

Computabilidade e Complexidade de Algoritmos



Cruzeiro do Sul Virtual
Educação a distância

Material Teórico



Problemas NP-Completos

Responsável pelo Conteúdo:

Prof. Dr. Luciano Rossi

Revisão Textual:

Prof.^a Esp. Kelciane da Rocha Campos

UNIDADE

Problemas NP-Completos



- A Classe P;
- A Classe NP;
- NP-Completeness.



OBJETIVOS DE APRENDIZADO

- Apresentar ao aluno as diferenças entre os algoritmos que podem obter soluções em tempo polinomial daqueles que não podem;
- Mostrar que as complexidades de muitos problemas podem estar interligadas e como isso pode ser utilizado para resolver uma classe inteira de problemas.

Orientações de estudo

Para que o conteúdo desta Disciplina seja bem aproveitado e haja maior aplicabilidade na sua formação acadêmica e atuação profissional, siga algumas recomendações básicas:



Assim:

- ✓ Organize seus estudos de maneira que passem a fazer parte da sua rotina. Por exemplo, você poderá determinar um dia e horário fixos como seu “momento do estudo”;
- ✓ Procure se alimentar e se hidratar quando for estudar; lembre-se de que uma alimentação saudável pode proporcionar melhor aproveitamento do estudo;
- ✓ No material de cada Unidade, há leituras indicadas e, entre elas, artigos científicos, livros, vídeos e sites para aprofundar os conhecimentos adquiridos ao longo da Unidade. Além disso, você também encontrará sugestões de conteúdo extra no item **Material Complementar**, que ampliarão sua interpretação e auxiliarão no pleno entendimento dos temas abordados;
- ✓ Após o contato com o conteúdo proposto, participe dos debates mediados em fóruns de discussão, pois irão auxiliar a verificar o quanto você absorveu de conhecimento, além de propiciar o contato com seus colegas e tutores, o que se apresenta como rico espaço de troca de ideias e de aprendizagem.

A Classe P

Antes de iniciarmos as definições sobre as classes de problemas, computáveis ou não, vamos pensar sobre a forma que utilizamos para definir o tempo que os algoritmos gastam no processamento destes problemas, ou seja, sua complexidade de tempo. Os algoritmos vistos até aqui têm a complexidade de tempo proporcional a um polinômio.

Um polinômio é uma expressão algébrica que é composta por monômios e por operadores aritméticos. Um monômio apresenta em sua constituição um coeficiente que multiplica uma variável. O grau de um polinômio é definido pelo maior expoente de uma variável.

Um algoritmo de tempo polinomial é aquele que tem seu tempo de execução proporcional a $O(n^k)$ para o pior caso. Assim, n representa o tamanho da entrada (instância) do algoritmo e k é alguma constante.

Existem algoritmos de tempo polinomial que resolvem alguns tipos de problema, mas não todos. O problema de decidir se um programa termina sua execução, dada uma determinada instância, não tem solução computacional conhecida. Esse problema é denominado problema da parada. Outros tipos de problemas podem ter soluções computacionais, mas não em tempo polinomial.

Os problemas que podem ser resolvidos com um algoritmo de tempo polinomial são definidos como problemas tratáveis; caso contrário, são definidos como intratáveis. Os problemas de um terceiro grupo, para os quais não se sabe se existe um algoritmo que os resolva, seja em tempo polinomial ou não, são denominados problemas NP-completos.

A classe P é aquela que reúne problemas que têm solução em tempo polinomial em uma máquina de Turing determinística. No início desta disciplina, vimos a definição de uma máquina de Turing, a qual descreve um modelo de um computador de propósito geral. Em termos gerais, uma máquina de Turing decide se aceita ou não uma determinada entrada.

Uma linguagem é o conjunto de cadeias que são reconhecidas por uma máquina de Turing; assim, dizemos que uma linguagem é Turing-reconhecível se alguma máquina de Turing a reconhece. Quando uma máquina de Turing está computando uma entrada, podemos ter três resultados sobre ela, a máquina pode aceitar ou rejeitar a entrada, ou ainda nunca parar (entrar em *looping*).

As máquinas de Turing que sempre param, ou seja, sempre aceitam ou rejeitam uma entrada, são chamadas decisores. Assim, uma máquina de Turing (M) decide se uma determinada entrada pertence à linguagem de M . O termo determinístico, referente à máquina de Turing, define um decisor.

A classe P reúne problemas que podem ser resolvidos em tempo polinomial por uma máquina de Turing determinística. Porém, para compreender melhor essa definição, é preciso definir o conceito de problema. Um **problema abstrato** é uma relação binária entre uma instância do problema e uma solução para ele; veja na Figura 1 uma ilustração da relação entre uma instância do problema e sua solução.

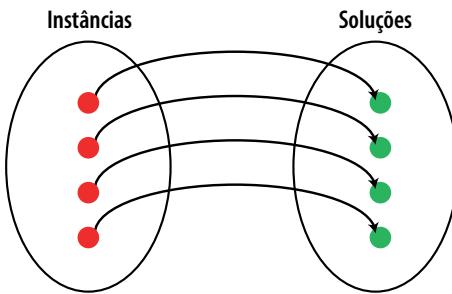


Figura 1 – Exemplo de problema abstrato

A definição anterior para problemas abstratos é mais geral do que é preciso para o contexto das classes de problemas. Nesse contexto, estamos interessados apenas em **problemas de decisão**. Para a definição de problemas de decisão abstratos, o conjunto de soluções, representado na Figura 1, teria apenas dois elementos $Soluções = \{0,1\}$.

Suponha um algoritmo que verifique se existe um caminho, de tamanho k , entre dois vértices em um grafo (veja a a unidade “Análise de Desempenho de Algorítimos”). Nesse contexto, o algoritmo retornará 1, indicando que existe um caminho de tamanho k , ou 0, caso contrário. Esse é um exemplo de problema de decisão, ou seja, a resposta para o problema está relacionada com a decisão da existência ou não de uma solução com aquelas características.

Considere a Figura 2, na qual há uma representação de um grafo. Suponha que se queira verificar se existe um caminho de tamanho $k = 5$ entre os vértices a e e . Um algoritmo de decisão A que realiza essa tarefa retornaria $A(G, a, e, 5) = 1$, indicando que existe um caminho entre os vértices a e e com cinco arestas $\{(a, b), (b, c), (c, g), (g, f), (f, e)\}$. Para o caso de não existir o caminho no grafo, o algoritmo retornará 0; por exemplo, $A(G, c, h, 2) = 0$, pois não existe um caminho entre os vértices c e h de tamanho $K = 2$.

Os **problemas de otimização** são aqueles que buscam uma solução que represente um valor; nesse caso, o melhor valor possível. Veja que estamos interessados em problemas de decisão; assim, os problemas de otimização não se aplicam a esse contexto.

Um algoritmo de otimização B poderia ser considerado para encontrar o menor caminho entre dois vértices no grafo da Figura 2. Por exemplo, $B(G, c, j) = 5$; essa instância do problema retorna o valor 5, indicando que existe um caminho entre os vértices c e j com cinco arestas $\{(c, g), (g, f), (f, e), (e, i), (i, j)\}$.

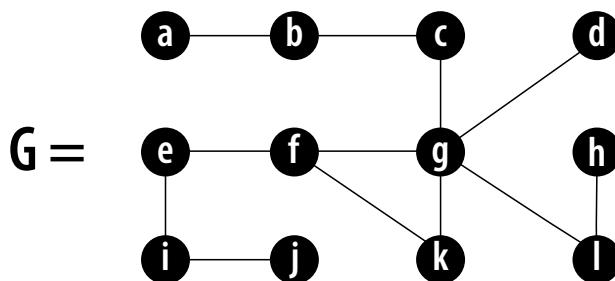


Figura 2 – Exemplo de grafo não direcionado

Há uma correspondência entre os problemas de **otimização** e de **decisão**. O exemplo do problema anterior, de encontrar um caminho de tamanho k em um grafo, é um

caso particular de um problema de otimização. O problema de otimização, mais geral, seria o de encontrar o menor caminho entre dois vértices em um grafo. Veja que poderíamos utilizar o caso particular diversas vezes com $k = \{1, 2, \dots, n\}$ tamanhos de caminho; assim, podemos obter uma solução para um problema de otimização considerando um problema de decisão.

A representação de um problema de otimização como um problema de decisão é útil para mostrar a qual classe o problema pertence. Denominamos de redução a representação de um problema de otimização em um equivalente de decisão. Considerar somente problemas de decisão nos permite utilizar os mecanismos das linguagens formais. Anteriormente, vimos a definição de modelos de computação, que são representações abstratas de um computador idealizado de modo que nos permita manipulá-lo por meio de uma teoria matemática. Nesse contexto, as definições consideradas pela teoria das linguagens formais nos são bastante úteis.

As cadeias de caracteres são sequências de símbolos e um alfabeto é um conjunto de símbolos que são utilizados para compor as cadeias. Assim, uma cadeia sobre um alfabeto é uma sequência finita de símbolos pertinentes ao alfabeto. Por exemplo, o alfabeto $\Sigma = \{0,1\}$ pode gerar cadeias do tipo: 01001, 111001, 100111, dentre outras infinitas possibilidades. Seja w uma cadeia sobre Σ , então o comprimento da cadeia $|w|$ é a quantidade de símbolos que ela contém. Assim, para $w = 110011$, $|w| = 6$. Uma cadeia de tamanho zero é uma cadeia vazia (ϵ).

Uma linguagem L é um conjunto de cadeias; assim, $L = \{001,0011,000111\}$ é uma linguagem sobre o alfabeto Σ . A linguagem de todas as cadeias sobre Σ é denotada por $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$.

Neste contexto, as instâncias de um problema de decisão Q são Σ^* , podemos considerar Q como uma linguagem $L = \{x \in \Sigma^* : Q(x) = 1\}$ em $\Sigma = \{0,1\}$. Assim, a linguagem L é composta por cadeias x pertinentes à linguagem Σ^* , de modo que as cadeias são instância do problema de decisão Q cujo resultado é 1.

Cormen (2009) define o problema de decidir se existe um caminho de tamanho k entre dois vértices em um grafo G , a partir da notação anterior, como sendo a linguagem:

$PATH = \{(G, u, v, k) : G = (V, E), u \in V, v \in V, k \in \mathbb{N}, \exists u \rightarrow v \in E, |u \rightarrow v| \leq k\}$. Ou seja, a linguagem de $PATH$ é o conjunto de instâncias G, u, v, k onde:

- G é um grafo não dirigido;
- u e v são vértices no grafo G ;
- k é um número inteiro positivo;
- e existe um caminho entre u e v com no máximo k arestas.

A linguagem $PATH$ é o conjunto de todas as instâncias que satisfazem as restrições definidas para o conjunto.

Um algoritmo A **aceita** uma instância $x \in \{0,1\}^*$ se $A(x) = 1$; assim, a linguagem $L = \{x \in \{0,1\}^* : A(x) = 1\}$ é o conjunto de instâncias que são aceitas pelo algoritmo A . O algoritmo A **aceita** a linguagem L em **tempo polinomial** se, além de L ser **aceita**

por A , existe um k para qualquer instância $x \in L$ e $|x| = n$, de modo que o tempo seja proporcional a $O(n^k)$.

Um algoritmo A **decide** a linguagem L se $A(x) = 1$, se $x \in L$, e $A(x) = 0$, se $x \notin L$. Em outras palavras, o algoritmo retorna sempre um resultado: aceita ou não aceita. Veja que um algoritmo que **aceita** a linguagem pode não retornar nada, caso a instância de entrada não pertença à linguagem. O algoritmo A **decide** a linguagem L em **tempo polinomial** se, além de L ser **decida** por A , existe um k para qualquer instância $x \in L$ e $|x| = n$, de modo que o tempo seja proporcional a $O(n^k)$.

Nesse contexto, podemos dar uma definição mais formal para os problemas da classe P, da seguinte forma:

$$P = \{L \subseteq \{0,1\}^*: \text{existe um algoritmo } A \text{ que decide } L \text{ em tempo polinomial}\}.$$



Importante!

Um algoritmo pode aceitar uma linguagem e não decidi-la. Estamos interessados em algoritmos determinísticos, ou seja, aqueles que decidem sobre a pertinência ou não a uma determinada linguagem.

Os algoritmos de verificação são utilizados para verificar se um determinado “certificado” pode confirmar ou não uma suposição. Considere o grafo da Figura 2, suponha que seja fornecida a seguinte lista de arestas $\{(c, g), (g, f), (f, e), (e, i), (i, j)\}$ como forma de comprovar a existência de um caminho entre os vértices c e i , de tamanho $k = 5$, no grafo G . Veja que é possível verificar, em tempo polinomial, se o certificado fornecido é um caminho entre os vértices c e i no grafo G .

Os **algoritmos de verificação** recebem dois argumentos, o primeiro é a cadeia de entrada x que no exemplo anterior é representada por todas as cadeias que compõem a linguagem reconhecida pelo algoritmo, e o segundo argumento é uma cadeia que certificará ou não o algoritmo.

Em termos mais formais, Comen (2009) define um algoritmo de verificação A como aquele que verifica a seguinte linguagem L :

$$L = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^* \text{ tal que } A(x, y) = 1\}$$

Para ilustrar o conceito dos algoritmos de verificação, vamos considerar como exemplo os ciclos hamiltonianos. Um grafo é hamiltoniano se ele possuir um ciclo simples com todos os vértices do grafo. A Figura 3 apresenta um exemplo de grafo hamiltoniano; veja que as arestas, destacadas em vermelho, formam um ciclo simples, reunindo todos os vértices do grafo.



Importante!

Um ciclo em um grafo é um caminho que começa e termina no mesmo vértice. Um ciclo é simples se não houver repetição de vértices.

Um algoritmo de verificação poderia verificar, em tempo polinomial, se um grafo é ou não hamiltoniano. Nesse caso, o algoritmo receberia o grafo a ser verificado e uma lista ordenada de vértices que compõem o ciclo hamiltoniano. Caso o algoritmo reconheça a lista de vértices como um caminho hamiltoniano no grafo, ele retorna 1, confirmado que o grafo é hamiltoniano. A lista de vértice é o certificado dessa verificação.

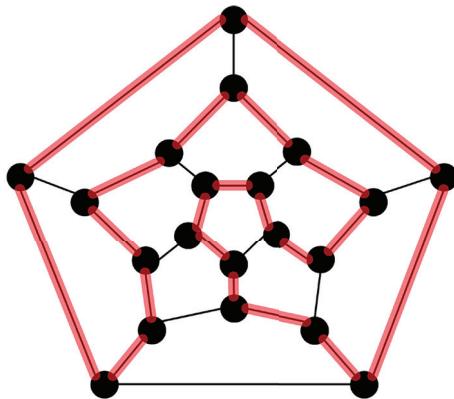


Figura 3 – Exemplo de grafo hamiltoniano

Fonte: Adaptada de CORMEN, 2009

A Classe NP

A classe de problemas que chamamos de NP, em referência a tempo superpolinomial, reúne os problemas cuja solução não pode ser obtida em tempo polinomial. Outra forma de definir a classe NP é como um conjunto de problemas que podem ser **verificáveis** em tempo polinomial. Veja que há uma diferença entre resolver e verificar um problema.

Considerando o exemplo anterior, o problema de se obter um ciclo hamiltoniano em um grafo requer a **resolução** do problema. Assim, teríamos que buscar por um ciclo hamiltoniano e, caso seja identificado, fornecer os vértices que compõem o ciclo ou informar que não existe tal ciclo. Desse modo estamos resolvendo o problema.

Para o caso de possuirmos uma lista de vértices que, supostamente, compõem um ciclo hamiltoniano, em um determinado grafo, teríamos que **verificar** se a lista de fato representa um ciclo hamiltoniano no grafo. Nesse caso, nosso problema é de verificação. Assim, os problemas da classe NP apresentam essa característica, ou seja, podem ser verificados em tempo polinomial.

Vimos anteriormente que um **problema abstrato** é uma relação entre uma instância e uma solução; podemos ter muitas instâncias e soluções para um mesmo problema. Para resolver um problema abstrato em um computador, devemos utilizar uma representação das suas instâncias de modo que o computador possa entender. O processo de representar as instâncias de um problema de forma que o computador entenda é chamado de codificação.

Podemos codificar qualquer instância de um problema abstrato em uma cadeia sobre um alfabeto finito com pelo menos dois símbolos. Nesse contexto, um **problema concreto** é aquele que considera as suas instâncias codificadas como cadeias de símbolos,

de modo que possa ser considerada por um computador. Nesse contexto, um conjunto de instâncias pode ser codificado em um conjunto de cadeias de símbolos - por exemplo, os binários. Veja que um conjunto de cadeias de símbolos forma uma linguagem; assim, as instâncias de um problema podem ser representadas por uma determinada linguagem.

Uma linguagem, que representa as instâncias codificadas de um determinado problema, pertence à classe NP se e somente se existe um algoritmo que a verifica em tempo polinomial. Formalmente, podemos definir os problemas na classe NP como aqueles cujas instâncias são representadas pela seguinte linguagem:

$$L = \{x \in \{0,1\}^*: \exists \text{ certificado } y \text{ com } |y| = O(|x|^c) \text{ tal que } A(x, y) = 1\}$$

Nesse contexto, a linguagem anterior é composta por cadeias no alfabeto $\{0,1\}^*$ para as quais existe um certificado y cujo tamanho é proporcional a uma função de complexidade polinomial em x , de modo que exista um algoritmo que valide a cadeia a partir do certificado.

Veja que se uma linguagem pertence à classe NP, então ela também pertence à classe P, pois se existe um algoritmo de tempo polinomial que decide sobre uma linguagem, esse algoritmo pode ser convertido em um algoritmo de verificação sem certificado, que aceita todas as cadeias de entrada.

As classes de complexidade dos problemas despertam o interesse de pesquisa de vários cientistas da computação. Essas pesquisas consideram diferentes hipóteses de relacionamento entre as classes. Veja a Figura 4, que ilustra as quatro hipóteses consideradas pelos pesquisadores.

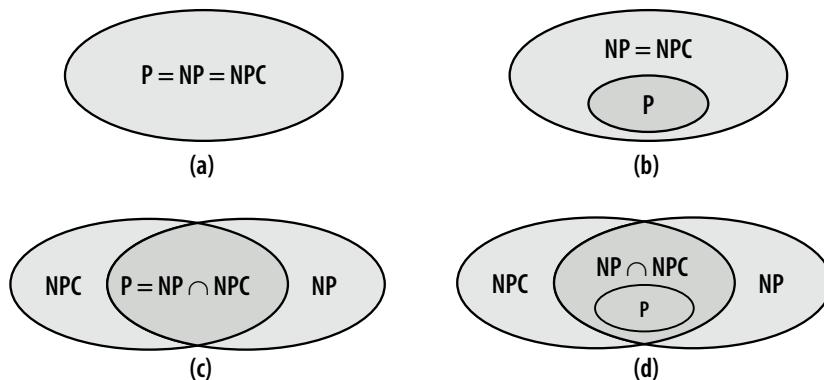


Figura 4 – Relacionamento entre as classes de complexidade de problemas

Fonte: Adaptada de CORMEN, 2009

Veja que não há consenso quanto a uma definição precisa sobre as classes de complexidade de problemas. As questões referentes às definições das classes estão, ainda, em aberto. Porém, existem evidências de que a classe P seja, de fato, diferente da classe NP, o que nos leva a outra classe de complexidade de problemas, que é denominada NP-completa.

Uma hipótese inicial, considerada sobre as classes de complexidade de problemas, é que as classes pertençam a um mesmo conjunto, como descrito na Figura 4a. Essa hipótese é a mais improvável, sendo rejeitada pela maioria dos pesquisadores na área. Uma segunda hipótese considera que as classes NP e NP-completa são iguais, mas a

classe P não é, necessariamente, igual à classe NP (Figura 4b). A terceira hipótese descreve a classe P como a interseção das classes NP e NP-completa, sendo essas últimas diferentes (Figura 4c). A última hipótese considera as classes NP e NP-completa como diferentes e a classe P é, também, diferente da interseção das classes NP e NP-completa, essa hipótese é considerada a mais provável pelos especialistas (Figura 4d).

NP-Completude

A classe de complexidade de problemas NP-completos é considerada como base da hipótese de que as classes P e NP sejam diferentes. Uma propriedade da classe de problemas NP-completos descreve que “se algum problema NP-completo pode ser resolvido em tempo polinomial, então todo problema em NP tem uma solução em tempo polinomial, isto é, $P = NP$ ” (CORMEN, 2009).



Importante!

Mesmo com toda pesquisa realizada pelos especialistas na área, até hoje não foi descoberto um algoritmo em tempo polinomial que resolva qualquer problema NP-completo.

A redutibilidade é o termo utilizado para descrever o processo de redução de um problema em outro cuja solução seja, também, uma solução para o primeiro. Se um problema Q pode ser reformulado em outro problema Q' e a solução de uma instância em Q' é, também, uma solução para Q , dizemos que Q pode ser reduzido a Q' .

A Figura 5 ilustra a redução de uma linguagem L_1 para uma linguagem L_2 . Veja que se uma instância $x \in L_1$ tem o mesmo resultado da função em tempo polinomial $f(x) \in L_2$. Em outras palavras, uma instância de um problema é reduzida em tempo polinomial a uma instância de outro problema, de modo que a solução de ambas as instâncias sejam iguais.

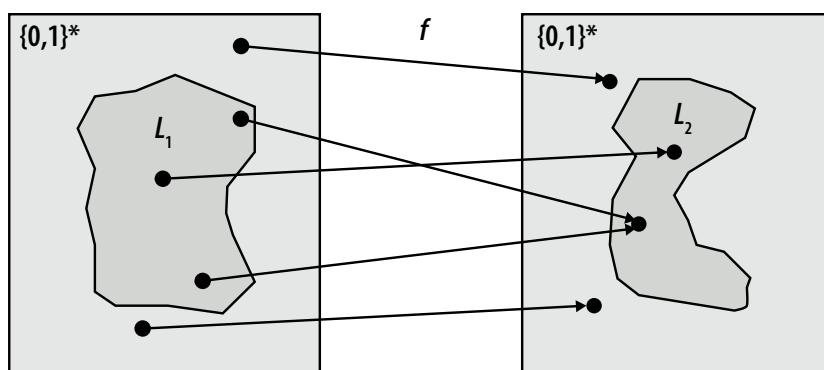


Figura 5 – Redução de uma linguagem L_1 para uma linguagem L_2
Fonte: Adaptada de CORMEN, 2009

Considere um exemplo, o problema de resolver uma equação linear pode ser reduzido no problema de resolver uma equação quadrática. Assim, uma instância de $ax + b = 0$ pode ser reduzida para uma instância $0x^2 + ax + b = 0$, veja que a solução de ambas as instâncias é a mesma.

As reduções são muito úteis para mostrar a pertinência de linguagens a determinadas classes. Suponha uma linguagem L_1 que pode ser reduzida em tempo polinomial para uma linguagem L_2 ; assim, se é possível mostrar que L_2 pertence à classe de complexidade P, então L_1 também pertence a P.

A Figura 6 ilustra a prova de pertinência de uma linguagem por meio da redução em outra linguagem. Considere o algoritmo de redução F que reduz, em tempo polinomial, a instância $x \in L_1$ para a instância $f(x) \in L_2$. O algoritmo A_2 decide a pertinência da instância $f(x)$ à linguagem L_2 ; consequentemente, o algoritmo A_1 decide a pertinência da instância x à linguagem L_1 a partir da decisão do algoritmo A_2 sobre a instância $f(x)$.

As reduções são ferramentas úteis para mostrar, de maneira formal, que um determinado problema é no mínimo tão difícil quanto outro. A partir disso, podemos obter uma definição para o conjunto de linguagens NP-completas. Assim, uma linguagem L é NP-completa se $L \in NP$ e L' é redutível em tempo polinomial à L , para toda linguagem $L' \in NP$.

A maioria dos especialistas em ciência da computação considera que tanto a classe P quanto a classe NP-completa (NPC) estão inteiramente contidas em NP ($P, NPC \subseteq NP$) e que não há problemas comuns a essas classes ($P \cap NPC = \emptyset$).

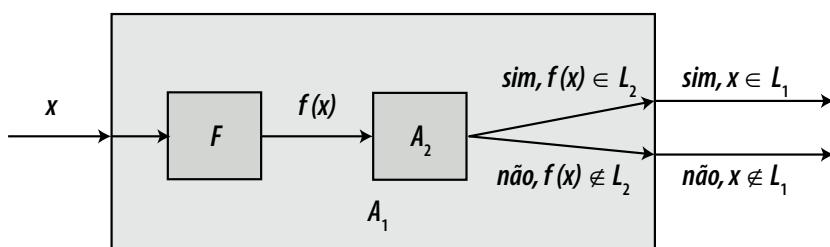


Figura 6 – Prova de pertinência a partir de redução

Fonte: Adaptada de CORMEN, 2009

Utilizando a noção de redutibilidade em tempo polinomial, se tivermos ao menos um problema que seja, comprovadamente, da classe NP-completa, poderíamos provar que instâncias de outros problemas também pertencem à classe NP-completa, por meio da redução de uma instância em outra.

O problema da satisfazibilidade de circuitos é um problema da classe NP-completa. Circuitos combinacionais booleanos são uma coleção de elementos combinacionais booleanos interligados por fios. Os elementos combinacionais booleanos são portas lógicas que apresentam como entrada e saída valores lógicos.



Os circuitos combinacionais apresentam o sinal de saída que depende exclusivamente das combinações dos sinais de entrada. Esses circuitos não consideram nenhum tipo de memória; assim, as suas saídas são independentes de estados anteriores do circuito.

A porta lógica unária NOT (não) recebe um valor binário de entrada e produz um valor binário como saída. A operação da porta NOT consiste da inversão do valor de entrada; por exemplo, a entrada 1 produzirá a saída 0 e vice-versa.

As portas lógicas binárias *AND* (e) e *OR* (ou) recebem dois valores binários de entrada e produzem um valor binário como saída. A operação da porta *AND* produz como saída o valor 1, caso ambas as entradas sejam correspondentes ao valor 1, e 0 caso contrário. A operação da porta lógica *OR* produz como saída o valor 0, caso ambas as entradas sejam correspondentes ao valor 0, e 1 caso contrário. Podemos generalizar as portas para que recebam mais que duas entradas. Assim, a porta *AND* produzirá como saída o valor 1 somente quando todas as entradas forem 1 e a porta *OR* produzirá o valor 0 somente quando todas as entradas forem 0.

Um circuito combinacional booleano, para este contexto específico, apresenta uma única saída. O problema da satisfazibilidade de circuitos é definido por Cormen (2009) da seguinte forma: “dado um circuito combinacional booleano composto pelas portas *AND*, *OR* e *NOT*, determinar se ele é satisfazível”. Um circuito será satisfazível se para uma determinada instância ele produz a saída 1.

A Figura 7 apresenta duas instâncias do problema da satisfazibilidade de circuitos. Cada circuito recebe uma atribuição composta por três valores binários (x_1 , x_2 , x_3). Uma atribuição verdade é aquela que produz uma saída de valor 1 no circuito. Veja que, na instância da Figura 7a, a atribuição verdade $x_1 = 1$, $x_2 = 1$ e $x_3 = 0$ está produzindo a saída de valor 1 para o circuito.

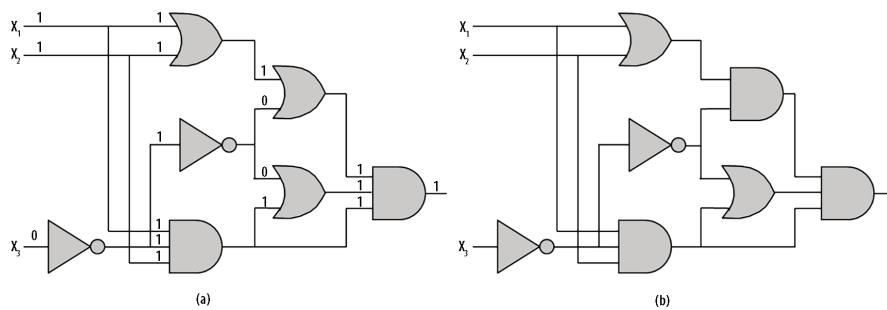


Figura 7 – Exemplos de instâncias do problema da satisfazibilidade de circuitos

Fonte: Adaptada de CORMEN, 2009

O circuito combinacional booleano da Figura 7b não produz valor 1 para nenhuma combinação de valores das entradas. Assim, dizemos que o primeiro circuito é satisfazível e o segundo é insatisfazível. Esse problema surge a partir da otimização de circuitos eletrônicos. Um subcircuito que sempre produz o valor 0 é desnecessário e pode ser substituído com a eliminação de suas portas lógicas. O desafio aqui é poder identificar esse tipo de circuito combinacional booleano, visto que as combinações possíveis para k entradas é 2^k atribuições possíveis. Nesse contexto, o problema da satisfazibilidade de circuitos booleanos demora um tempo superpolinomial em relação ao tamanho do circuito.

Para demonstrar que o problema da satisfazibilidade de circuitos booleanos é NP-completo, devemos considerar as duas partes da definição de NP-completude, as quais foram apresentadas anteriormente, que define uma linguagem $L \subseteq \{0,1\}^*$ como NP-completa se:

- $L \in NP$ e
- $L' \leq_p L$ para todo $L' \in NP$

Neste caso, a notação $L' \leq_p L$ indica que a linguagem L' é redutível em tempo polinomial à linguagem L .

Inicialmente, devemos mostrar que o problema da satisfazibilidade de circuitos booleanos pertence à classe NP. Para isso, teremos que obter um algoritmo que receba duas entradas: (i) uma codificação de um circuito booleano e (ii) um certificado que descreve as entradas do circuito. O algoritmo deve retornar 1 sempre que o certificado produzir como saída do circuito o valor 1; caso contrário, o algoritmo deve retornar 0.

A segunda parte da prova consiste em mostrar que todo problema em NP é reduzível em tempo polinomial ao problema da satisfazibilidade de circuitos booleanos.

Mostrar que o problema da satisfazibilidade de circuitos booleanos é NP-completo não é uma tarefa trivial. Existem muitos detalhes técnicos que devem ser considerados para a prova. Por exemplo, a forma como um programa é armazenado na memória do computador e como um processo executa as instruções deve ser plenamente conhecida, de modo que se possa compreender a sua codificação como um circuito combinacional booleano.

Para que se possa ter uma intuição a respeito da prova, uma sugestão é a leitura da obra de Cormen (2009), mais especificamente as páginas 779-785 trazem uma descrição detalhada sobre o assunto.

Para o nosso contexto, o importante é saber especificar a classe de problemas NP-completos e que, a partir da identificação de um problema NP-completo, como é o caso da satisfazibilidade de circuitos booleanos, podemos mostrar que outros problemas também o são, a partir da sua redução para o problema da satisfazibilidade.

Considerando a definição anterior para determinar se um dado problema é NP-completo, suponha que toda linguagem reduzida a partir de uma linguagem L seja pertinente à classe NP, satisfazendo a segunda propriedade das linguagens NP-completas, porém não é possível verificar se, de fato, a linguagem L pertence à classe NP, como previsto pela primeira propriedade das linguagens NP-completas. Desse modo, dizemos que a linguagem L é **NP-difícil**.

O problema da satisfazibilidade de fórmulas booleanas é um problema NP-completo, pois pode ser reduzido a um problema de circuito, o qual é pertinente à classe NP-completa.

Uma instância do problema de satisfazibilidade de fórmulas considera n variáveis booleanas e m operadores lógicos. Uma instância desse tipo tem a seguinte forma:

$$\phi = ((x_1 \rightarrow x_2) \vee \sim ((\sim x_1 \leftrightarrow x_3) \vee x_4)) \wedge \sim x_2$$

Na notação anterior, x_1 , x_2 , x_3 e x_4 são variáveis lógicas (booleanas ou binárias) os operadores \rightarrow , \vee , \leftrightarrow e \wedge são os operadores binários “implicação”, OR, “se e somente se” e AND, respectivamente, e \sim é o operador unário NOT.

A fórmula ϕ tem a seguinte atribuição $x_1 = 0$, $x_2 = 0$, $x_3 = 1$ e $x_4 = 1$ que a satisfaz, ou seja, com as entradas descritas o resultado da fórmula é 1. Essa atribuição é um certificado que satisfaz a fórmula ϕ . Veja que, dessa forma, o problema da satisfazibilidade de fórmulas pertence à classe NP se existir um algoritmo de decisão em tempo polinomial que recebe a codificação de uma fórmula e um certificado e retorna 1 se a fórmula produzir o valor 1 a partir do certificado ou retorna 0 caso contrário.

O algoritmo que decide se um certificado é uma atribuição que satisfaz a codificação da fórmula substitui cada variável pelo seu valor de atribuição correspondente e realiza a avaliação da expressão lógica. Veja que a tarefa descrita pode ser realizada facilmente em tempo polinomial, assim o problema da satisfazibilidade de fórmulas booleanas pertence à classe NP.

O próximo passo consiste em mostrar que qualquer problema de satisfazibilidade de circuitos pode ser reduzido, em tempo polinomial, em um problema de satisfazibilidade de fórmulas. Assim, podemos representar para cada fio x_i do circuito uma variável x_i na fórmula, em seguida expressar o funcionamento de cada porta no circuito como uma cláusula. Considere o circuito representado na Figura 8.

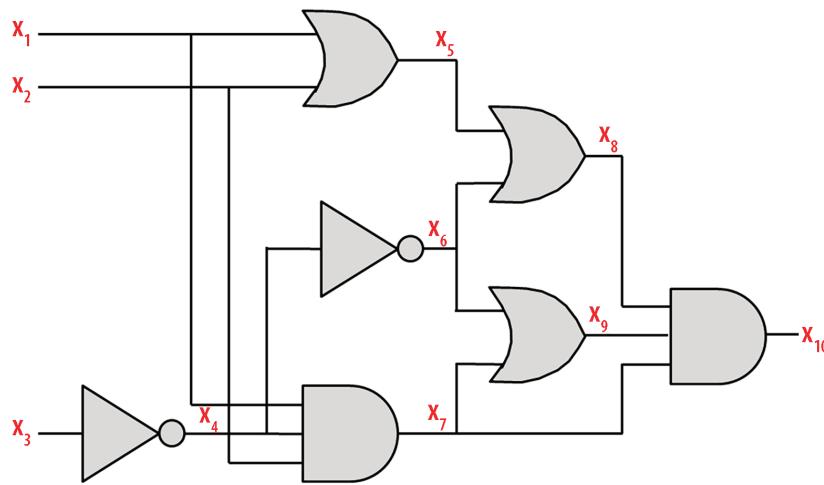


Figura 8 – Redução do problema da satisfazibilidade de circuitos à satisfazibilidade de fórmula

Fonte: Adaptada de CORMEN, 2009

As cláusulas são pequenas fórmulas que descrevem a funcionalidade de cada porta lógica do circuito. Por exemplo, a porta AND de saída do circuito pode ter seu funcionamento descrito pela cláusula $x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)$, ou seja, essa porta produzirá o valor 1 se e somente se ambos os operandos tiverem valores iguais.

Considerando a descrição das cláusulas, o circuito da Figura 8 seria equivalente à seguinte expressão:

$$\begin{aligned}\phi = & x_{10} \wedge (x_4 \leftrightarrow \sim x_3) \\ & \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \sim x_4) \\ & \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\ & \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))\end{aligned}$$

Desse modo, a redução de um circuito para uma fórmula é realizada em tempo polinomial. Portanto, o problema da satisfazibilidade de fórmulas booleanas é pertinente à classe NP-difícil.

A partir da redução de um problema em outro se pode mostrar a pertinência de diferentes problemas à classe NP-completa. Como exemplo, podemos considerar problemas no âmbito da lógica booleana, grafos, aritmética, projeto de redes, conjuntos e partições, dentre outros possíveis de serem listados.

Material Complementar

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

▶ Vídeos

Projeto e Análise de Algoritmos – Aula 14 – Classes de problemas: Problemas P, NP e NP-completos

<https://youtu.be/wsvXnmUHJX8>

Isto é Matemática T07E02 O problema do Caixeiro Viajante

https://youtu.be/_vKMyRj855A

Problemas NP-completos

<https://youtu.be/VJucgdXGHyE>

Classes de problemas – P e NP

<https://youtu.be/ksspU2DXNJ4>

Referências

CORMEN, T. H. *et al.* **Introduction to algorithms**. MIT press, 2009.

TOSCANI, L. V. **Complexidade de algoritmos**, v. 13: UFRGS. 3^a edição. Porto Alegre: Bookman, 2012. (*e-book*)

ZIVIANI, N. **Projeto de algoritmos**: com implementações em JAVA e C. São Paulo: Cengage Learning, 2012. (*e-book*)



Cruzeiro do Sul
Educacional