Teoria da Complexidade Computacional

25 de novembro de 2011

Enquanto a teoria de análise de algoritmos estuda a análise de complexidade de algoritmos, a teoria da complexidade estuda a classificação de problemas com base na complexidade dos algoritmos que os resolvam.

Um algoritmo é denominado *polinomial* (*exponencial*, respectivamente) quando sua complexidade é uma função polinomial (exponencial, respectivamente) no tamanho da entrada. Um algoritmo é eficiente quando sua complexidade é um polinômio no tamanho da entrada. Exemplos de algoritmos polinomiais (função de complexidade é $O(p(n))$, onde $p(n)$ é um polinômio) são:

1. pesquisa binária - $O(\log n)$;
2. pesquisa seqüencial - $O(n)$;
3. ordenação por inserção - $O(n^2)$;
4. multiplicação de matrizes - $O(n^3)$;

Exemplo de algoritmos exponenciais (função de complexidade é $O(c^n)$, $c > 1$) são:

1. caixeiro viajante - $O(n!)$. Pode ser reduzido para $O(n^2 2^n)$ através de programação dinâmica mas complexidade de espaço é de $O(n2^n)$;

Algoritmos com complexidade de tempo não-polinomial demandam tal quantidade de tempo para executar que mesmo problemas de tamanho pequeno a moderado não podem ser resolvidos. Garey e Johnson classificam da seguinte forma os problemas, segundo sua complexidade:

1. **Indecidível:** são problemas tão difíceis que nenhum algoritmo pode ser dado para resolvê-los. Tome, como exemplo, o problema da parada: Turing provou que, dado um programa de computador e uma entrada qualquer para esse programa, não existe algoritmo que possa decidir se o programa, quando aplicado àquela entrada, finalizará em uma quantidade finita de passos;

2. **Intratável:** problemas decidíveis porém difíceis para os quais possivelmente não existe algoritmo que os resolvam em tempo polinomial;

3. **Tratável:** problemas para os quais existe algoritmo que os resolvam em tempo polinomial.

A partir deste ponto, concentraremos a nossa atenção na questão de determinar se um problema é tratável/intratável. Precisaremos introduzir mais um pouco de teoria, primeiramente classificando os problemas conforme a resposta esperada. Os problemas podem ser classificados em:

1. **Problemas de decisão:** consiste na verificação (decisão) da veracidade ou não de determinada questão para o problema (resposta SIM ou NÃO). Verificação da existência de determinada solução.
   Exemplo (Problema do Caixeiro Viajante): dadas n cidades e distância c_{ij} entre duas cidades i e j e um inteiro não negativo k, existe um percurso do caixeiro viajante de comprimento menor ou igual a k?
   Exemplo (Problema do Caminho): dado um grafo $G = (V, E)$, orientado e ponderado nas arestas, dois vértices $u, v \in V$ e um inteiro não negativo k, existe um caminho em G entre u e v com comprimento menor ou igual a k? Então, dada uma instância do problema, a resposta será "sim" se houver tal caminho, ou "não", caso contrário.

2. **Problemas de localização:** consiste na verificação da existência e identificação (localização) de uma solução (segundo algum critério) para o problema.
   Exemplo (Problema do Caixeiro Viajante): dadas n cidades e distância c_{ij} entre duas cidades i e j e um inteiro não negativo k, o problema consiste em localizar, caso exista, um percurso do caixeiro viajante com custo menor ou igual a k. A resposta seria uma seqüência de vértices $u, \ldots, v$.
   Exemplo (Problema do Caminho): dado um grafo $G = (V, E)$, orientado e ponderado nas arestas, dois vértices $u, v \in V$ e um inteiro não

negativo k. O problema consiste em localizar, caso exista, um caminho com comprimento menor ou igual a k. A resposta seria uma seqüência de vértices $u, \ldots, v$.

3. **Problemas de otimização:** consiste na verificação da existência e identificação da melhor (otimização) solução possível, dentre as soluções factíveis para o problema.
   Exemplo (Problema do Caixeiro Viajante): dadas n cidades e distância c_{ij} entre duas cidades i e j, o problema consiste em obter um percurso do caixeiro viajante de menor custo possível.
   Exemplo (Problema do Caminho): dado um grafo $G = (V, E)$, orientado e ponderado nas arestas, dois vértices $u, v \in V$, o problema consiste em encontrar um caminho de menor comprimento possível.

Observe que um problema de decisão é a forma mais simples de um problema. Assim, podemos afirmar que um problema de decisão apresenta dificuldade não maior que de um problema de localização, e este, por sua vez, apresenta dificuldade não maior que de um problema de otimização. Sendo o problema de decisão normalmente mais fácil que os demais, logo, toda a teoria da complexidade baseia-se em problemas de decisão, sem perda de generalidade.

Todo problema de otimização (ou localização) pode ser transformado em uma versão de decisão do problema, de tal forma que a versão de decisão possa ser resolvida em tempo polinomial se a versão de otimização (ou localização) também puder; ou seja, a versão de decisão não pode ser mais difícil de resolver do que a versão de otimização (ou localização). Por exemplo, dada uma instância G, u, v para o problema do caminho mínimo, seja $v_1, v_2, \ldots, v_\ell$, com $v_1 = u$ e $v_\ell = v$ um caminho mínimo obtido por meio de um algoritmo em tempo polinomial. De posse dessa resposta, podemos fornecer uma resposta em tempo polinomial (neste caso, constante) para a versão de decisão: se $k \geq \ell$ a resposta é *sim*; caso contrário, a resposta é *não*. A sequência de vértices $v_1, v_2, \ldots, v_\ell$ é dita ser um *certificado* para a resposta *sim* do problema.

A seguir, classificaremos os problemas segundo o tempo necessário para: i) encontrar uma solução e ii) para verificar se uma resposta fornecida é realmente uma solução para o problema. As classes mais estudadas nesses dois quesitos, respectivamente, são denominadas P e NP.

A **classe** P consiste nos problemas que podem ser resolvidos em tempo polinomial por algum algoritmo determínistico. Alguns exemplos: or-

denação, caminho mínimo em grafos e fluxo máximo em redes. Observe que se os algoritmos conhecidos de um dado problema A são todos exponenciais, não necessariamente $A \notin P$.

A **classe** NP consiste nos problemas que são "verificáveis" em tempo polinomial. Para estes problemas são conhecidos algoritmos não-determinísticos polinomiais, ou seja, o algoritmo gera uma solução candidata ao problema e verifica sua viabilidade em tempo polinomial. Alguns exemplos: problema do caminho hamiltoniano, cliques em grafos, conjunto independente em grafos e problema da mochila.

Se um problema é da classe P então também é da classe NP, pois um algoritmo polinomial determinístico usado para resolver o problema pode ser utilizado para gerar uma solução correta. Portanto, podemos concluir que $P \subseteq NP$. A grande questão da Teoria da Complexidade é: $P = NP$? Ou seja, $NP \subset P$, o que nos diria que todos os problemas em NP são tratáveis? Este é o problema em aberto mais conhecido e importante em Computação.

0.1 NP-Completude

Em 1971, Stephen Cook formulou a seguinte questão: "Existirá algum problema em NP tal que, sendo provada sua pertinência a P, implicaria que $P = NP$?" Cook provou tal problema ser o SAT (*"satisfiability"* ou satisfabilidade). Devido a importância do fato, tal problema foi dito ser NP-Completo.

Um problema de decisão $A \in NP$ é NP-Completo se todos os outros problemas de NP se transformam polinomialmente em A. Assim, se existir um algoritmo (determinístico) polinomial para a resolução de algum problema NP-Completo, então todos os problemas da classe NP também poderão ser resolvidos em tempo polinomial. Assim, seria possível mostrar que $P = NP$. Há fortes evidências de que isso não é possível.

Um problema A pertence a classe NP-Difícil se existe um problema $B \in NP$-Completo que pode ser transformado em A em tempo polinomial. Portanto, a única diferença com relação a classe NP-Completo é que A não é provado ser NP.

Apenas problemas de decisão podem ser NP-Completo. Problemas de otimização podem ser NP-difícil, mas geralmente, se Π_1 é um problema de decisão e Π_2 um problema de otimização, então é bem possível que $\Pi_1 \propto \Pi_2$. A dificuldade de um problema NP-difícil não é menor do que a dificuldade de um problema NP-Completo. $\Pi_1 \propto \Pi_2$ significa que Π_1 é polinomialmente

transformável em Π_2 (ou redutível). Por exemplo, o conhecido problema de parada (*"halting problem"*) de Turing consiste em determinar, para um algoritmo determinista qualquer A com entrada de dados E, se o algoritmo A termina (ou entra em *loop* infinito). Este problema é indecidível, pois até agora não existe algoritmo de qualquer complexidade para resolvê-lo. Mas $SAT \propto$ problema de parada. Considere o algoritmo A cuja entrada é uma expressão booleana na forma normal conjuntiva com n variáveis. Basta tentar 2^n possibilidades e verificar se E é satisfatível. Se for A pára; senão, entra em *loop*. Logo, o problema de parada é NP-difícil, mas não é NP-Completo.

Para exemplificar uma prova de NP-Completude, a seguir mostramos que o problema do Caixeiro Viajante é NP-Completo.

0.2 Caixeiro Viajante é NP-Completo

Um ciclo hamiltoniano em um grafo é um ciclo simples (que passa por todos os vértices uma única vez). O problema de encontrar um ciclo hamiltoniano (PCH) é um caso especial do problema do caixeiro viajante (PCV), no qual cada aresta (u, v) possui distância c_{uv}.

O PCH foi um dos primeiros problemas a ser provado ser NP-Completo. Vamos mostrar que o $PCV \propto CH$. Primeiro, mostraremos que o problema está em NP. Dada uma instância do PCV, esta deve ser verificada em tempo polinomial. Usando a seqüência de n vértices do circuito, o algoritmo de verificação confirma que a seqüência contém cada vértice exatamente uma vez, totaliza os custos de arestas e verifica se a soma é no máximo k. Esse processo certamente pode ser feito em tempo polinomial.

Para provar que PCV é NP-difícil, apresentamos uma redução polinomial do PCH para o PCV. Seja $G = (V, E)$ uma instância do PCH. Construímos uma instância (em tempo polinomial) do PCV formalizando o grafo completo $G' = (V, E')$, onde $E' = \{(i, j) : i, j \in V \text{ e } i \neq j\}$ onde a função de custo c é

$$c(i,j) = \begin{cases} 0 \text{ se } (i,j) \in E, \\ 1 \text{ se } (i,j) \neq E. \end{cases} \tag{1}$$

Agora, mostraremos que o grafo G tem um ciclo hamiltoniano se e somente se o grafo G' tem um circuito de custo máximo 0. Suponha que o grafo G tem um ciclo hamiltoniano h. Cada aresta em h pertence a E e, portanto, tem custo 0 em G'. Desse modo, h é um circuito em G' com custo 0. Reciprocamente, suponhamos que o grafo G' tem um circuito h' de custo

máximo 0. Tendo em vista que os custos das arestas em E' são 0 e 1, o custo da viagem h' é exatamente 0, e cada aresta no circuito deve ter custo 0. Então, h' contém apenas arestas em E. Concluímos que h' é um ciclo hamiltoniano no grafo G.

As técnicas usadas para provar resultados de NP-Completude variam como os próprios problemas NP-Completos variam. Entretanto, há alguns tipos de provas que acontecem com frequência e que fornecem uma sugestão para decidir como provar um novo problema NP-Completo. Algumas dessas técnicas, classificadas por Garey e Johnson, são conhecidas por prova por: restrição, substituição local e projeto de componentes.

A prova por restrição é a mais simples e, talvez a mais frequentemente aplicada, das três técnicas. Uma prova de NP-Completude por restrição para um dado problema $\Pi \in NP$ consiste simplesmente de mostrar que Π contém um conhecido problema NP-Completo Π' como um caso especial. A essência da prova encontra-se na especificação das restrições adicionais a serem colocadas nas instâncias de Π tal que o problema restrito resultante seja idêntico a Π'. Não é necessário que o problema restrito e o problema conhecido como sendo NP-Completo sejam exatas duplicatas um do outro, mas deve existir uma correspondência óbvia entre suas instâncias que preserva as respostas "sim" e "não". A prova de NP-Completude que apresentamos para o problema do Caixeiro Viajante é um exemplo de prova por restrição.

A prova de NP-Completude por substituição local é muito bem exemplificada na transformação do problema SAT para 3SAT (de fato, é um exemplo clássico). Em relação à técnica de prova por projeto de componentes, alguns exemplos desse tipo de prova são: *3-Dimensional Matching*, *Vertex Cover* e Ciclo Hamiltoniano. Todas essas provas podem ser encontradas no clássico livro de Garey e Johnson.

1 Conclusão

Parte fundamental da complexidade de algoritmos, a análise de algoritmos é fundamental para ajudar no projeto de algoritmos eficientes, seja na escolha de um algoritmo entre outros já existentes, seja no desenvolvimento de um algoritmo para um problema ainda não resolvido. Por isso, nós discutimos, ainda que brevemente, algumas das principais técnicas de análise de algoritmos.

Sobre a teoria da complexidade computacional, existem ainda outras clas-

ses que não discutimos. Considere, por exemplo, o complemento dos problemas em NP, a chamada classe co-NP. Se Π é um problema de decisão em que procuramos identificar um certificado para uma instância *sim* do problema, o seu complementar Π' é aquele em que buscamos um certificado para o *não*. A classe co-NP é definida então pelos complementares dos problemas em NP. A classe P está contida em co-NP, o que indica que há interseção entre as classes NP e co-NP, porém até o presente momento não se sabe se NP=co-NP, ou se há problemas em uma classe que não estão em outra.

Poderíamos ainda citar a classe de NP-Completude Forte. Para os problemas dessa classe, só pode existir algum algoritmo pseudo-polinomial se P=NP.

Concluímos este texto frisando que, até obtermos uma prova positiva ou negativa à questão P versus NP, não poderemos ainda dizer se os problemas NP-completos são tratáveis. Contudo, mesmo que a resposta seja que P $\neq$ NP, ainda assim há problemas NP-completos que admitem um tratamento razoável desde que algumas concessões sejam feitas. Como exemplos, considere os algoritmos que buscam soluções que estão a um fator ϵ do ótimo, conhecidos como algoritmos aproximativos. Ou ainda, os algoritmos executados em tempo polinomial com uma probabilidade $p < 1$ de encontrar uma solução ótima (algoritmos randomizados). É ainda possível empreender uma busca por soluções ótimas através do uso de heurísticas e várias outras abordagens.

2 Bibliografia Utilizada

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. e STEIN, C. *Introduction to Algorithms*, 3^a edição, MIT Press, 2009.

GAREY, M. R. e JOHNSON, D. S. Computers and Intractability. A guide to the theory of NP-Completeness, W.H. Freeman and Company: New York, 1979.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Pascal e C, 2^a edição, Cengage Learning, 2009.