Existem algoritmos que não possuem tempo ótimo, um exemplo é o “problema de parada” de Turing, que não podem ser resolvidos por qualquer computador.

Algoritmos que podem ser resolvidos em tempo polinomial são considerados tratáveis, já os que não são em tempo polinomial e portanto precisão de um tempo maior que n^k para o *k* arbitrário e inteiro.

Esses problemas que não possuem solução polinomial fazem parte de uma classe especial de conhecida como NP-Completo ao qual ainda não foi descoberto nenhum algoritmo de tempo polinomial para um problema NP-Completo, porém ninguém foi capaz de provar que não exista um algoritmo de tempo polinomial para qualquer deles. (CORMEN, p.763)

O interessante que esses problemas NP-Completos são bastante semelhantes a problemas que possuem soluções e tempo polinomial, um exemplo é o caso do maior percurso em um grafo ao qual passa por todas as arestas, a primeira impressão que temos é de que este problema é muito semelhante ao caso de encontrar o menor caminho ao qual sabemos que o algoritmo de KRUSKAL.

Porém encontrar o caminho simples mais longo entre dois vértices é NP-completo, ele é NP-completo até mesmo se todos os pesos das arestas forem iguais a 1.

Isso mostra a dificuldade de verificar se um dado problema pertence à classe à NP-completo.

Portanto segundo o Cormen, para mostrar que um problema pertence à classe de NPC(NP-completo), contamos com três conceitos fundamentais:

Problemas de decisão *versus* problemas de otimização

“Muitos problemas de interesse são **problemas de otimização**, em que cada solução possível (válida) tem um valor associado, e desejamos encontrar a solução possível com o melhor valor” (CORMEN, p.765)

“Porém, o NP-completo não se aplica diretamente a problemas de otimização, mas a **problemas de decisão**, em que a resposta é simplesmente “sim” ou “não””. (CORMEN, p.765)

Assim se um problema de otimização é fácil, seu problema de decisão relacionado também é fácil e se pudermos provar que um problema de decisão é difícil, também existira evidências que seu problema de otimização é difícil.

Reduções

Considerando um problema de decisão, que iremos chamar de A, chamamos a entrada para um determinado problema de **instância** desse problema, na qual uma instancia seria um dado grafo G, vértices específicos *u* e *v* de G, e um certo inteiro *k*.

Agora suponha que exista um problema de decisão diferente, chamaremos de B, no qual conhecemos uma solução para ele em tempo polinomial. Suponha ainda que, temos um procedimento que transforma qualquer instância α de A em alguma instância β de B com as seguintes características:

1. A transformação demora tempo polinomial.
2. As respostas são as mesmas para ambos, ou seja, α é "sim" se e somente se β também for "sim".

Tal procedimento é conhecido como **algoritmo de redução** de tempo polinomial e ele nos oferece um meio para resolver o problema de A em tempo polinomial.

O problema NP-completo não podemos provar que não exista nenhum algoritmo polinomial que resolve o caso A. Porém, a partir da metodologia a cima temos que se provamos que o problema B é NP-completo então por hipótese A também será NP-completo.

Porém para a técnica da redução temos que conhecer de primeiramente um problema NP-completo e só então podemos afirmar que um segundo problema também é NP-completo.

Problema do Caixeiro Viajante:

Suponha que um caixeiro viajante tenha de visitar **n** cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. Suponha, também, que não importa a ordem com que as cidades são visitadas e que de cada uma delas pode-se ir diretamente a qualquer outra.
O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total.

Exemplificando o caso n = 4:
se tivermos quatro cidades A, B, C e D, uma rota que o caixeiro deve considerar poderia ser: *saia de A e daí vá para B, dessa vá para C, e daí vá para D e então volte a A*. Quais são as outras possibilidades? É muito fácil ver que existem seis rotas possíveis:

- ABCDA
- ABDCA
- ACBDA
- ACDBA
- ADBCA
- ADCBA

Complexidade computacional do problema do caixeiro:

O problema do caixeiro é NP-completo, desse modo é improvável que possamos encontrar um algoritmo em tempo polinomial para resolver esse problema com exatidão. Assim, este é um clássico exemplo de **problema de otimização combinatória**. A primeira coisa que podemos pensar para resolver esse tipo de problema é reduzi-lo a um **problema de enumeração**: achamos todas as rotas possíveis e calculamos o comprimento de cada uma delas e então vemos qual a menor. (É claro que se acharmos todas as rotas estaremos contando-as, daí podermos dizer que estamos reduzindo o problema de otimização a um de enumeração).

Para acharmos o número **R(n)** de rotas para o caso de **n** cidades, basta fazer um raciocínio combinatório simples. Por exemplo, no caso de n = 4 cidades, a primeira e última posição são fixas, de modo que elas não afetam o cálculo; na segunda posição podemos colocar qualquer uma das 3 cidades restantes B, C e D, e uma vez escolhida uma delas, podemos colocar qualquer uma das 2 restantes na terceira posição; na quarta posição não teríamos nenhuma escolha, pois sobrou apenas uma cidade; consequentemente, o número de rotas é 3 x 2 x 1 = 6, resultado que tínhamos obtido antes contando diretamente a lista de rotas acima.
De modo semelhante, para o caso de n cidades, como a primeira é fixa, o leitor não terá nenhuma dificuldade em ver que o número total de escolhas que podemos fazer é (n-1) x (n-2) x ... x 2 x 1. De modo que, para se resolver tal problema usando a notação assintótica é: **O (n!)**.

Já sabemos que esse tipo de problema não possui uma solução ótima pois não pertence a classe P (polinomiais). Suponhamos computador muito veloz, capaz de fazer 1 bilhão de adições por segundo. No caso de 20 cidades, o computador precisa apenas de 19 adições para dizer qual o comprimento de uma rota e então será capaz de calcular 10^9 / 19 = 53 milhões de rotas por segundo. Contudo, essa imensa velocidade é um nada frente à imensidão do número 19! de rotas que precisará examinar. Com efeito, acredite se puder, o valor de 19! é 121 645 100 408 832 000 (ou , aproximadamente, 1.2 x 10^{17} em notação científica). Consequentemente, ele precisará de

- 1.2 x 10^{17} / (53 milhões) = 2.3 x 10^9 segundos
- Para completar sua tarefa, o que equivale a cerca de **73 anos** . O problema é que **a quantidade (n - 1)! cresce com uma velocidade alarmante, sendo que muito rapidamente o computador torna-se incapaz de executar o que lhe pedimos.** Constate isso mais claramente na tabela a seguir:

n	rotas por segundo	(n - 1)!	cálculo total
5	250 milhões	24	Insignificante
10	110 milhões	362 880	0.003 seg
15	71 milhões	87 bilhões	20 min
20	53 milhões	1.2×10^{17}	73 anos
25	42 milhões	6.2×10^{23}	470 milhões de anos

Provando que o problema do caixeiro viajante é NP-Completo:

Para provarmos que o problema do caixeiro viajante é NP-completo vamos usar a metodologia apresentada na seção X.X na qual conhecendo um problema A no qual possui solução polinomial e dado um problema B então aplicando redução provamos que o problema B também tem solução polinomial. Porém, estamos tentando provar que o nosso problema não possui solução em tempo polinomial, assim, dado um problema A ao qual sabemos que não possui solução polinomial então, dado um problema B semelhante ao A concluímos que B também será NP-completo.
Vamos tomar como problema A para o problema do

A existência ou não de um método polinomial para resolver o problema do caixeiro viajante é um dos grandes problemas em aberto da Matemática na medida em que S. A. COOK (1971) e R. M. KARP (1972)) mostraram que uma grande quantidade de problemas importantes (como é o caso de muitos tipos de problemas de otimização combinatória, o caso do problema da decifragem de senhas criptografadas com processos modernos como o DES, etc) podem ser reduzidos, em tempo polinomial, ao problema do caixeiro.
Consequentemente: se descobrirmos como resolver o problema do caixeiro em tempo polinomial ficaremos sendo capazes de resolver, também em tempo polinomial, uma grande quantidade de outros problemas matemáticos importantes; por outro lado, se um dia alguém provar que é impossível resolver o problema do caixeiro em tempo polinomial no número de cidades, também se terá estabelecido que uma grande quantidade de problemas importantes não tem solução prática.

Costuma-se resumir essas propriedades do problema do caixeiro dizendo que ele pertence à categoria dos problemas **NP - completos**.

Um problema computacional é *de decisão* se cada uma de suas instâncias admite uma e apenas uma de duas respostas: SIM e NÃO. Diremos que uma instância é *positiva* se tem solução SIM e *negativa* em caso contrário.

ALGORITMOS POLINOMIAIS

Dizemos que um algoritmo *resolve* um determinado problema de decisão se, ao receber qualquer das instâncias do problema, devolve SIM se a instância for positiva e NÃO se a instância for negativa.

Um algoritmo que resolve um problema é *polinomial* se o seu consumo de tempo no pior caso é limitado por uma função polinomial dos tamanhos das instâncias. Em outras palavras, o algoritmo é polinomial se existe um número j tal que o consumo de tempo do algoritmo é

$O(N^j)$

sendo N o tamanho de uma instância. É polinomial, por exemplo, todo algoritmo que consome $O(N^9 \log N)$ unidades de tempo.

Algoritmos polinomiais são considerados rápidos. Algoritmos não polinomiais são considerados inaceitavelmente lentos.

CLASSE P DE PROBLEMAS

Um problema é *polinomial* se existe um algoritmo polinomial que resolve o problema. Problemas desse tipo são considerados tratáveis. Eis alguns exemplos de problemas polinomiais: QUADRADO-PERFEITO, EQ-GRAU-2, SUBSEQ-CRESC-LONGA, CAMINHO-CURTO.

Um problema é *não polinomial* se nenhum algoritmo polinomial resolve o problema. Problemas desse tipo são considerados intratáveis. Não é fácil dar exemplos interessantes de problemas não polinomiais

Suspeita-se que os seguintes sejam desse tipo: MOCHILA-VALIOSA, CAMINHO-LONGO, CICLO-LONGO, CICLO-HAMILT, CLIQUE-GRANDE, COBERTURA-PEQUENA.

Certificados e algoritmos verificadores

Diante da dificuldade de decidir se um determinado problema está ou não em P, é interessante estudar a complexidade *relativa* de problemas. Antes de fazer isso, entretanto, é preciso introduzir os conceitos de certificado e algoritmo verificador.

Um *algoritmo verificador* para um problema de decisão recebe dois objetos: uma instância do problema e uma cadeia de caracteres que chamaremos de *certificado*. (A ideia é que o certificado seja uma "prova" de que a instância é positiva. Num certo sentido, o conceito de certificado substitui o conceito de solução, que não faz sentido em problemas de decisão.) Ao receber esses dois objetos, o verificador pode responder SIM ou NÃO. Se responder SIM, dizemos que o verificador *aceitou* o certificado.

Observe que o conceito de verificador é assimétrico: ele cuida apenas das instâncias positivas do problema.

Um verificador para um determinado problema de decisão é *polinomial* se satisfaz as seguintes condições:

1. para cada instância positiva do problema, existe um certificado que o verificador aceita em tempo limitado por uma função polinomial do tamanho da instância;
2. para cada instância negativa do problema, não existe certificado que o verificador aceite.

É claro que a limitação polinomial do consumo de tempo faz com que o tamanho de um certificado aceitável também seja limitado por um polinômio no tamanho da instância. Ou seja, os certificados aceitáveis devem ser "curtos".

Alguns exemplos de certificados e algoritmos verificadores:

- Um certificado natural para uma instância (a,b,c) do problema EQ-GRAU-2 é um número inteiro x tal que $ax^2 + bx + c = 0$. Ao receber a, b, c e x, o verificador calcula $ax^2 + bx + c$ e compara esse número com 0. O verificador é polinomial pois qualquer x aceitável tem número de dígitos limitado por um

polinômio no tamanho da instância (a,b,c), uma vez que $|x| < (|a|+1)(|b|+1)(|c|+1)$.

- Um certificado natural para uma instância n do problema NÚMERO-COMPOSTO é um número natural p que divide n. O correspondente algoritmo verificador aceita p se e somente se n/p é inteiro. Esse verificador é polinomial pois qualquer p aceitável é menor que n.
- Um certificado para uma instância G do problema CICLO-HAMILT é a sequência de vértices de um ciclo hamiltoniano em G. O verificador aceita a sequência se ela de fato representa um ciclo hamiltoniano. É evidente que a verificação pode ser feita em tempo polinomial.

A classe NP de problemas

A *classe NP* de problemas é o conjunto de todos os problemas de decisão para os quais existem verificadores polinomiais. [Cuidado: "NP" não é abreviatura de "não polinomial"! "NP" é abreviatura de *"nondeterministic polynomial"*.]

Podemos dizer, informalmente, que um problema está em NP se possui um "certificado curto". (A classe NP corresponde à classe dos problemas que poderíamos chamar "razoáveis".)

Não é fácil dar bons exemplos de problemas que estejam fora de NP. Um exemplo bobo é o problema de construir uma lista de todos os subconjuntos de {1,2,…,n}, uma vez dado n.

Não é difícil mostrar que a classe NP inclui a classe P, ou seja, que todo problema polinomial de decisão está em NP.

Todos os problemas de decisão que mencionamos acima estão evidentemente em NP. Mas a pertinência de um problema de decisão à classe NP nem sempre é tão evidente, como mostra a seção seguinte.

A classe coNP de problemas

Os problemas de decisão ocorrem, naturalmente, em pares complementares. O *complemento* de um problema de decisão X é o problema de decisão Y que se obtém trocando todas as respostas SIM por NÃO e vice versa. Mais precisamente, Y é complemento de X se tem as mesmas instâncias que X mas toda instância positiva de X é negativa em Y e vice-versa. Por exemplo, o complemento do problema NÚMERO-COMPOSTO é o seguinte problema

- NÚMERO-PRIMO: Decidir se um dado número natural $n > 1$ é primo.

Os complementos dos problemas EQ-GRAU-2, CICLO-LONGO e CICLO-HAMILT consistem no seguinte:

- CO-EQ-GRAU-2: Dados inteiros a, b, c, decidir se $ax^2 + bx + c \neq 0$ para todo inteiro x.
- CO-CICLO-LONGO: Dado um grafo G e um número k, decidir se todo ciclo em G tem menos que k arestas.

- CO-CICLO-HAMILT: Decidir se um dado grafo não tem um ciclo hamiltoniano.

A *classe coNP* de problemas é o conjunto de todos os problemas de decisão cujo complemento está em NP. (Podemos dizer, informalmente, que coNP é a classe dos problemas de decisão que têm verificadores polinomiais para a resposta NÃO.) É fácil mostrar que a classe coNP inclui a classe P.

Considere alguns exemplos:

- O problema NÚMERO-PRIMO está em NP (embora isso esteja longe de ser óbvio) e portanto o problema NÚMERO-COMPOSTO está em coNP.

  (A propósito: descobriu-se recentemente que o problema NÚMERO-PRIMO está em P. Portanto, o problema NÚMERO-COMPOSTO também está em P. Acredita-se, entretanto, que o problema FATORAÇÃO não está em P.)
- O problema EQ-GRAU-2 está em coNP pois está em P (graças à fórmula $(-b \pm (b^2-4ac)^{½})/2a$).
- Não se sabe se o problema CICLO-LONGO está em coNP, pois não se sabe se o problema CO-CICLO-LONGO está em NP.
- Não se sabe se o problema CICLO-HAMILT está em coNP.
- Não se sabe se o problema CLIQUE-GRANDE está em coNP.

Exercícios

1. É verdade que o complemento de todo problema em P também está em P?
2. É verdade que todo problema em NP também está em coNP?

A questão "P = NP?"

Como já observamos, P é parte de NP. O bom senso sugere que P é apenas uma pequena parte de NP. Surpreendentemente, ninguém conseguiu ainda encontrar um problema de NP que não esteja em P.

Essa situação abre caminho para a suspeita de que P seja, talvez, igual a NP. Mas a maioria dos especialistas não acredita nessa possibilidade. [O Instituto Clay de Matemática oferece um milhão de dólares pela solução da questão "P=NP?".]

Veja verbete *P = NP problem* na Wikipedia. Veja também página de Woeginger com um histórico de provas frustadas de "P=NP" e "P≠NP".

Redução entre problemas

Diante da dificuldade de decidir se um dado problema está em P, é interessante estudar a complexidade *relativa* dos problemas da classe NP. Dados dois problemas *X* e *Y* em NP, gostaríamos de poder dizer algo como "*X* é tão difícil quanto *Y*".

Um problema de decisão *Y* é *redutível* a um problema de decisão *X* se existe um algoritmo que transforma qualquer instância J de *Y* em uma instância I de *X* tal que J é positiva se e somente se I é positiva. Pode-se dizer, muito informalmente, que *Y* é redutível a *X* se *Y* for um "subproblema", ou "caso particular", de *X*.

Estamos interessados apenas em reduções *polinomiais*, ou seja, reduções em que o algoritmo de redução é polinomial. Se existe uma redução polinomial de *X* a *Y*, escrevemos $X \leq_P Y$. Alguns exemplos:

- o problema QUADRADO-PERFEITO é polinomialmente redutível ao problema EQ-GRAU-2
- SUBSET-SUM é polinomialmente redutível a MOCHILA-VALIOSA
- CAMINHO-HAMILT $\leq_P$ CICLO-HAMILT
- CICLO-HAMILT $\leq_P$ CAMINHO-HAMILT
- COBERTURA-PEQUENA $\leq_P$ CLIQUE-GRANDE
- COBERTURA-PEQUENA $\leq_P$ CICLO-HAMILT

Para quaisquer dois problemas *X* e *Y* na classe NP, se *X* está em P e existe uma redução polinomial de *Y* a *X* então *Y* também está em P. De fato, dada uma instância J de *Y*, basta reduzir J a uma instância I de *X* e em seguida submeter I a um algoritmo polinomial que resolve *X*.

Exercícios

1. Suponha que os algoritmos *A* e *B* transformam cadeias de caracteres em outras cadeias de caracteres. O algoritmo *A* consome $O(N^2)$ unidades de tempo e o algoritmo *B* consome $O(N^4)$ unidades de tempo, sendo *N* é o número de caracteres da cadeia de entrada. Considere agora o algoritmo *AB* que consiste na composição de *A* e *B*, com *B* recebendo como entrada a saída de *A*. Qual o consumo de tempo de *AB*?
2. Mostre que todo problema em P é polinomialmente redutível ao problema CICLO-HAMILT.
3. Mostre que CAMINHO-HAMILT $\leq_P$ CICLO-HAMILT.
4. Mostre que CICLO-HAMILT $\leq_P$ CAMINHO-HAMILT.
5. Mostre que SET-PARTITION $\leq_P$ SUBSET-SUM.
6. Mostre que SUBSET-SUM $\leq_P$ SET-PARTITION.

Problemas completos em NP

Um problema de decisão *X* é *completo em NP* (ou *NP-completo*) se *X* está em NP e *qualquer* outro problema em NP pode ser polinomialmente reduzido a *X*. (Informalmente, *X* é NP-completo se for tão difícil quanto qualquer outro problema em NP.)

A existência de problemas completos em NP é um fato surpreendente e fundamental. O fato foi demonstrado, por volta de 1970, por S. Cook e L. Levin (independentemente).

Para provar que P = NP basta encontrar um algoritmo polinomial para um único problema NP-completo.

Exercícios

1. Sabendo-se que SUBSET-SUM é NP-completo, mostre que SET-PARTITION é NP-completo.

A classe NPC de problemas

A *classe NPC* é o conjunto de todos os problemas completos em NP. Portanto, NPC é a classe dos problemas mais "difíceis" de NP.

Eis alguns problemas em NPC: CICLO-HAMILT, CICLO-LONGO, CAMINHO-LONGO, CLIQUE-GRANDE, COBERTURA-PEQUENA, SUBSET-SUM, MOCHILA-VALIOSA. (Não se sabe, entretanto, se FATORAÇÃO está em NPC.)

Uma lista com centenas de problemas NP-completos pode ser encontrada no livro de Garey e Johnson. Veja também *An Annotated List of Selected NP-complete Problems*.