MAC 5711 - Análise de Algoritmos

Departamento de Ciência da Computação
Primeiro semestre de 2022

Complexidade computacional

Por que alguns problemas parecem ser (computacionalmente) mais difíceis do que outros? Como podemos medir ou comprovar este diferente e aparentemente intrínseco grau de dificuldade? Quais problemas podem ser resolvidos por um algoritmo? Quais não podem? Por quê? Quais problemas podem ser resolvidos por um algoritmo eficiente? Quais não podem? Por quê?

Em complexidade computacional estamos interessados em questões deste tipo.

1. Problemas e algoritmos

Um *problema abstrato de decisão* consiste de um conjunto I de instâncias e uma função $f : I \to \{0, 1\}$ (ou $\{\text{SIM}, \text{NÃO}\}$).

Exemplo: Problema CIRCUITO HAMILTONIANO.
O conjunto I neste caso consiste do conjunto de todos os grafos.
Para todo G em I, $f(G) = 1$ se G tem um circuito hamiltoniano (ou seja, um circuito que passa por todos os vértices de G) e $f(G) = 0$ caso contrário. O valor de $f(X)$ é a *resposta* do problema para a instância X em I.

Para um problema abstrato ser resolvido, é necessário que as instâncias do mesmo sejam convenientemente codificadas. Um *problema concreto de decisão* é um problema abstrato de decisão em que as instâncias estão codificadas convenientemente em um alfabeto finito Σ. Vamos supor que $\Sigma = \{0, 1\}$, embora para os nossos propósitos qualquer Σ com pelo menos dois elementos sirva. Daqui para frente, assumimos que todos os problemas são concretos.

Nessas notas, um *algoritmo* é um programa escrito em uma linguagem como `C` ou `Pascal`, onde estabelecemos que o custo de uma operação é proporcional ao número de bits dos operandos e o custo de um acesso à memória é proporcional ao número de bits dos operandos. (Isto é polinomialmente equivalente à *máquina de Turing*.)

Um algoritmo *resolve* ou *decide* um problema de decisão (I, f) se, para cada X em I, o algoritmo encontra a resposta para X, isto é, o algoritmo calcula $f(X)$.

Existem problemas para os quais não existe um algoritmo — são os ditos problemas *indecidíveis*. Um exemplo de problema indecidível é o *problema da parada*: dado um programa e um arquivo de entrada para este programa, decidir se tal programa pára ou não quando executado com este arquivo de entrada. Não existe um algoritmo que resolva o problema da parada.

2. A classe P

Um algoritmo resolve (ou decide) um problema de decisão (I, f) *em tempo* $O(T(n))$ se, para cada n em $\mathbf{N}$ e cada X em I com $|X| = n$, o algoritmo encontra a resposta $f(X)$ para X em tempo $O(T(n))$.

Um problema de decisão é *solúvel em tempo polinomial* se existe algum k para o qual existe um algoritmo que resolve o problema em tempo $O(n^k)$. Tais problemas são ditos *tratáveis*.

A *classe de complexidade* P é o conjunto dos problemas de decisão tratáveis, isto é, que são solúveis em tempo polinomial.

3. As classes NP e coNP

Considere o problema CIRCUITO HAMILTONIANO. Se a resposta para uma instância G for SIM, então existe um circuito hamiltoniano C no grafo. Neste caso, dados G e C, é possível verificar em tempo polinomial em $|G|$ se C é de fato um circuito hamiltoniano em G ou não. Ou seja, temos como *certificar eficientemente* que a resposta SIM está correta. Será que conseguimos também certificar eficientemente a resposta NÃO? Surpreendentemente, não se conhece nenhum método de certificação eficiente para a resposta NÃO no caso do problema CIRCUITO HAMILTONIANO.

A *classe de complexidade* NP consiste nos problemas para os quais a resposta SIM pode ser certificada e verificada eficientemente, ou seja, em tempo polinomial. Mais precisamente, um problema de decisão está em NP se existe um algoritmo A tal que

(1) para qualquer instância X do problema com resposta SIM, existe um Y em Σ^* tal que $A(X, Y)$ devolve SIM;
(2) para qualquer instância X do problema com resposta NÃO, para todo Y em Σ^*, $A(X, Y)$ devolve NÃO;
(3) A consome tempo polinomial em $|X|$.

Y é chamado de *certificado para* SIM da instância X.

O *complemento de um problema de decisão* Q é o problema de decisão Q' obtido invertendo-se o SIM e o NÃO na resposta ao problema Q.

Exemplo: O complemento do problema CIRCUITO HAMILTONIANO é o problema cuja resposta é SIM sempre que o grafo não tem um circuito hamiltoniano e NÃO caso contrário. É o problema NÃO-HAMILTONIANO.

A *classe de complexidade* coNP consiste nos problemas de decisão que são complementos de problemas de decisão em NP. Ou seja, problemas para os quais existe um certificado (curto) para a resposta NÃO.

O problema CIRCUITO HAMILTONIANO é um exemplo de problema em NP enquanto que o problema NÃO-HAMILTONIANO é um exemplo de problema em coNP. Note ainda que P $\subseteq$ NP $\cap$ coNP.

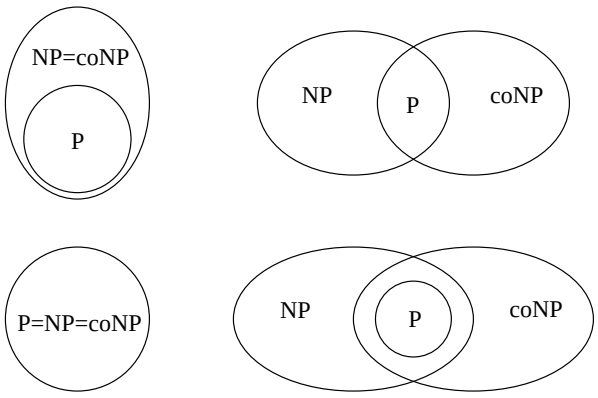


FIGURA 1. Possibilidades de como é a relação de inclusão entre as classes P, NP e coNP.

4. REDUÇÃO DE PROBLEMAS

Um problema de decisão Q pode ser *reduzido em tempo polinomial* a outro problema de decisão Q' se existe um algoritmo A que, dada uma instância X de Q, $A(X)$ é uma instância de Q' tal que

(1) X e $A(X)$ têm a mesma resposta;
(2) A é polinomial em $|X|$.

A definição acima implica que, se soubermos resolver Q' em tempo polinomial, também sabemos resolver Q em tempo polinomial. O problema Q não é mais difícil do que Q'. Notação: $Q \leq_P Q'$.

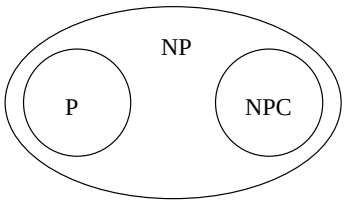
Lema 1. *Se $Q_1 \leq_p Q_2$ e $Q_2 \in P$ então $Q_1 \in P$.*

Um problema Q é *NP-completo* se

(1) $Q \in$ NP;
(2) $Q' \leq_p Q$ para todo Q' em NP.

Um problema é *NP-difícil* se $Q' \leq_p Q$ para todo Q' em NP.

Conjectura 1. *A figura abaixo está correta, onde NPC denota a classe dos problemas NP-completos.*



Teorema 1. *Se algum problema NP-completo está em P, então P = NP. Se algum problema NP-completo não pode ser resolvido em tempo polinomial, então nenhum problema NP-completo pode ser resolvido em tempo polinomial.*

Existem tantos problemas NP-completos, aparecendo em diferentes áreas e há tanto tempo, que não se acredita que P = NP.

O livro de Garey e Johnson (*Computers and Intractability: a Guide to the Theory of NP-Completeness*, QA810 G229c) é uma referência padrão nesse assunto. Nele você encontra uma lista de problemas NP-completos, com referências sobre as reduções envolvidas, etc.

5. Problema da satisfatibilidade

Seja $X := \{x_1, \ldots, x_n\}$ um conjunto de variáveis booleanas. Uma *atribuição* a X é uma função $t : X \to \{V, F\}$. Para x em X, x e $\bar{x}$ são *literais* sobre X. Um literal $\tilde{x}$ é verdadeiro sob t se $\tilde{x} = x$ para algum x em X e $t(x) = V$ ou $\tilde{x} = \bar{x}$ para algum x em X e $t(x) = F$.

Uma *cláusula* sobre X é um conjunto de literais sobre X. Ela representa a disjunção (ou) dos literais e é *satisfeita* por uma atribuição se e somente se pelo menos um dos literais é verdadeiro sob a atribuição. Uma coleção $\mathcal{C}$ de cláusulas é *satisfatível* se e somente se existe uma atribuição que satisfaça simultaneamente todas as cláusulas de $\mathcal{C}$. Neste caso, dizemos que tal atribuição satisfaz $\mathcal{C}$.

Problema SAT: Dados X e $\mathcal{C}$, decidir se existe uma atribuição a X que satisfaça $\mathcal{C}$.

Cook demonstrou, em 1971, que SAT é NP-completo. A demonstração de que SAT está em NP é trivial. É a segunda parte da demonstração, que consiste da redução de um problema arbitrário em NP a SAT, que é a parte difícil da demonstração. Considere um problema arbitrário em NP. A redução consiste em, para uma instância X do problema Q, construir em tempo polinomial uma coleção de cláusulas $\mathcal{C}$, instância do SAT, que é satisfatível se e somente se a resposta para X é SIM. Grosseiramente falando, considere uma instância X de Q cuja resposta seja SIM. Para tal instância, por definição, existe um certificado Y e um algoritmo A, polinomial, que recebe X e Y como entrada e produz SIM. As cláusulas em $\mathcal{C}$ são derivadas da execução de A para a entrada (X, Y) e verificam se a resposta produzida pela execução é SIM.

Essa foi a primeira demonstração de NP-completude. A partir desta, demonstrou-se que muitos outros problemas são NP-completos. O lema abaixo mostra como isso pode ser feito de uma maneira mais direta.

Lema 2. *Se $Q' \leq_p Q$ para algum Q' em NPC, então Q é NP-difícil. Ademais, se $Q \in$ NP, então $Q \in$ NPC.*

O problema 3-SAT consiste na restrição de SAT a instâncias em que toda cláusula tem exatamente três literais. Essa restrição do SAT é muito utilizada na demonstração de que outros problemas são NP-completos. A seguir mostramos que ela é tão difícil quanto a versão geral do SAT.

Teorema 2. *3-SAT é NP-completo.*

Prova: A primeira parte da prova é mostrar que 3-SAT está em NP. Para tanto, basta descrever um certificado para instâncias de resposta SIM e um algoritmo verificador. Uma instância é um conjunto de cláusulas $\mathcal{C}$ sobre um conjunto de variáveis X. O certificado é evidente: basta tomar uma atribuição de valores a X que satisfaça $\mathcal{C}$. Tal certificado existe, claro, apenas se $\mathcal{C}$ for satisfatível, ou seja, se X, $\mathcal{C}$ for uma instância de resposta SIM. Além disso, é fácil pensar em um algoritmo que leia X, $\mathcal{C}$ e uma

atribuição de valores para X e que imprima SIM se e somente se a atribuição satisfizer $\mathcal{C}$. Ademais, tal algoritmo pode ser implementado de forma a consumir tempo polinomial em $|X|$ e $|\mathcal{C}|$.

A segunda parte da prova é a mais difícil. Trata-se da redução. Vamos mostrar que SAT $\leq_p$ 3-SAT. Seja X, $\mathcal{C}$ uma instância do SAT. Vamos construir uma instância X', $\mathcal{C}'$ do 3-SAT tal que $\mathcal{C}$ é satisfatível se e somente se $\mathcal{C}'$ é satisfatível.

Denotemos as cláusulas do conjunto $\mathcal{C}$ por $C_1, \ldots, C_m$. Para cada C_i, definimos um conjunto de variáveis extras X_i' e um conjunto de cláusulas $\mathcal{C}_i'$, todas com três literais, sobre as variáveis $X \cup X_i'$, que preserva uma forte relação com C_i: (1) qualquer atribuição a $X \cup X_i'$ que satisfaça $\mathcal{C}_i'$, se restrita a X, satisfaz C_i, e (2) qualquer atribuição a X que satisfaça C_i pode ser estendida a uma atribuição a $X \cup X_i'$ que satisfaz $\mathcal{C}_i'$. Feito isso, basta definir $X' = X \cup X_1' \cup \cdots \cup X_m'$ e $\mathcal{C}' = \mathcal{C}_1' \cup \cdots \cup \mathcal{C}_m'$.

Como definimos X_i' e $\mathcal{C}_i'$? Consideramos quatro casos separadamente. Se $|C_i| = 1$, denotemos por $\tilde{x}$ o único literal em C_i. Então tome $X_i' = \{y_1^i, y_2^i\}$ e $\mathcal{C}_i' = \{\{\tilde{x}, y_1^i, y_2^i\}, \{\tilde{x}, y_1^i, \overline{y}_2^i\}, \{\tilde{x}, \overline{y}_1^i, y_2^i\}, \{\tilde{x}, \overline{y}_1^i, \overline{y}_2^i\}\}$. Note que (1) e (2) valem para tal escolha de X_i' e $\mathcal{C}_i'$: qualquer atribuição que satisfaça $\mathcal{C}_i'$ deve fazer $\tilde{x}$ verdadeiro e é trivial estender qualquer atribuição que satisfaça C_i para uma atribuição que satisfaça $\mathcal{C}_i'$ (qualquer extensão serve).

Se $|C_i| = 2$, ou seja, $C_i = \{\tilde{x}, \tilde{z}\}$, para x e z em X, então tome $X_i' = \{y^i\}$ e $\mathcal{C}_i' = \{\{x, z, y^i\}, \{x, z, \overline{y}^i\}\}$. Verifique que (1) e (2) valem. Se $|C_i| = 3$, tome $X_i' = \emptyset$ e $\mathcal{C}_i' = \{C_i\}$.

Se $|C_i| = k \geq 4$, ou seja, $C_i = \{\tilde{x}_1, \ldots, \tilde{x}_k\}$ com $k \geq 4$, então $X_i' = \{y_j^i : j = 1, \ldots, k-3\}$ e $\mathcal{C}_i' = \{\{\tilde{x}_1, \tilde{x}_2, y_1^i\}\} \cup \bigcup \{\{\overline{y}_j^i, \tilde{x}_{j+2}, y_{j+1}^i\} : j = 1, \ldots, k-4\} \cup \{\{\overline{y}_{k-3}^i, \tilde{x}_{k-1}, \tilde{x}_k\}\}$. Verifique que, novamente, (1) e (2) valem.

Finalmente, observe que X' e $\mathcal{C}'$ podem ser obtidos de X e $\mathcal{C}$ em tempo polinomial em $|X|$ e $|\mathcal{C}|$. $\square$