Complexidade Computacional

Prof. Daniel Kikuti

Universidade Estadual de Maringá

Visão geral

Complexidade Computacional

Ramo da Teoria da Computação que estuda a dificuldade inerente aos problemas que podem ser resolvidos por meio de um computador.

- Problemas tratáveis *vs.* intratáveis.
- Dificuldade computacional (classes $\mathbf{P}$, $\mathbf{EXP}$ e $\mathbf{R}$).
- Máquinas não determinísticas e a classe $\mathbf{NP}$.
- Termos "difícil" e "completo".
- Questão $\mathbf{P} = \mathbf{NP}$?
- Reduções.

Algoritmos × Problemas

Complexidade de Algoritmos

É uma medida (quantidade) dos recursos necessários para um algoritmo efetuar sua computação.

Complexidade de Problemas

- A dificuldade computacional de um problema é a complexidade do melhor algoritmo que resolve o problema (nem sempre é o melhor algoritmo que conhecemos). Exemplo: o melhor algoritmo de ordenação por comparação consome tempo $\Theta(n \lg n)$ no pior caso.
- Complexidade computacional considera a classificação dos problemas conforme sua dificuldade.

Linguagens formais e problemas de decisão

Linguagem formal

- Uma linguagem formal L definida sobre um alfabeto Σ é um subconjunto de Σ^*.
- Ex.: Dado $\Sigma = \{0, 1\}$, $L = \{10, 11, 101, 111, 1011, 1101, \ldots\}$ é a linguagem da representação binária dos números primos.

Problema de decisão

- Função que associa o conjunto de instâncias I ao conjunto solução {**não**, **sim**} (conjunto {0, 1}).
- Um algoritmo A **aceita** uma instância x se $A(x) = 1$ e A **rejeita** x se $A(x) = 0$. A linguagem aceita por A é o conjunto $L = \{x : A(x) = 1\}$.
- Um algoritmo para o problema é **decidível** se para qualquer instância ele sempre termina em aceitação ou rejeição.

Codificação (*encodings*)

- Uma codifcação e de um conjunto de objetos S é uma associação de elementos de S a cadeias (*strings*) definidas sobre um alfabeto (com pelo menos dois elementos). Ex.: representação binária de números, tabela ASCII, etc.
- Podemos codificar um objeto composto como uma cadeia formada pela combinação das representações de suas partes constituintes. Ex.: vetores de inteiros, polígonos, grafos, funções, programas, etc.
- $e : I \rightarrow \{0, 1\}^*$ é a representação computacional de uma instância de um problema.
- A eficiência na resolução de um problema depende de como o problema é codificado (medimos a eficiência de um algoritmo em função do tamanho da entrada).

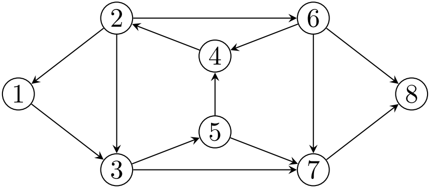
Classe $\mathbf{P}$

- Conjunto de problemas de decisão que podem ser resolvidos em tempo polinomial em uma máquina determinística.
- Problemas que podem ser resolvidos no pior caso em tempo $O(n^k)$, para alguma constante k.
- Problemas em $\mathbf{P}$ são chamados de **tratáveis**. Problemas que não estão em $\mathbf{P}$ são chamados de **intratáveis**.
- Exemplo de problema em $\mathbf{P}$:
  - Dado um conjunto S de n inteiros, verificar se um inteiro x pertence a este conjunto.
  - Usando a notação de linguagens, $\textsc{Busca} = \{\langle S, x\rangle : x \in S\}$.
  - $\langle\{3, 5, 2, 1\}, 2\rangle \in \textsc{Busca}$, $\langle\{3, 5, 2, 1\}, 4\rangle \notin \textsc{Busca}$.
  - Para mostrar que $\textsc{Busca} \in \mathbf{P}$, devemos apresentar um algoritmo polinomial (no tamanho da instância de entrada) para este problema.

Exercício

Mostre que $\textsc{Path}$ pertence a $\mathbf{P}$

Dado um grafo G e dois vértices s, t, existe caminho de s a t?.
$\textsc{Path} = \{\langle G, s, t\rangle : \text{existe um caminho de } s \text{ a } t\}$.



Exemplo: para o grafo G acima e vértices $s = 1$ e $t = 4$ o algoritmo deverá devolver **sim**. Se $s = 7$ e $t = 2$ o algoritmo deverá devolver **não**.

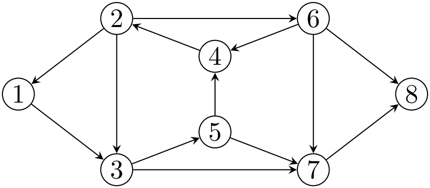
Classe $\mathbf{NP}$

- Conjunto de problemas de decisão que podem ser resolvidos em tempo polinomial em uma máquina não determinística.
- Alternativamente, é o conjunto de problemas que podem ser verificados em tempo polinomial usando uma máquina determinística.
  - Um **algoritmo verificador** A é um algoritmo que recebe como argumentos uma instância x do problema e um **certificado** y.
  - Um algoritmo A verifica a linguagem L se $\forall x \in L$, existe um y que pode ser usado para provar que $x \in L$. Além disso, $\forall x \notin L$ não pode haver um certificado provando que $x \in L$.
  - $L = \{x : \exists y$ tal que $A(x, y) = 1\}$.
  - Ex.: Determinar se um número é composto pertence a $\mathbf{NP}$, pois podemos definir o certificado y como sendo um divisor de x. Assim, $A(x, y) = 1$ se e somente se $y|x$, $y \neq 1$ e $y \neq x$.
- **IMPORTANTE:** $\mathbf{NP}$ significa polinomial não determinístico (não significa "não polinomial").

Exemplo

Mostre que $\textsc{Hamiltonian-Path}$ pertence a $\mathbf{NP}$

Dado um grafo G e dois vértices s, t, existe caminho hamiltoniano (caminho que passa por todos os vértices uma única vez) de s a t?.
$\textsc{Path} = \{\langle G, s, t\rangle : \text{existe um caminho hamiltoniano de } s \text{ a } t\}$.
Exemplo:



Termos completo e difícil

Completo

- Designa o conjunto de problemas “mais difíceis” dentro de uma determinada classe.
- Exemplo: O problema de determinar se um grafo possui um caminho Hamiltoniano pertence a $\mathbf{NP}$-**Completo**.

Difícil

- Designa o conjunto de problemas que são pelo menos tão difíceis quanto os problemas mais difíceis de uma determinada classe.
- Exemplo: O Problema do Caixeiro Viajante é $\mathbf{NP}$-**difícil**.
- Problemas em $\mathbf{NP}$-**difícil** não precisam ser problemas de decisão.

$\mathbf{P} = \mathbf{NP}$?

Proposição: $\mathbf{P} \subseteq \mathbf{NP}$

Considere qualquer problema $X \in \mathbf{P}$.

- Pela definição, existe um algoritmo polinomial $A(x)$ que resolve X.
- Certificado: $y = \varepsilon$ (símbolo representando entrada vazia), verificador $A(x, y) = A(x)$.

Questão aberta: $\mathbf{NP} \subseteq \mathbf{P}$?

- Prêmio de 1 milhão de dólares (7 *Millenium Prize Problems*).
- Maioria dos pesquisadores acreditam que $\mathbf{P} \neq \mathbf{NP}$.

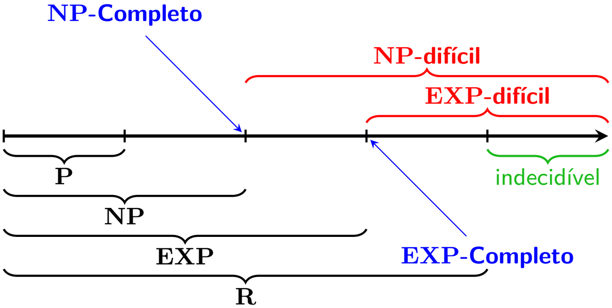
Classe $\mathbf{EXP}$

- Problemas de decisão que podem ser resolvidos em tempo exponencial em uma máquina determinística.
- Contém todos os problemas da classe $\mathbf{P}$, $\mathbf{NP}$ e outros que não estão em $\mathbf{P}$ (ainda não se sabe se $\mathbf{NP} = \mathbf{EXP}$).
- Problemas que não estão em $\mathbf{P}$
  - Podem ser resolvidos em tempo razoável apenas para instâncias pequenas.
  - Abordagens heurísticas ou aproximadas para lidar com o problema.
- Exemplo: Xadrez em tabuleiro $n \times n$ pertence à $\mathbf{EXP}$, mas não pertence à $\mathbf{P}$.

Classe $\mathbf{R}$ (recursivamente enumerável)

- Problemas que podem ser resolvidos em tempo finito.
- Turing demonstrou na década de 30 que existem problemas que não podem ser resolvidos por qualquer algoritmo.
- O mais famoso destes problemas é o problema da parada:
  - Dado um algoritmo qualquer e sua instância de entrada, este algoritmo irá eventualmente parar ou continuará executando em um "loop infinito"?
- Fato: Existem muito mais problemas que não podem ser resolvidos computacionalmente do que problemas que podem ser resolvidos computacionalmente.

Classes de problemas e suas dificuldades



Reduções

Conceito

- Uma redução é uma transformação de um problema em outro.
- Captura a noção informal de que "um problema seja pelo menos tão difícil quanto outro problema". Por exemplo, se um problema X pode ser resolvido usando um algoritmo para Y, X não é mais difícil do que Y, e dizemos que X se reduz a Y.
- Estamos interessados em redução em tempo polinomial, isto é, transformações de instâncias de X em instâncias de Y que possam ser feitas em tempo polinomial.
- Notação: $X \preceq_p Y$ (X é redutível em tempo polinomial a Y, ou Y é pelo menos tão difícil quanto X).

Exemplo

QUADRADO $\preceq_p$ MULTIPLICA

- Isto significa que podemos resolver o problema do quadrado de um número inteiro z usando o algoritmo para multiplicação de dois inteiros x e y.
- Para isto, pegamos a instância de entrada z para o problema QUADRADO e usamos $x = y = z$ como instância do problema MULTIPLICA.
- A resposta para o problema MULTIPLICA será a resposta para o problema QUADRADO.
- Assim, podemos afirmar que o problema do quadrado de um inteiro não é mais difícil do que o problema da multiplicação.

Mais alguns exemplos

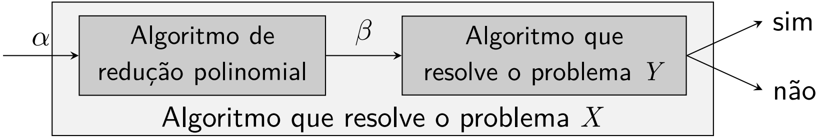
Expressões lineares $\preceq_p$ Expressões quadráticas

- Seja $X = \{ax + b = 0\}$ e $Y = \{a'x^2 + b'x + c' = 0\}$.
- Podemos resolver X, por meio de Y, fazendo com que a instância de entrada para Y seja $a' = 0$, $b' = a$ e $c' = b$.

Fibonacci $\preceq_p$ Potência de Matrizes

- Seja $X = \mathit{fib}(x)$ e $Y = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^n$
- Podemos resolver X, por meio de Y, fazendo com que a instância de entrada para Y seja $a = 1$, $b = 1$, $c = 1$, $d = 0$ e $n = x$.
- A solução para $\mathit{fib}(x)$ estará nas células $(1, 2)$ e $(2, 1)$ da matriz resultante.

Redução em tempo polinomial



- α é uma instância de X e β é uma instância de Y.
- A resposta do Algoritmo X para a instância α é sim se e somente se a resposta do Algoritmo Y para a instância β for sim.

Supondo que $X \preceq_p Y$

- Se Y pode ser resolvido em tempo polinomial, então X também pode ser resolvido em tempo polinomial.
- Se X não pode ser resolvido em tempo polinomial, então Y também não pode ser resolvido em tempo polinomial [contrapositiva].

Classe $\mathbf{NP}$-Completo

Definição

Um problema X é $\mathbf{NP}$-**Completo** quando:

1. $X \in \mathbf{NP}$;
2. $Y \preceq_p X$ para todo problema $Y \in \mathbf{NP}$.

Teorema

Suponha que X é um problema $\mathbf{NP}$-**Completo**. Então X pode ser resolvido em tempo polinomial se e somente se $\mathbf{P} = \mathbf{NP}$.

($\Leftarrow$) Se $\mathbf{P} = \mathbf{NP}$ então X pode ser resolvido em tempo polinomial, pois $X \in \mathbf{NP}$ (item 1 da Definição);

($\Rightarrow$) Suponha que X pode ser resolvido em tempo polinomial.

- Seja Y um problema qualquer em $\mathbf{NP}$. Como $Y \preceq_p X$, podemos resolver Y em tempo polinomial. Isto implica que $\mathbf{NP} \subseteq \mathbf{P}$.
- Sabemos que $\mathbf{P} \subseteq \mathbf{NP}$ (Por quê?). Portanto, $\mathbf{P} = \mathbf{NP}$.

Um primeiro problema **NP-Completo**

Para mostrar que existe um problema **NP-Completo**, seria necessário mostrar que um problema deve ser capaz de representar/codificar qualquer problema em **NP**.

- Em 1971, Cook e Levin mostraram de maneira independente como fazer isto para quaisquer problemas em **NP**.
- CIRTUIT SATISFIABILITY é **NP-Completo**.
- Representação de problemas por meio e circuitos (ver Cormen capítulo 34.3 ou Kleinberg & Tardos capítulo 8.4).

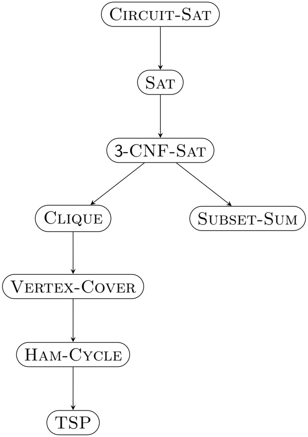
Mostrando que há outros problemas em $\mathbf{NP}$-**Completo**

Estratégia geral

1. Mostre que $X \in \mathbf{NP}$.
2. Escolha um problema Y que é $\mathbf{NP}$-**Completo**.
3. Mostre que $Y \preceq_p X$.

Note que o fato de $Y \in \mathbf{NP}$-**Completo** implica $Z \preceq_p Y$ para qualquer problema $Z \in \mathbf{NP}$ e, pela redução $Y \preceq_p X$, segue que $Z \preceq_p X$.

Algumas reduções apresentadas no Cormen



Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. **Introduction to Algorithms**, Third Edition. The MIT Press. Chapter 34.
- Kleinberg J., and Tardos E. **Algorithm Design**. 2005. Pearson. Chapter 8.
- Cook, Stephen. *The complexity of theorem proving procedures*. Proceedings of the Third Annual ACM Symposium on Theory of Computing [S.l.: s.n.], 1971. pp. 151–158.