MAC5727 – Algoritmos de Aproximação

Departamento de Ciência da Computação – IME/USP
Primeiro Semestre de 2000

Um pouco de Teoria de Complexidade de Algoritmos

1. Problemas, instâncias e representações de instâncias

A Teoria de Complexidade se preocupa em categorizar os problemas de acordo com a sua dificuldade. Para tanto, precisamos formalizar a definição de *problema*.

Em uma primeira fase, esta teoria se aplica apenas aos chamados *problemas de decisão*. Um *problema de decisão* é um problema cuja resposta para cada instância é ou *sim* ou *não*.

Um exemplo de problema de decisão é o *Problema do Circuito Hamiltoniano* (HC): dado um grafo G, determinar se G é ou não Hamiltoniano. (Um grafo é Hamiltoniano se tem um circuito Hamiltoniano—um circuito que contém todos os vértices do grafo.)

Podemos representar um problema de decisão abstrato pelo seu conjunto de instâncias $\mathcal{I}$ (as possíveis entradas do problema) e uma função (resposta) $f : \mathcal{I} \to \{0, 1\}$ (0 para *sim*, 1 para *não*). O problema seria então: dada uma instância $I \in \mathcal{I}$, determinar $f(I)$.

Quem é então o conjunto $\mathcal{I}$ para o HC? É o conjunto de possíveis entradas do problema, ou seja, o conjunto de todos os grafos.

Na verdade, o tipo de análise feita em complexidade requer um pouco mais de precisão nesta definição. Precisamos nos preocupar em *como* a instância (entrada) é dada, pois estaremos falando de tempo de execução em função do *tamanho* da instância. Neste caso então necessitamos de uma definição ainda mais precisa de um problema.

Um *problema concreto de decisão* é um conjunto $\mathcal{R}$ de **representações** das instâncias do problema e uma função $f : \mathcal{R} \to \{0, 1\}$.

No caso do HC por exemplo, as instâncias são grafos. Um grafo pode ser representado por sua matriz de adjacência ou por sua matriz de incidência ou por listas de adjacência... Instâncias portanto podem ter mais do que uma representação. Em um problema concreto de decisão já decidimos que representação usaremos para as instâncias.

Neste texto, vamos pensar na representação de uma instância da seguinte forma. Considere um algoritmo que você já tenha implementado e que faz leitura de um arquivo texto. A representação de uma instância é o conteúdo do arquivo de entrada que representa aquela instância.

2. Algoritmos e algoritmos polinomiais — a classe **P** de problemas

O *tamanho da representação de uma instância* é neste contexto o número de *bytes* do arquivo de entrada correspondente (número de caracteres se você preferir).

Um algoritmo resolve um problema concreto de decisão se, para toda representação R em $\mathcal{R}$ de uma instância do problema, o algoritmo determina $f(R)$.

Um algoritmo que resolve um problema concreto de decisão é *polinomial* se existe um polinômio $p(t)$ tal que, para todo inteiro $t > 0$ e para toda representação R de tamanho t, o número de instruções executadas pelo algoritmo para determinar $f(R)$ é no máximo $p(t)$.

Denotamos por **P** a classe de problemas concretos de decisão para os quais existe um algoritmo polinomial.

Esperamos que nesse ponto você tenha entendido a diferença entre problemas **abstratos** de decisão e problemas **concretos** de decisão, bem como a diferença entre **instância** e **representação de instância** e a necessidade desta diferenciação.

De agora em diante, vamos abusar da notação e nos referir apenas a **problemas de decisão** e a **instâncias**. Fica implícito que estamos falando de problemas **concretos** de decisão e de **representações de instâncias**.

3. Certificados e algoritmos de verificação — a classe NP de problemas

Para definir a classe **NP** de problemas precisamos levantar uma questão em princípio não muito natural.

Considere um problema de decisão Π dado por um conjunto $\mathcal{I}$ de intâncias e por uma função $f : \mathcal{I} \to \{0, 1\}$. (Lembre-se que 0 significa *sim* e 1 significa *não*.) Denotemos por $\mathcal{I}_0$ o conjunto de instâncias I para as quais $f(I) = 0$.

Digamos que alguém nos apresenta uma instância de Π e diz que esta está em $\mathcal{I}_0$. Como podemos nos certificar disso?

Mais concretamente, olhemos para o Problema do Circuito Hamiltoniano e digamos que alguém nos apresente um particular grafo e diga que este grafo é Hamiltoniano. O que poderia ser uma prova indiscutível para esta afirmação? [Pense por alguns segundos.] Oras, um circuito Hamiltoniano neste grafo!

Dizemos que C é um *certificado* para uma instância I em $\mathcal{I}_0$ se C pode ser usado por um *algoritmo de verificação* para comprovar que I está de fato em $\mathcal{I}_0$.

Um certificado para uma instância I é *curto* se tem tamanho (o número de bytes necessários para representá-lo) polinomial no tamanho de I.

Um *algoritmo de verificação para* Π consiste em um algoritmo que tem como entrada um par (I, C) tal que I é uma instância de Π e C é um (candidato a) certificado curto para I, e como saída, *sim* se de fato I está em $\mathcal{I}_0$, *não* caso contrário.

Denotamos por **NP** a classe de problemas concretos de decisão para os quais existe um algoritmo de verificação polinomial.

Em outras palavras, um problema está em **NP** se, para cada instância cuja resposta é *sim*, existe um certificado curto que pode ser usado para verificar em tempo polinomial que de fato a resposta para essa instância é *sim*.

Vamos voltar ao HC (Hospital das Clínicas??). Como dissemos, um circuito Hamiltoniano em G pode servir de certificado para provar que G é Hamiltoniano. Podemos representar um circuito em G por uma seqüência de vértices de G. Desta forma, um circuito Hamiltoniano em G torna-se um certificado curto: o número de *bytes* necessários para representá-lo é limitado por um polinômio no número de *bytes* necessários para representar o grafo G. Além disso, é fácil imaginar um algoritmo polinomial de verificação para o HC: o algoritmo tem como entrada um grafo G e uma seqüência de vértices de G e verifica se a seqüência dada representa ou não um circuito Hamiltoniano em G. Isso tudo significa que HC está em **NP**.

Note que todo problema em **P** está em **NP**, ou seja, $\mathbf{P} \subseteq \mathbf{NP}$. [Certifique-se que você acredita nessa afirmação.]

4. Problemas NP-completos, NP-difíceis e reduções polinomiais

Uma ferramenta importante na categorização de problemas, introduzida por Karp [4] em 1972, são as chamadas reduções. São através delas que conseguimos "comparar" problemas.

Dados dois problemas de decisão A e B, uma redução de A para B é um algoritmo X que converte instâncias do problema A em instâncias do problema B com a seguinte característica: uma instância I_A do problema A é tal que $f_A(I_A) = 0$ se e somente se a instância $I_B := X(I_A)$ é tal que $f_B(I_B) = 0$.

A redução é polinomial se o algoritmo X é polinomial.

Um problema Π é **NP**-*completo* se Π está em **NP** e, para cada problema A em **NP**, existe uma redução polinomial de A para Π.

Neste ponto, você deve ser capaz de acreditar no seguinte

Theorem 4.1. *Se existe um algoritmo polinomial para um problema* **NP**-*completo, então existe um algoritmo polinomial para todo problema em* **NP** *(em outras palavras,* **P=NP***).*

[Certifique-se que você acredita neste teorema.]

Neste sentido os problema **NP**-completos são os problema mais difíceis da classe **NP**.

Mas será que existe algum problema **NP**-completo?? Sim, existem muitos deles. Cook [1] em 1971 mostrou a existência de um problema **NP**-completo—o *Problema da Satisfabilidade*: dada uma fórmula booleana em CNF (*conjuntive normal form*), determinar se esta fórmula é ou não satisfatível, ou seja, se existe ou não uma atribuição de valores às suas variável que a faça verdadeira. A demostração de Cook consiste basicamente de uma redução polinomial deste problema para um problema arbitrário em **NP**. Apesar da idéia parecer simples, as tecnicalidades tornam esta demonstração complicada.

No momento em que temos um primeiro problema **NP**-completo, as coisas se tornam mais simples: para demonstrar que um outro problema A em **NP** é **NP**-completo é suficiente fazer uma redução polinomial do primeiro para A. E assim sucessivamente. Atualmente conhecemos centenas de problemas **NP**-completos (ou seja, foram feitas centenas destas reduções). Portanto, para demonstrar que um novo problema em **NP** é **NP**-completo, podemos escolher um problema dentre estas centenas para fazer uma redução polinomial.

Para terminar, vamos dizer o que é um problema **NP**-*difícil*. Um problema Π (não necessariamente de decisão) é **NP**-difícil se, para todo problema A em **NP**, existe uma redução polinomial de A para Π.

Para mostrar que um problema Π é **NP**-difícil basta conseguirmos uma redução polinomial de um problema **NP**-completo para Π.

No curso de algoritmos de aproximação estaremos estudando em via de regra *problemas de otimização* **NP**-difíceis.

No Capítulo 36 de [2] você encontra esse tema discutido com um pouco mais de detalhes. Para uma versão mais completa e profunda, veja [3] ou [5].

Referências

[1] S. Cook, "The Complexity of Theorem Proving Procedures", *Proc. 3rd. ACM Symposium on Theory of Computing*, 151–158 (1971).

[2] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press & McGraw-Hill (1992).

[3] M.R. Garey and D.S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, Freman, New York (1979).

[4] R. Karp, "Reducibility among Combinatorial Problems", *Complexity of Computer Computations* (R. Miller and J. Thatcher, eds.), 85–104. Plenum Press, New York (1972).

[5] C.H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Massachusetts (1994).