Projeto e Análise de Algoritmos

Flávio L. C. de Moura*

20 de abril de 2022

1 A classe NP

A classe NP consiste dos problemas que podem ser resolvidos em tempo polinomial por um algoritmo não-determinístico. A ideia é que inicialmente, o algoritmo advinhe uma solução (fase não-determinística), e em seguida esta solução deve ser verificada em tempo polinomial deterministicamente. Assim, a forma mais usual de apresentar a classe NP, consiste em considerar os problemas que podem ser verificados em tempo polinomial por um algoritmo determinístico [1, 2].

Como vimos, em alguns casos é possível evitar uma abordagem força bruta e encontrar soluções polinomiais para o problema em questão. Mas é fácil imaginar que isto nem sempre será possível. De fato, veremos que existem diversos problemas interessantes/importantes para os quais soluções polinomiais não foram encontradas até hoje, mas que ainda é possível verificar em tempo polinomial, dado um certificado.

O problema de encontrar ciclos Hamiltonianos em grafos (não dirigidos) tem sido estudado por muito tempo (mais de 100 anos!). Formalmente, um ciclo Hamiltoniano de um grafo $G = (V, E)$ é um ciclo simples que contém cada vértice de V, i.e. cada vértice de G é visitado uma única vez. Um grafo que contém um ciclo Hamiltoniano é dito Hamiltoniano.

Denotaremos por HAM-CYCLE o problema de encontrar ciclos Hamiltonianos em grafos.

HAM-CYCLE = $\{\langle G\rangle : G$ é um grafo Hamiltoniano $\}$

Podemos verificar uma possível solução em tempo polinomial: Suponha que um colega te diz que um grafo G é Hamiltoniano, e como justificativa, fornece uma sequência de vértices na ordem que ele diz formar um caminho Hamiltoniano.

1. Verifique que os vértices dados constituem o conjunto V dos vértices de G;
2. Verifique que cada par de vértices consecutivos da sequência dada corresponde a uma aresta de G.

Como a verificação acima pode ser feita em tempo polinomial, temos que HAM-CYCLE $\in$ NP.

1.1 Algoritmos de Verificação

Definição 1.1. *Um algoritmo de verificação é um algoritmo A que recebe dois argumentos x e y, e verifica se $A(x, y) = 1$. Uma linguagem verificada por um algoritmo de verificação A é*
$L = \{x \in \{0,1\}^* : \exists y \in \{0,1\}^*$ *tal que* $A(x, y) = 1\}$

Intuitivamente, o algoritmo A verifica a linguagem L se, para cada $x \in L$, existe um certificado y tal que A é usado para provar que $x \in L$. Formalmente, a classe NP é a classe das linguagens que podem ser verificadas em tempo polinomial.

$NP = \{L \subseteq \{0,1\}^* :$ existe um algoritmo determinístico A que verifica L em tempo polinomial$\}$

Mais precisamente, $L \in$ NP se, e somente se, existe um algoritmo polinomial A e uma constante c tais que $L = \{x \in \{0,1\}^* : \exists y, |y| = O(|x|^c)$ tal que $A(x, y) = 1\}$.

*flavio@flaviomoura.info

Agora é fácil ver que HAM-CYCLE $\in$ NP. Basta checar que o caminho dado (sequência de vértices) é uma permutação de V, isto é, cada vértice ocorre apenas uma vez, e que existe uma aresta em G para cada par de vértices consecutivos do caminho dado, e entre o primeiro e último vértices.

Estamos interessados em algoritmos que verificam se uma instância está ou não em uma linguagem. Por exemplo, dada uma instância (G, u, v) do problema de decisão PATH, e um caminho p de u para v, podemos facilmente verificar se p é um caminho em G.

A classe NP consiste dos problemas que podem ser verificados em tempo polinomial, i.e. dado um certificado, podemos verificar em tempo polinomial que este certificado é correto. Por exemplo, considerando o problema dos ciclos hamiltonianos em um digrafo $G = (V, E)$ com $n = |V|$, um certificado pode ser uma sequência $\langle v_1, v_2, \ldots, v_n \rangle$ de vértices que pode ser verificada em tempo polinomial. Para o problema 3-SAT, um certificado pode ser uma designação de valores para as variáveis da fórmula que pode ser verificado (se satisfaz ou não a fórmula dada) em tempo polinomial.

Um clique em um grafo (não dirigido) é um subgrafo onde dois vértices quaisquer estão ligados por uma aresta. Um k-clique é um clique que contém k vértices. O problema CLIQUE consiste em determinar se um grafo contém um clique de um tamanho especificado:

CLIQUE $= \{(G, k) : G$ é um grafo com um k-clique$\}$

Afirmação: CLIQUE $\in$ NP

O clique é o certificado. Para a entrada $((G, k), c)$

1. Verifique se c é um subconjunto de $G.V$ de tamanho k;
2. Verifique se G contém todas as arestas que conectam vértices em c;
3. Se ambas as verificações podem ser feitas então retorne 1, caso contrário, retorne 0.

Teorema 1.2. *3-SAT $\leq_P$ CLIQUE*

Demonstração. Nos grafos a serem construídos, cliques de um tamanho específico correspondem a designações satisfatíveis da fórmula. Seja φ uma fórmula com k cláusulas

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots (a_k \vee b_k \vee c_k)$$

A redução f constrói a codificação $\langle G, k \rangle$ onde G é um grafo não dirigido dado por:

- Os vértices de G são organizados em k grupos de 3 vértices cada $t_1, t_2, \ldots, t_k$. Cada tripla t_i corresponde a uma das cláusulas de φ, e cada vértice na tripla corresponde a um literal da cláusula associada. Marque cada vértice de G com o literal correspondente em φ. As arestas de G conectam todos os vértices exceto:
  - vértices contraditórios, como x e $\neg x$;
  - vértices da mesma tripla.

Afirmação: φ é satisfatível se, e somente se, G possui um k-clique.

Suponha que φ é satisfatível, e portanto cada cláusula possui pelo menos um literal verdadeiro. Em cada tripla em G, selecionamos um vértice correspondente ao literal verdadeiro. Se mais de um literal for verdadeiro na mesma cláusula, escolhemos um deles aleatoriamente. Os vértices selecionados formam um k-clique: o número de vértices selecionados é k, cada par de vértices selecionado está ligado por uma aresta.

Suponha que G possui um k-clique. Nenhum par de vértices do clique ocorre na mesma tripla porque vértices da mesma tripla não são ligados por arestas. Portanto, cada tripla contém exatamente um dos vértices do k-clique. Designamos valores para as variáveis de φ de forma que cada literal que marca um vértice assume valor 1 (verdadeiro). Isto é possível porque vértices contraditórios não são ligados. Esta designação de variáveis satisfaz a fórmula φ porque cada tripla corresponde a um vértice do clique, e portanto cada cláusula de φ tem valor 1. $\square$

2 A classe NPC

Dizemos que um problema é *NP-completo*, i.e. que está na classe NPC, se está na classe NP e é tão difícil quanto qualquer problema em NP. Ao mostrarmos que um problema é NP-completo, não estamos tentando provar a existência de um algoritmo eficiente, mas concluir que a existência de um tal algoritmo é improvável. De fato, estamos concluindo sobre o quão difícil ele é, e não sobre quão fácil como fizemos até então. Portanto, um algoritmo eficiente provavelmente não existe para este problema.

Uma importante questão em aberto é quando P é ou não um subconjunto próprio de NP, o que corresponde ao problema $P \neq NP$ citado anteriormente. Problemas NP-completos normalmente são considerados intratáveis dada a grande quantidade de problemas NP-completos já estudados, e sem solução polinomial encontrada.

Propriedade interessante: Se algum problema NP-completo puder ser resolvido em tempo polinomial então qualquer problema em NP terá solução polinomial.

Seria uma surpresa encontrar uma solução polinomial para algum (e portanto para todos) destes problemas. Neste sentido, se um problema é NP-completo então isto pode ser visto como uma evidência da sua intratabilidade. Neste caso, algoritmos aproximados devem ser considerados, ao invés de soluções rápidas e exatas.

Por que até hoje ninguém conseguiu encontrar soluções polinomiais para estes problemas? Não sabemos, talvez porque elas simplesmente não existam, ou porque elas estejam baseadas em princípios ainda desconhecidos. Qualquer problema em P está também em NP porque pode ser verificada em tempo polinomial pelo mesmo algoritmo que a decide em P sem a necessidade de utilizar certificados.

Definição 2.1. *Uma linguagem $L \subseteq \{0,1\}^*$ é dita **NP-completa** se:*

1. *$L \in NP$;*
2. *$L' \leq_P L, \forall L' \in NP$.*

Denotamos por NPC a classe das linguagens NP-completas.

Se uma linguagem L satisfaz a propriedade 2 acima, mas não necessariamente a propriedade 1, dizemos que L é **NP-difícil**.

Como NP-completude consiste em mostrar quão difícil é um problema, utilizamos a redução polinomial na outra direção para mostrar que um problema é NP-completo:

Para mostrarmos que um problema B não possui solução polinomial, consideremos um problema A, para o qual sabemos não existir solução polinomial. Suponha também que temos um algoritmo de redução que transforma instâncias de A em instâncias de B em tempo polinomial. Agora, se existisse uma solução polinomial para B então poderíamos construir uma solução polinomial para A como acima, contradizendo a suposição de que A não possui solução polinomial.

Lema 2.2. *Se L é uma linguagem tal que $L' \leq_P L$ para algum $L' \in NPC$, então L é NP-difícil. Se adicionalmente, $L \in NP$ então $L \in NPC$.*

Demonstração. Como L' é NP-completo, então $\forall L'' \in$ NP, temos que $L'' \leq_P L'$, e por hipótese temos que $L' \leq_P L$. Logo. $L'' \leq_P L$, o que mostra que L é uma linguagem NP-difícil. Agora, se $L \in$ NP então, por definição temos que $L \in$ NPC. □

O lema anterior nos dá um método para mostrar que uma linguagem L é NP-completa:

1. Prove que $L \in$ NP;
2. Escolha uma linguagem L' que seja NP-completa;
3. Descreva um algoritmo que computa uma função f que mapeia cada instância x de L' em uma instância $f(x) \in L$;
4. Prove que $x \in L'$, se e somente se, $f(x) \in L$;
5. Prove que o algoritmo que computa f é polinomial

Os passos de 2-5 mostram que L é NP-difícil.

SAT $= \{\langle\varphi\rangle : \varphi$ é uma fórmula booleana satisfatível$\}$.

Podemos determinar em tempo exponencial se uma dada fórmula booleana φ contendo n variáveis é satisfatível: basta checar cada uma das 2^n possíveis valorações para as variáveis de φ. Nenhum algoritmo polinomial é conhecido para SAT. O teorema a seguir, mostra que é muito improvável que tal algoritmo exista.

O teorema a seguir é conhecido como o *teorema de Cook-Levin*:

Teorema 2.3. *SAT $\in$ NPC.*

Teorema 2.4. *3-SAT $\in$ NPC.*

Demonstração. 1. 3-SAT $\in$ NP: Dada uma designação de valores para as variáveis de uma 3-FNC fórmula (certificado), o algoritmo de verificação substitui cada variável pelo valor dado e avalia a expressão. Se a expressão resulta em 1 então o certificado é válido e a fórmula é satisfatível.

2. SAT $\leq_P$ 3-SAT.

□

Teorema 2.5. *CLIQUE $\in$ NPC.*

Demonstração. 1. CLIQUE $\in$ NP;

2. 3-SAT $\leq_P$ CLIQUE

□

Referências

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* The MIT Press, 3rd edition, 2009.

[2] Michael Sipser. *Introduction to the Theory of Computation.* International Thomson Publishing, 1st edition, 1996.