# Introdução à Teoria da Computação Exercícios

Livro: Michel Sipser, Introdução à Teoria da Computação – 2ª Ed. – Capítulo 07

Obs: Exercícios 7.7 e 7.20 estão apresentados em versões simplificadas.

## EXERCÍCIOS

**7.1** Responda VERDADEIRO ou FALSO para cada parte.

**a.** $2n = O(n)$.
**b.** $n^2 = O(n)$.
**[R]c.** $n^2 = O(n \log^2 n)$.
**[R]d.** $n \log n = O(n^2)$.
**e.** $3^n = 2^{O(n)}$.
**f.** $2^{2^n} = O(2^{2^n})$.

**7.2** Responda VERDADEIRO ou FALSO para cada parte.

**a.** $n = o(2n)$.
**b.** $2n = o(n^2)$.
**[R]c.** $2^n = o(3^n)$.
**[R]d.** $1 = o(n)$.
**e.** $n = o(\log n)$.
**f.** $1 = o(1/n)$.

**7.3** Quais dos seguintes pares de números são primos entre si? Mostre os cálculos que levaram às suas conclusões.

**a.** 1274 e 10505
**b.** 7289 e 8029

**7.4** Preencha a tabela descrita no algoritmo de tempo polinomial para reconhecimento de linguagem livre-do-contexto do Teorema 7.16 para a cadeia $w =$ `baba` e a GLC $G$:

$$S \to RT$$
$$R \to TR \mid \mathtt{a}$$
$$T \to TR \mid \mathtt{b}$$

**7.5** A fórmula a seguir é satisfazível?

$$(x \vee y) \wedge (x \vee \overline{y}) \wedge (\overline{x} \vee y) \wedge (\overline{x} \vee \overline{y})$$

**7.6** Mostre que P é fechada sob união, concatenação e complementação.

**7.7** Mostre que NP é fechada sob união

**7.8** Seja $CONEXO = \{\langle G \rangle \mid G$ é um grafo não-direcionado conexo$\}$. Analise o algoritmo dado na página 165 para mostrar que essa linguagem está em P.

**7.9** Um ***triângulo*** em um grafo não-direcionado é um 3-clique. Mostre que $TRIÂNGULO \in$ P, onde $TRIÂNGULO = \{\langle G \rangle \mid G$ contém um triângulo$\}$.

**7.10** Mostre que $TODAS_{\mathsf{AFD}}$ está em P.

**7.11** Chame os grafos $G$ e $H$ ***isomorfos*** se os nós de $G$ podem ser reordenados de modo que ele fique idêntico a $H$. Seja $ISO = \{\langle G, H \rangle \mid G$ e $H$ são grafos isomorfos$\}$. Mostre que $ISO \in$ NP.

## PROBLEMAS

**7.12** Seja

$$EXPMOD = \{\langle a, b, c, p \rangle \mid a, b, c \text{ e } p \text{ são inteiros em binário tais que } a^b \equiv c \pmod{p}\}.$$

Mostre que $EXPMOD \in$ P. (Note que o algoritmo mais óbvio não roda em tempo polinomial. Dica: Tente primeiro com $b$ sendo uma potência de 2.)

**7.13** Uma ***permutação*** sobre o conjunto $\{1, \ldots, k\}$ é uma função bijetora sobre o mesmo. Quando $p$ é uma permutação, $p^t$ denota a composição de $p$ com si mesma $t$ vezes. Seja

$$POT\text{-}PERM = \{\langle p, q, t \rangle \mid p = q^t \text{ onde } p \text{ e } q \text{ são permutações sobre } \{1, \ldots, k\} \text{ e } t \text{ é um inteiro em binário}\}.$$

Mostre que $POT\text{-}PERM \in$ P. (Note que o algoritmo mais óbvio não roda em tempo polinomial. Dica: Tente primeiro com $t$ sendo uma potência de 2.)

**7.14** Mostre que P é fechada sob a operação estrela. (Dica: Use programação dinâmica. Sobre uma entrada $y = y_1 \cdots y_n$ for $y_i \in \Sigma$, construa uma tabela que indique, para cada $i \leq j$, se a subcadeia $y_i \cdots y_j \in A^*$ para qualquer $A \in$ P.)

**R7.15** Mostre que NP é fechada sob a operação estrela.

**7.16** Seja *SOMA-SUBCU* o problema da soma de subconjuntos no qual todos os números são representados em unário. Por que a prova de NP-completude para *SOMA-SUBC* falha em mostrar que *SOMA-SUBCU* é NP-completa? Mostre que *SOMA-SUBCU* $\in$ P.

**7.17** Mostre que, se P = NP, então toda linguagem $A \in$ P, exceto $A = \emptyset$ e $A = \Sigma^*$, é NP-completa.

*7.18 Mostre que *PRIMOS* $= \{m |\ m$ é um número primo em binário$\} \in$ NP. (Dica: Para $p > 1$ o grupo multiplicativo $Z_p^* = \{x |\ x$ e $p$ são primos entre si e $1 \leq x < p\}$ é, ambos, cíclico e de ordem $p - 1$ sse $p$ é primo. Você pode usar esse fato sem justicá-lo. A afirmativa mais forte *PRIMOS* $\in$ P é atualmente conhecida como verdadeira, mas é mais difícil de provar.)

**7.19** Em geral, acredita-se que *CAM* não é NP-completa. Explique a razão por trás dessa crença. Mostre que provar que *CAM* não é NP-completa provaria que P $\neq$ NP.

**7.20** Suponha que $G$ represente um grafo não-direcionado. Seja também

$$CAMMÁX = \{\langle G, a, b, k\rangle |\ G \text{ contém um caminho simples de comprimento no máximo } k \text{ de } a \text{ para } b\},$$

e

$$CAMMIN = \{\langle G, a, b, k\rangle |\ G \text{ contém um caminho simples de comprimento no mínimo } k \text{ de } a \text{ para } b\}.$$

**a.** Mostre que *CAMMÁX* $\in$ P.

**b.** Mostre que *CAMMIN* $\in$ NP

**7.21** Seja *DUPLO-SAT* $= \{\langle\phi\rangle |\ \phi$ tem pelo menos duas atribuições que a satisfazem$\}$. Mostre que *DUPLO-SAT* é NP-completa.

R7.22 Seja *MEIO-CLIQUE* $= \{\langle G\rangle |\ G$ é um grafo não-direcionado que tem um subgrafo completo com pelo menos $m/2$ nós, onde $m$ é o número de nós em $G\}$. Mostre que *MEIO-CLIQUE* é NP-completa.

**7.23** Seja $FNC_k = \{\langle\phi\rangle |\ \phi$ é uma fnc-fórmula satisfazível em que cada variável ocorre em, no máximo, $k$ posições$\}$.

**a.** Mostre que $FNC_2 \in$ P.

**b.** Mostre que $FNC_3$ é NP-completa.

**7.24** Seja $\phi$ uma 3fnc-fórmula. Uma ***≠-atribuição*** às variáveis de $\phi$ é aquela em que cada cláusula contém dois literais com diferentes valores-verdade. Em outras palavras, uma ≠-atribuição satisfaz $\phi$ sem atribuir verdadeiro a três literais em qualquer cláusula.

**a.** Mostre que a negação de qualquer ≠-atribuição a $\phi$ é também uma ≠-atribuição.

**b.** Seja *≠SAT* a coleção de 3fnc-fórmulas que têm uma ≠-atribuição. Mostre que obtemos uma redução em tempo polinomial de *3SAT* para *≠SAT* substituindo cada cláusula $c_i$

$$(y_1 \vee y_2 \vee y_3)$$

## SOLUÇÕES SELECIONADAS

**7.1** **(c)** FALSO; **(d)** VERDADEIRO.

**7.2** **(c)** VERDADEIRO; **(d)** VERDADEIRO.

**7.15** Seja $A \in$ NP. Construa a MTN $M$ para decidir $A$ em tempo polinomial não-determinístico.

$M$ = "Sobre a entrada $w$:

1. Não-deterministicamente divida $w$ em pedaços $w = x_1 x_2 \cdots x_k$.
2. Para cada $x_i$, adivinhe não-deterministicamente os certificados que mostrem que $x_i \in A$.
3. Verifique todos os certificados se possível e, então, *aceite*. Caso contrário, se a verificação falhar, *rejeite*."

**7.22** Vamos dar uma redução por mapeamento polinomial de *CLIQUE* para *MEIO-CLIQUE*. A entrada para a redução é um par $\langle G, k \rangle$ e a redução produz o grafo $\langle H \rangle$ como saída, onde $H$ é como segue. Se $G$ tem $m$ nós e $k = m/2$, então $H = G$. Se $k < m/2$, então $H$ é o grafo obtido de $G$ pelo acréscimo de $j$ nós, cada um conectado a todos os nós originais e aos outros $j - 1$, onde $j = m - 2k$. Assim, $H$ tem $m + j = 2m - 2k$ nós. Observe que $G$ tem um $k$-clique sse $H$ tem um clique de tamanho $k + j = m - k$ e, portanto, $\langle G, k \rangle \in$ *CLIQUE* sse $\langle H \rangle \in$ *MEIO-CLIQUE*. Se $k > 2m$, então $H$ é o grafo obtido pela adição de $j$ nós a $G$ sem quaisquer arestas adicionais, onde $j = 2k - m$. Assim, $H$ tem $m + j = 2k$ nós e, logo, $G$ tem um $k$-clique sse $H$ tem um clique de tamanho $k$. Portanto, $\langle G, k_k \rangle \in$ *CLIQUE* sse $\langle H \rangle \in$ *MEIO-CLIQUE*. Precisamos mostrar também que *MEIO-CLIQUE* $\in$ NP. O certificado é simplesmente o clique.

## Dicas para alguns exercícios:

**Exercício 7.6:**
A ideia é simples: se temos um decisor determinístico de tempo polinomial $M_1$ para a linguagem $A_1$ e um decisor determinístico de tempo polinomial $M_2$ para a linguagem $A_2$, então podemos construir decisores determinísticos de tempo polinomial para $A_1 \cup A_2$, $A_1 \circ A_2$ e $A_1^C$.
A ideia é descrever em alto nível, para cada uma dessas operações, o respectivo decisor (que usará $M_1$ e/ou $M_2$ em sua construção) e discutir sua respectiva complexidade de tempo.

Exemplo: Vamos demonstrar que a intersecção de duas linguangens em P também é uma linguagem em P. Sejam $A_1, A_2 \in P$. Denote por $M_1$ e $M_2$ seus respectivos decisores determinísticos, com complexidades de tempo $f_1(n) = O(n^{k_1})$ e $f_2(n) = O(n^{k_2})$, respectivamente. A seguinte MT decide $A = A_1 \cap A_2$:

$M$ = "Sobre a entrada $w$:

1. Execute $M_1$ sobre $w$;
2. Execute $M_2$ sobre $w$;
3. Se ambas aceitarem, *aceite*; caso contrário, *rejeite*."

É imediato que *M* é um decisor para $A_1 \cap A_2$, já que *M* somente aceita uma cadeia *w* se esta for que seja simultaneamente aceita por $M_1$ e $M_2$ (ou seja, $w \in A_1 \cap A_2$). Mostremos agora que *M* tem custo de tempo polinomial, analisando em alto nível o custo de cada passo. O passo 1 tem custo $O(n^{k_1})$, o passo 2 tem custo $O(n^{k_2})$ e o passo 3 tem custo constante $O(1)$. Logo, o custo total de tempo de *M* será $O(n^{k_1}) + O(n^{k_2}) + O(1) = \max(n^{k_1}, n^{k_2}, 1) = O(n^{\max(k_1,k_2)})$, provando assim que $A_1 \cap A_2 \in P$.

**Exercício 7.7:**
Mostre que, se temos um verificador de tempo polinomial $V_1$ para a linguagem $A_1$ e um verificador de tempo polinomial $V_2$ para a linguagem $A_2$, então podemos construir um verificador de tempo polinomial para $A_1 \cup A_2$.
Algumas ideias do Exercício 6.6 podem ajudar aqui.

**Exercício 7.9:**
Descreva, em alto nível (ou seja, nível similar aos algoritmos apresentados no livro do Sipser), um algoritmo de "força bruta" que explore todas as combinações possíveis de 3 vértices, verificando, para cada combinação, se os 3 vértices correspondentes formam um triângulo (3-clique).
Denotando por $n$ a quantidade de vértices e por $m$ a quantidade de arestas no grafo, você deve mostrar que:

- O número de combinações de 3 vértices possíveis é $O(n^3)$ – para mostrar isso, basta simplificar o coeficiente binomial e notar que $n(n-1)(n-2) \leq n^3 = O(n^3)$.
- Para cada combinação de 3 vértices testada, leva-se, no pior caso, tempo $O(m) = O(n^2)$ (pois o número máximo de arestas em um grafo não direcionado é $O(n^2)$);
- Calcule, em notação assintótica (ou seja, em notação "O" grande), o custo final do algoritmo (dado pelo número de combinações testadas vezes o custo de testar se cada combinação forma um triângulo) e argumente que esse custo é polinomial.

Atenção: O argumento acima pode dar a falsa impressão de que a linguagem

$$CLIQUE = \{\langle G, k\rangle | G \text{ é um grafo não-direcionado com um } k\text{-clique}\}$$

está em P (o que parece contradizer o texto da Seção 7.3). A diferença entre as linguagens *TRIANGULO* e *CLIQUE* é que, na primeira o valor de *k* é constante (e portanto a complexidade é exponencial em *k*, mas polinomial em *n*); já na segunda linguagem, o valor de *k* pode varia livremente entre *1* e *n*. Por exemplo, se $k = \lfloor n/2 \rfloor$, então o número de combinações necessárias de vértices a testar seria $O\left(n^{n/2}\right) = O(n^n)$!
Logo, $CLIQUE \in NP$.

**Exercício 7.20(a):**
Note que *CAMMÁX* é uma variante da linguagem *CAM*, e portanto seria possível apresentar uma modificação da MT que decide a linguagem *CAM*. Essa modificação consistiria em fazer uma busca em largura que somente marcaria novos vértices localizados até uma distância máxima *k* de *s*. Em seguida bastaria verificar se o vértice *t* foi ou não marcado.

Argumente por que essa variação tem custo de tempo polinomial.

**Exercício 7.20(b):**
Como pede-se para mostrar que $CAMMIN \in NP$, deve-se demonstrar que há um verificador de tempo polinomial para *CAMMIN*. Para isso:

a) Indique qual seria o certificado *c* (ou seja, qual informação seria suficiente para verificar se um grafo *G* tem um caminho simples de comprimento no mínimo *k* de a para *b*)
b) Apresente em alto nível um verificador, que receberá como entrada $\langle G, a, b, k \rangle$ e o certificado *c*, e testará se o certificado *c* demonstra $\langle G, a, b, k \rangle \in CAMMIN$. É necessário argumentar por que este verificador tem custo de tempo polinomial (discuta a complexidade de cada passo do verificador, em termos do número de vértices do grafo).

## Exercícios adicionais recomendados:

1) Mostre que $V_{AFD} \in P$.

2) Mostre que $V_{GLC} \in P$.
Dica: No Capítulo 4 foi apresentado o decisor para $V_{GLC}$. Argumente por que aquele decisor tem complexidade de tempo polinomial na representação da gramática de entrada.

3) Mostre que $A_{AFN} \in P$.
Dica: A solução trivial é converter o AFN *N* de entrada para um AFD *M*, e em seguida simular o AFD *M* sobre a cadeia *w*. Embora a conversão de AFN para AFD tenha custo exponencial no número de nós do AFN original, tipicamente considera-se que a cadeia *w* pode ser muito maior do que a descrição do AFN *N*. (Isso é bastante razoável, pois é bastante comum realizar busca por substrings ou por expressões regulares em bases com milhares ou milhões de caracteres, que são várias ordens de grandeza maiores do que as representações dos respectivos autômatos reconhecedores).
Assim, o que interessa é mostrar que o custo de determinar se *w* é ou não aceito por *N* é polinomial no comprimento de *w*.

4) Mostre que $EQ_{AFD} \in P$.

5) Seja $GLC_{\varepsilon}$ a linguagem de todas as GLCs que geram a cadeia vazia. Mostre que $GLC_{\varepsilon} \in P$.
Dica: Há dois algoritmos possíveis: o primeiro seria uma variante do decisor para $V_{GLC}$. O segundo algoritmo possível seria converter a gramática de entrada *G* para uma GLC *G´* na Forma Normal de Chomsky, e verificar se a variável inicial $S_0$ de *G´* tem uma regra $S_0 \rightarrow \varepsilon$. Se sim, aceite; caso contrário, rejeite.
Justifique que esse algoritmo tem custo de tempo polinomial (não esqueça de incluir o tempo da conversão da gramática original para a Forma Normal de Chomsky).

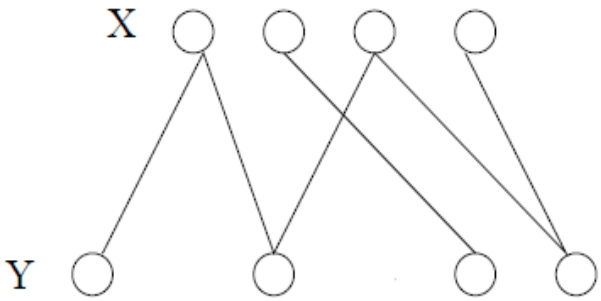
6) Um *grafo bipartido* $G = (V, A)$ é um grafo no qual o conjunto de vértices $V$ pode ser dividido em dois subconjuntos distintos $X$ e $Y$, tal que cada vértice $e \in A$ possui uma extremidade em $X$ e outra extremidade em $Y$.
Um *emparelhamento* em $G$ é um subconjunto das arestas tal que cada vértice aparece no máximo em uma aresta de $G$ (ou seja, duas arestas do emparelhamento não podem incidir sobre o mesmo vértice).
Um *emparelhamento perfeito* em $G$ é um emparelhamento no qual cada nó do grafo tem exatamente uma aresta incidente sobre ele. A figura abaixo ilustra um grafo bipartido (esquerda) e um emparelhamento perfeito correspondente (direita).

Considere a seguinte linguagem e responda às questões abaixo:

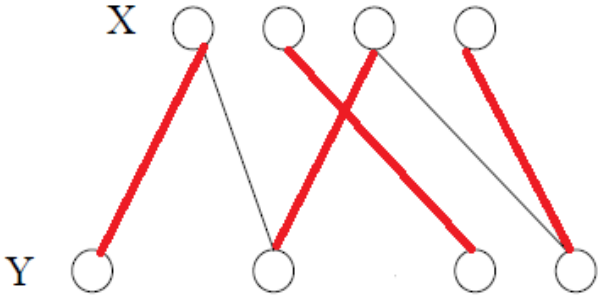
$$EMP = \{\langle G \rangle | G \text{ é um grafo bipartido não orientado que possui um emparelhamento perfeito}\}$$

a) Indique qual seria o certificado da linguagem (ou seja, se queremos testar se um grafo bipartido tem um emparelhamento perfeito, o que seria suficiente apresentar?) e descreva em alto nível um verificador para este problema.

b) Faça uma breve análise da complexidade de tempo do verificador apresentado, e indique se $EMP \in NP$ (ou seja, se $EMP$ é polinomialmente verificável).




Exemplo de grafo bipartido




Emparelhamento perfeito no grafo bipartido (arestas vermelhas)