Capítulo 1

NP-completude

Praticamente todos os algoritmos estudados até aqui têm complexidade polinomial, i.e. o tempo de execução destes algoritmos no pior caso está em $O(p(n))$, onde $p(n)$ é um polinômio no tamanho da entrada n [2, 3]. Estes algoritmos formam a classe dos algoritmos que são considerados "eficientes". Os problemas que podem ser resolvidos por algoritmos polinomiais são chamados de "tratáveis". No entanto, vimos algoritmos cujos tempos de execução são exponenciais no tamanho da entrada, como por exemplo, a abordagem força bruta para calcular o n-ésimo número de Fibonacci, para descobrir a melhor forma de associar o produto de n matrizes, entre outros.

Neste capítulo estudaremos uma classe de problemas para os quais as técnicas estudadas até aqui não se aplicam. Estes problemas não têm soluções polinomiais conhecidas, mas também não existe uma prova de que tais soluções não sejam possíveis: esta é a famosa questão "P versus NP"que constitui um dos mais interessantes problemas em aberto na Computação. Adicionalmente, essa classe de problemas possui uma característica muito interessante: um algoritmo eficiente que resolva qualquer um dos problemas desta classe resulta de forma imediata em algoritmos eficientes para todos os problema da classe.

Definimos um problema como a seguir:

Definição 1. *Um problema (abstrato) Q é uma relação binária que associa um conjunto I de instâncias a um conjunto S de soluções.*

Por exemplo:

- O problema PATH é uma relação que associa cada instância de um digrafo e dois vértices com um caminho que contém os dois vértices.

- O problema SHORTEST-PATH é uma relação que associa cada instância de um digrafo e dois vértices com um caminho mínimo que contém os dois vértices. Como caminhos mínimos não são necessariamente únicos, uma instância de um problema pode ter mais de uma solução.

Neste capítulo trabalharemos essencialmente com *problemas de decisão*, que são problemas cujas respostas são sempre *sim* ou *não*:

Definição 2. *Um problema de decisão é uma relação binária sobre um conjunto I de instâncias e um conjunto binário (sim ou não) de soluções.*

Assim, podemos ver um problema de decisão como sendo uma função que associa instâncias em I ao conjunto de soluções $\{0, 1\}$.

- O problema PATH pode ser visto como um problema de decisão: Dados um digrafo G, e vértices u e v de G, determinar se existe um caminho de u para v em G.

Daqui em diante, trataremos apenas de problemas de decisão, que chamaremos apenas de problemas.

1.1 A classe P

A classe P consiste dos problemas que podem ser resolvidos em tempo polinomial por algoritmos determinísticos.

Afirmação. PATH está em P

Prova. Considere o seguinte algoritmo que recebe como entrada um digrafo G e vértices u e v.

1. Marque o vértice u

2. Enquanto existir aresta $(a, b) \in G$ com a marcado, e b não-marcado, marque b.

3. Se v está marcado retorne 1, caso contrário retorne 0.

Análise do algoritmo: O laço da linha 2 pode ser executado segundo o algoritmo de busca em largura, por exemplo, que é linear no tamanho da representação do grafo G. As outras linhas do algoritmo são executadas apenas uma vez, portanto o algoritmo é polinomial.

1.1.1 Redução polinomial

A *redutibilidade* entre dois problemas é uma técnica de projeto de algoritmos muito utilizada para dar informações sobre a dificuldade de problemas. Por exemplo, considere um problema A que queremos resolver, e suponha que sabemos como resolver um outro problema B. Se for possível transformar instâncias do problema A em instâncias do problema B com as seguintes características:

- A transformação é feita em tempo polinomial;

- As respostas são as mesmas para ambas as instâncias.

Então chamamos este procedimento de *algoritmo de redução*, e o mesmo nos fornece uma forma de resolver o problema A a partir do problema B:

- Dada uma instância α do problema A, usamos o algoritmo de redução para transformá-la em uma instância β do problema B;

- Executamos o algoritmo polinomial para a instância β de B;

- Usamos a resposta de β como resposta de α.

Ou seja, podemos assim construir um algoritmo para o problema A. O que podemos concluir a partir da existência de um tal algoritmo de redução? A existência de um algoritmo eficiente para B nos permite construir um algoritmo eficiente para A, mas também que a não existência de um algoritmo eficiente para A implica na não existência de um algoritmo eficiente para B. Em outras palavras, o problema B é tão difícil quanto o problema A, ou ainda, A não é mais difícil do que B.

Definição. Dizemos que um problema L_1 é redutível polinomialmente ao problema L_2, notação $L_1 \leq_p L_2$, se existe uma função computável em tempo polinomial $f : \{0,1\}^* \to \{0,1\}^*$ tal que para todo $x \in \{0,1\}^*$, $x \in L_1$ se, e somente se, $f(x) \in L_2$. A função f é chamada de *função de redução*, e o algoritmo polinomial F que computa a função f é chamado de *algoritmo de redução.*

Observe que ao reduzirmos uma linguagem (ou problema) L_1 para outra linguagem (ou problema) L_2, queremos que cada instância de L_1 seja transformada em uma instância de L_2, isto é, se $x \in L_1$ então $f(x) \in L_2$.

Adicionalmente, precisamos que elementos que não sejam instância de L_1 não sejam levados em L_2: $x \notin L_1$ então $f(x) \notin L_2$, o que é equivalente por contraposição a se $f(x) \in L_2$ então $x \in L_1$. Juntando estas duas informações temos a equivalência "$x \in L_1$ se, e somente se, $f(x) \in L_2$"da definição acima.

Reduções polinomiais são uma ferramenta poderosa que nos permitem provar que outras linguagens estão em P:

Lema. Sejam $L_1, L_2 \in \{0,1\}^*$ linguagens tais que $L_1 \leq_p L_2$, então $L_2 \in P$ implica que $L_1 \in P$.

Prova. Seja A_2 um algoritmo polinomial que decide a linguagem L_2, e F um algoritmo de redução polinomial que computa a função de redução f. Construiremos um algoritmo A_1 que decide L_1 da seguinte forma: dado $x \in \{0,1\}^*$, o algoritmo A_1 inicialmente usa F para transformar x em $f(x)$, e então usa o algoritmo A_2 para responder. Note que A_1 é polinomial já que tanto A_2 quanto F são polinomiais.

Considere o problema 2-SAT, cujas instâncias são expressões lógicas formadas por conjunções de disjunções de dois literais, onde um literal é uma variável booleana ou a negação de uma variável booleana. Por exemplo, a expressão a seguir é uma instância de 2-SAT:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (x_1 \vee x_2)$$

Uma solução da instância acima é uma designação de valores booleanos (0 ou 1) para as variáveis que satisfaçam a expressão, ou seja, que retornem 1. Por exemplo, a designação $x_1 = 1$, $x_2 = 1$ e $x_3 = 0$ satisfaz a expressão acima.

Exercício. Mostre que 2-SAT $\in$ P.

1.2 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, 4].

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.

1.2.1 Exemplo

O problema de encontrar ciclos Hamiltonianos em (di)grafos 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 (di)grafo que contém um ciclo Hamiltoniano é dito Hamiltoniano.

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

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

Podemos verificar uma possível solução em tempo polinomial: Suponha que um colega te diz que um (di)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.

1.2.2 Exemplo

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.

1.2.3 Exemplo

Afirmação. 3-SAT $\leq_P$ CLIQUE

Prova. 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.

Referências Bibliográficas

[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] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, Cambridge, MA, USA, 4 edition, April 2022.

[3] A. V. Levitin. *Introduction to the Design and Analysis of Algorithms, Third Edition.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2012.

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