O problema da Satisfazibilidade (SAT)

- É o problema de dedicir se uma fórmula em FNC é satisfazível para alguma atribuição de valores.
- Uma fórmula é satisfazível se resulta em valor verdadeiro para alguma atribuição de valores.

Exemplos

- $u = (x \vee y) \wedge (\neg y \vee \neg z)$
- $v = (x \vee \neg y \vee \neg z) \wedge (x \vee z) \wedge (\neg x \vee \neg y)$
- $w = \neg x \wedge (x \vee y) \wedge (\neg y \vee x)$

Seja a atribuição de valores:

- $x = 1$
- $y = 0$
- $z = 1$

Solução determinística para SAT

- Todas as atribuições de valores devem ser checadas, até que uma seja encontrada.
- O número total de possibilidades é 2^n, onde n é o número de variáveis.
- Isto é exponencial, porém, dada uma valoração, a checagem pode ser feita em tempo polinomial.

Teorema

SAT está em $\mathcal{NP}$.

Prova na próxima aula.

Teorema

SAT está em $\mathcal{NP}$-difícil.

Prova na próxima aula.

Teorema

Seja Q um problema $\mathcal{NP}$-completo. Se Q é redutível em tempo polinomial para L, então L está em $\mathcal{NP}$-difícil.

Teorema

3-SAT está em $\mathcal{NP}$-completo.

- 3-SAT é um problema similar à SAT, mas cujas cláusulas têm exatamente 3 variáveis.
- Prova: 3-SAT está em $\mathcal{NP}$ e SAT reduz polinomialmente para 3-SAT.

Teorema

Cobertura de vértices está em $\mathcal{NP}$-completo.

- Seja $G = (V, A)$ um grafo não dirigido. Um subconjunto $V_c \subseteq V$ é uma cobertura de vértices de G se para cada aresta $[u, v] \in A$ pelo menos um dentre u ou v está em V_c.
- O problema da cobertura de vértices é, dado um grafo não dirigido G e um inteiro k, existe uma cobertura de vértices de G contendo k arestas?
- Prova: Cobertura de vértices está em $\mathcal{NP}$ e 3-SAT reduz polinomialmente para cobertura de vértices.

Teorema

Circuito Hamiltoniano está em $\mathcal{NP}$-completo.

- Prova: Circuito Hamiltoniano está em $\mathcal{NP}$ e 3-SAT reduz polinomialmente para circuito Hamiltoniano.

A classe $\mathcal{NPC}$

- $\mathcal{NPC}$: é a classe dos problemas $\mathcal{NP}$-completos.
- Acredita-se que $\mathcal{NPC} \neq \mathcal{P}$.
- Se $\mathcal{NPC} \cap \mathcal{P} \neq \emptyset$, então $\mathcal{P} = \mathcal{NP}$.

A classe $\mathcal{NPI}$

- $\mathcal{NPI}$: é a classe dos problemas em $\mathcal{NP}$ que não estão nem em $\mathcal{P}$ nem em $\mathcal{NP}$-completos.
- $\mathcal{NPI} = \mathcal{NP} - (\mathcal{NPC} \cup \mathcal{P})$
- Esta classe existe? $(\mathcal{NPI} = \mathcal{NP} - (\mathcal{NPC} \cup \mathcal{P}))$
- Se $\mathcal{P} \neq \mathcal{NP}$, então $\mathcal{NPI} \neq \emptyset$.

Linguagens complementares

- Seja L uma linguagem. Então $\bar{L}$ é a linguagem complementar à L, isto é, $\bar{L} = \Sigma^* - L$.
- Uma família de linguagens F é dita ser fechada sob complementação se $\bar{L} \in F$ quando $L \in F$.
- Por exemplo, a classe $\mathcal{P}$ é fechada sob complementação.
- Basta trocar os estados finais pelos não finais na MT que reconhece um problema em $\mathcal{P}$.

Problema para o não determinismo

- Uma MT não determinística que roda em tempo polinomial adivinha uma solução e testa;
- Mas para reconhecer a linguagem complementar, teria que testar todas as possibilidades. É o problema de reconhecer quando uma linguagem **não** é reconhecida.
- Exemplo: Saber se uma valoração em uma fórmula não é SAT.
- O problema é que não há instância para ser adivinhada e todas devem ser testadas. Logo, parece que este problema não está em $\mathcal{NP}$.

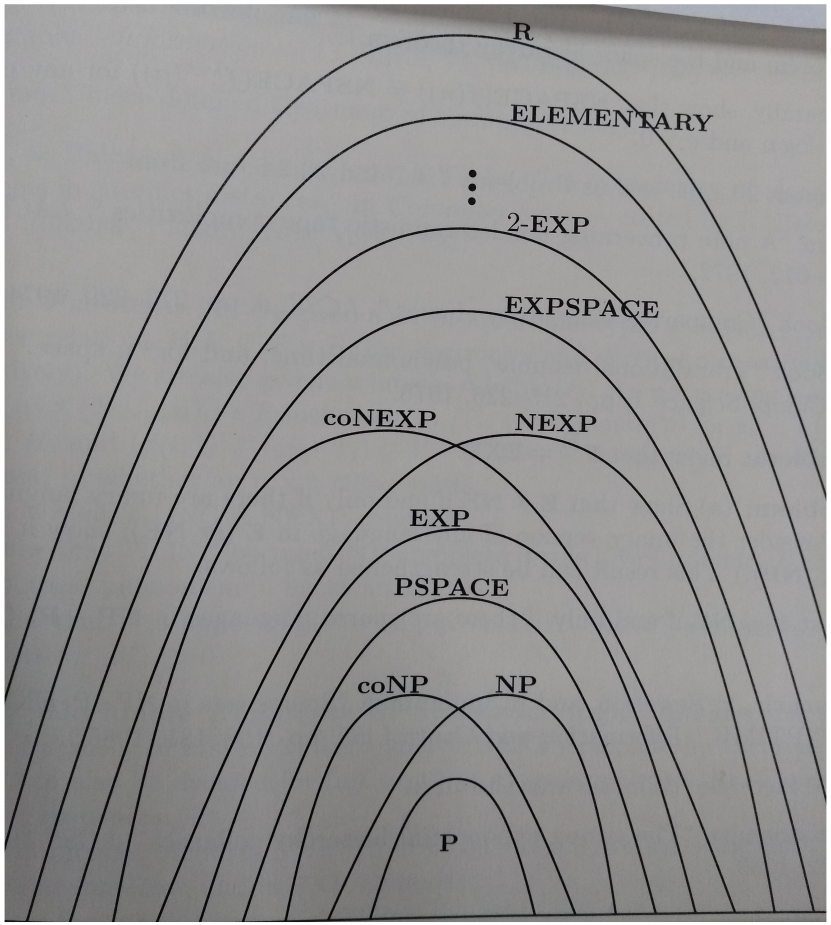
A classe co-$\mathcal{NP}$

- Seja a família co-$\mathcal{NP}$ definida como contendo os complementos de todas as linguagens que estão em $\mathcal{NP}$.
- Isto é, co-$\mathcal{NP} = \{\bar{\mathcal{L}} \mid \mathcal{L} \in \mathcal{NP}\}$.
- Se $\mathcal{NP} \neq$ co-$\mathcal{NP}$, então $\mathcal{P} \neq \mathcal{NP}$.
- Se existe uma linguagem L em $\mathcal{NP}$-completo com $\bar{L} \in \mathcal{NP}$, então $\mathcal{NP} =$ co-$\mathcal{NP}$.

Enfim

- Existem muitas outras classes de problemas.
- Elas servem para comparar problemas e relacionar os que são similares.
- Mas um estudo delas requer pelo menos mais outro semestre de estudo, se for para fazer bem feito.
- A seguir, uma figura que dá uma ideia disso tudo. . . .

Classes de complexidade



Licença

- Slides feitos em LaTeX usando beamer e tikz, editados com vim.
- Licença