Projeto e Análise de Algoritmos

Flávio L. C. de Moura*

14 de abril de 2022

1 NP-completude

Praticamente todos os algoritmos estudados até aqui são polinomiais, 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[1, 2]. 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, busca em grafos utilizando força bruta. De fato, suponha que, dados um digrafo G e vértices u e v de G, queremos determinar quando existe um caminho de u para v em G. Utilizando um algoritmo força bruta, precisamos considerar todos os potenciais caminhos em G, e determinar quando algum deles é um caminho de u para v. Um potencial caminho é uma sequência de vértices de G de tamanho menor ou igual a $|V|$ (o número de vértices de G). O número de potenciais caminhos é $|V|^{|V|}$, e portanto exponencial.

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 prova de que tais soluções não existam: esta é a famosa questão "P versus NP"que constitui um dos mais interessantes problemas em aberto na Computação. Os problemas pertencentes a esta classe são chamados NP-completos.

Formalmente, definimos um problema como a seguir:

Definição 1.1. *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 1.2. *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 $\{0, 1\}$. 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. Este problema 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.

1.1 A classe P

A classe P consiste dos problemas que podem ser resolvidos em tempo polinomial por um algoritmo determinístico.

Teorema 1.3. *PATH está em P*

*flavio@flaviomoura.info

Demonstração. 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* (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 Reduções em tempo polinomial

A ideia de que um problema não é mais fácil e nem mais difícil que outro também se aplica quando ambos são problemas de decisão. Por exemplo, considere um problema A que queremos resolver em tempo polinomial. Suponha que sabemos como resolver um outro problema B em tempo polinomial, e que temos um procedimento que transforma 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.

Chamamos este procedimento de *algoritmo de redução*, que nos fornece uma forma de resolver o problema A em tempo polinomial:

- 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.

Como cada um destes passos é realizado em tempo polinomial, todo o algoritmo também é realizado em tempo polinomial, e portanto temos uma forma de decidir A em tempo polinomial.

1.1.2 Linguagem formal

Um problema de decisão pode ser visto como um problema de reconhecimento de linguagem: seja U o conjunto de todas as entradas possíveis para o problema de decisão. Seja $L \subseteq U$, o conjunto de todas as entradas para as quais a resposta do problema é *sim*. Chamamos L a *linguagem* correspondente ao problema, e utilizamos os termos *problema* e *linguagem* de forma intercambiável. O problema de decisão deve então reconhecer se uma dada entrada pertence ou não à linguagem L.

Definição 1.4. *Seja Σ um conjunto finito (de símbolos) que chamaremos de alfabeto. O conjunto de todas as palavras (*strings*) que podem ser formadas com os símbolos de Σ é denotado por Σ^*. Uma linguagem L sobre Σ é qualquer subconjunto de Σ^*. O alfabeto vazio é denotado por ϵ, enquanto que a linguagem vazia é denotada por $\emptyset$.*

Diversas operações podem ser realizadas sobre linguagens: união, interseção, complemento, concatenação e fechamento. Do ponto de vista da teoria de linguagens formais, o conjunto das instâncias de qualquer problema de decisão Q é simplesmente o conjunto Σ^*, onde $\Sigma = \{0, 1\}$. Como o problema Q é caracterizado pelas instâncias que produzem resposta 1 (*sim*), podemos ver a linguagem L sobre $\Sigma = \{0, 1\}$ como sendo $L = \{x \in \Sigma^* \mid Q(x) = 1\}$.

Definição 1.5. *Dizemos que um algoritmo A aceita a palavra $x \in \{0, 1\}^*$ se $A(x) = 1$, i.e. a resposta do algoritmo A ao receber a entrada x é igual a 1. A linguagem aceita pelo algoritmo A é o conjunto das palavras aceitas pelo algoritmo: $L = \{x \in \{0, 1\}^* \mid A(x) = 1\}$. Dizemos que o algoritmo A rejeita a palavra x, se $A(x) = 0$.*

Note que, mesmo que a linguagem L seja aceita pelo algoritmo A, isto não significa que A rejeita uma palavra $x \notin L$ porque, por exemplo, A pode entrar em *loop*.

Definição 1.6. *Uma linguagem L é decidida pelo algoritmo A, se A aceita toda palavra em L, e rejeita toda palavra que não está em L. Uma linguagem L é aceita em tempo polinomial pelo algoritmo A, se A aceita L, e se existe uma constante k tal que para qualquer palavra $x \in L$ de comprimento n, o algoritmo A aceita x em tempo $O(n^k)$, para algum $k \geq 0$. Uma linguagem L é decidida em tempo polinomial pelo algoritmo A se existir uma constante não-negativa k tal que para qualquer palavra $x \in \{0,1\}^*$, o algoritmo decide em tempo $O(n^k)$ se $x \in L$.*

Assim, para aceitar uma linguagem, o algoritmo precisa apenas produzir uma resposta para toda palavra de L, mas para decidir uma linguagem, o algoritmo precisa aceitar ou rejeitar toda palavra em $\{0,1\}^*$. Como exemplo, o problema de decisão PATH corresponde a seguinte linguagem:

PATH $= \{\langle G, u, v, k\rangle : G = (V, E)$ é um grafo (não dirigido), $u, v \in V, k \geq 0$ é um inteiro, e existe um caminho de u para v em G contendo, no máximo, k arestas$\}$. Assim, a linguagem PATH pode ser aceita em tempo polinomial pelo seguinte algoritmo: Dada uma instância (G, u, v), o algoritmo executa BFS para computar o menor caminho de u para v, e então compara o número de arestas deste menor caminho com k. Se o menor caminho possui até k arestas então o algoritmo retorna 1, e para. Caso contrário, o algoritmo entra em *loop*. Note que este algoritmo não decide PATH. Neste caso, para decidir PATH basta que o algoritmo retorne 0 e pare, ao invés de entrar em *loop*. Para outros problemas, como o *problema da parada para máquinas de Turing*, existem algoritmos de aceitação, mas não de decisão.

Por fim, definimos a classe P em termos de linguagens:

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

Definição 1.7. *Dizemos que uma linguagem L_1 é redutível polinomialmente à linguagem 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*.*

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

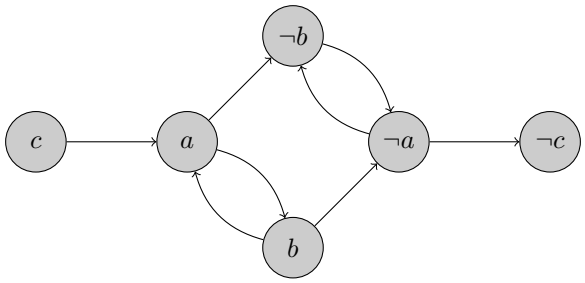
Lema 1.8. *Sejam $L_1, L_2 \in \{0,1\}^*$ linguagens tais que $L_1 \leq_p L_2$, então $L_2 \in P$ implica que $L_1 \in P$.*

Demonstração. 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.



Teorema 1.9. *2-SAT $\in P$.*

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.