

# UNIVERSIDADE FEDERAL DE UBERLÂNDIA

## Faculdade de Computação

Disciplina : Teoria da Computação 1 - 1<sup>o</sup> Semestre 2008

Professora : Sandra Aparecida de Amo

### Lista de Problemas n<sup>o</sup> 3

Um pequeno RESUMO dos conceitos e técnicas cujo conhecimento é necessário para resolução da lista

Para se provar que um problema  $X$  pertence à classe  $P$  é preciso exibir um algoritmo (máquina de Turing DETERMINISTA simples a uma fita que sempre pára para qualquer input) que decide o problema em tempo  $O(n^k)$  para algum  $k \geq 0$ .

A complexidade em tempo de uma máquina de Turing  $M$  é uma função  $f : \mathbb{N} \rightarrow \mathbb{N}$  que para cada número natural  $n$  retorna o número  $f(n)$  calculado da seguinte maneira:

- (1) considera-se todos os strings de tamanho  $n$
- (2) considera-se o número de passos que  $M$  executa sobre um string de tamanho  $n$  até parar e aceitar ou rejeitar o string.
- (3)  $f(n)$  = o maior número de passos para todos os strings de tamanho  $n$

Repare que a função  $f$  depende do COMPRIMENTO do input. Assim, se o input é um número natural  $m$ , a função  $f$  atua sobre o tamanho da representação do número  $n$  numa base  $b > 1$ ; normalmente considera-se a representação binária.

**Notação  $O$  (Big O):** Sejam duas funções  $f$  e  $g$  definidas sobre os números naturais, com valores em  $\mathbb{R}^+$ . Dizemos que  $f$  é  $O(g(n))$  se existem números naturais  $c$  e  $n_0$  tais que para todo  $n \geq n_0$  temos:

$$f(n) \leq cg(n)$$

Isto significa que para  $n$  suficientemente grande,  $f(n)$  está abaixo (ou é igual) de um múltiplo constante de  $g(n)$ , isto é,  $cg(n)$  é um limitante superior de  $f(n)$  para  $n \geq n_0$ . Por exemplo:  $f(n) = 2n^2 + 3n + 2$  é  $O(n^2)$ , pois:

$$\begin{aligned} 3n + 2 &= 3\left(n + \frac{2}{3}\right) \\ &\leq 3(n + n) \text{ para } n \geq 1 \\ &= 6n \\ &\leq n^2 \text{ para } n \geq 6 \end{aligned}$$

Logo,  $f(n) \leq 3n^2$  para  $n \geq 6$ . Portanto, provamos que  $f(n)$  é  $O(n^2)$ .

**Notação  $o$  (Small o)**

Dizer que uma função  $f(n)$  é  $o(g(n))$  (repare o  $o$  pequeno), significa que para qualquer constante real  $c$ , existe um  $n_0$  tal que a partir daí  $f(n)$  com certeza está **abaixo** de  $cg(n)$  (nunca mais é

igual a  $cg(n)$ , sempre é menor). Por exemplo,  $f(n) = 2n^2 + 3n + 2$  é  $o(n^3)$ , pois:

$$2n^2 + 3n + 2 = n^2\left(2 + \frac{3}{2n} + \frac{1}{n^2}\right).$$

É fácil mostrar que para todo número real  $c > 0$  existe um  $n_0$  tal que se  $n \geq n_0$ :  $2 + \frac{3}{2n} + \frac{1}{n^2} < cn$ . Logo, para  $n \geq n_0$  temos que  $f(n) < cn^3$ .

Os conceitos de ordem assintótica  $O$  e ordem assintótica  $o$  permitem comparar os valores de duas funções  $f(n)$  e  $g(n)$  para  $n$  tendendo ao infinito. A notação  $O$  compara os valores em termos de  $\leq$  (permitindo igualdade) e a notação  $o$  compara os valores em termos de  $<$ , não permitindo igualdade.

Quando um algoritmo tem complexidade  $O(t(n))$  significa que o número de passos para inputs grandes é inferior ou igual a  $ct(n)$ . Quando um algoritmo tem complexidade  $o(t(n))$  significa que para qualquer  $c$ , existe um tamanho mínimo de input tal que a partir deste tamanho o número de passos é **inferior** a  $ct(n)$ . Por exemplo, o segundo algoritmo que vimos para decidir a linguagem  $A = \{0^k 1^k | k \geq 0\}$  tem complexidade  $O(n \log n)$  e  $o(n^2)$ .

**Esquema para mostrar que um determinado algoritmo  $M$  determinista tem complexidade  $O(t(n))$**

1. Descreva o algoritmo em *estágios*.
2. Analise o número de passos do algoritmo em cada estágio, em função do *comprimento do input*. Encontre um limitante superior deste número, utilizando a notação  $O$ .
3. Avalie o número de estágios utilizados na execução do algoritmo. Se existir um estágio  $X$  que corresponde a um loop do tipo : *Repita os estágios a seguir até que ...*, avalie o número de vezes que este loop vai ser executado até parar. Na verdade este estágio  $X$  corresponde a diversos estágios, um para cada vez que o loop é executado.
4. Para avaliar o número de vezes que um loop vai ser executado antes de parar, verifique o que acontece com a variável  $X$  envolvida no teste do loop (*Repita até que  $X$  verifique o teste ...*) a cada passada. Avalie quantas passadas vão ser necessárias no máximo para que a variável  $X$  passe no teste para a parada da execução do loop.
5. Se cada estágio  $E_i$  tem complexidade  $O(f(n))$ , se o número de estágios é  $O(g(n))$ , verifique se  $O(g(n)) \cdot O(f(n)) \leq O(t(n))$ . Neste caso, você mostrou que a complexidade do algoritmo é  $O(t(n))$ .

Para mostrar que um determinado problema  $A$  é NP-completo é necessário provar 2 fatos:

1. O problema  $A$  é NP. Para provar isto, você tem duas opções de prova:
  - se a pergunta do problema  $A$  claramente envolve a existência de algum objeto  $c$  verificando alguma propriedade (por exemplo, no caso do caixeiro viajante, a pergunta envolve a existência de um circuito passando por todos os vértices e com comprimento total inferior a um  $B > 0$  dado), você pode exibir um algoritmo  $V$  (chamado de “verificador”) que tem complexidade  $O(n^k)$  (polinomial) e que faz o seguinte:

para cada input  $w$  de  $A$  e para cada “sugestão”  $c$ ,  $V$  verifica se  $c$  satisfaz a propriedade necessária para que o input  $w$  seja uma instância para a qual o problema  $A$  responde “sim”. Por exemplo, no caso do caixeiro,  $c$  seria uma sequência qualquer de vértices do grafo input  $G$ . O algoritmo  $V$  simplesmente verifica se  $c$  é um circuito em  $G$  e se o seu comprimento é inferior ou igual a  $B$ .

- se a pergunta do problema  $A$  não envolve claramente a existência de um objeto satisfazendo uma certa propriedade, exiba um algoritmo Não-determinista Polinomial que decida o problema  $A$ .

2. Prove que o problema  $A$  é “*mais complicado*” do que um problema  $B$  que você sabe ser NP-completo. Atenção: é importante que a prova de que  $B$  é NP-completo não envolva o fato de que o problema  $A$  é NP-completo!! **Cuidado com os círculos viciosos !!**.

Para isto, você deve exibir uma redução polinomial  $f : B \rightarrow A$ , isto é:

- $f$  é um algoritmo de COMPLEXIDADE POLINOMIAL
- Se  $w \in B$  ( $w$  tem resposta “sim” para o problema  $B$ ) então  $f(w) \in A$  ( $f(w)$  tem resposta “sim” para o problema  $A$ ).
- **Reciprocamente:** Se  $f(w) \in A$  ( $f(w)$  tem resposta “sim” para o problema  $A$ ) então  $w \in B$  ( $w$  tem resposta “sim” para o problema  $B$ )

Denotamos por NPC = classe dos problemas NP-completos

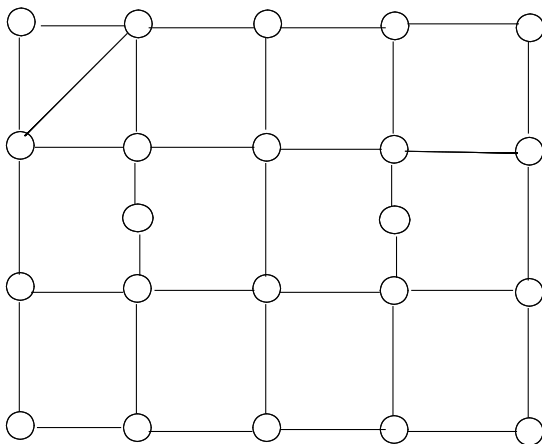
## Lista de Exercícios

1. Exercício 7.1 do Livro M. Sipser, página 271.
2. Considere as funções  $f(n) = 1000000n$ ;  $g(n) = 10 \cdot n^3$ ;  $h(n) = 2^n$ . Quais são as constantes  $c$  utilizadas para provar  $f(n)$  é  $O(g(n))$ , que  $g(n)$  é  $O(h(n))$  e que  $f(n)$  é  $O(h(n))$ ? Qual o menor valor de  $n$  para o qual  $f(n) \leq g(n) \leq h(n)$ ?
3. Ordene as seguintes funções em termos de sua taxa de crescimento, isto é, em termos da notação  $O$ :

$$n^2, 2^n, n \log n, n!, 4^n, n^n, n^{\log n}, 2^{2^n}, 2^{2n}, 2^{2^{n+1}}$$

4. Mostre que para qualquer  $a \in \mathbb{R}^+$ ,  $a^n$  é  $o(n!)$ .
5. Exercício 7.2 do Livro M. Sipser, página 271.
6. Suponha que você tem 5 algoritmos que resolvem um problema, com as seguintes funções de complexidade temporal:  $10^6 n$ ,  $10^4 n^2$ ,  $n^4$ ,  $2^n$ ,  $n!$ , em termos do tamanho  $n$  do input.
  - Seu computador executa  $10^8$  passos de cálculo por segundo. Qual o máximo tamanho de input  $n$  que você pode resolver com cada um dos algoritmos em um segundo (isto é, o algoritmo produz a resposta em até 1 segundo)?
  - Responda a mesma pergunta para 1 dia: qual o tamanho máximo do input que o algoritmo consegue decidir em até um dia de execução?
  - Como os números em (a) e (b) mudariam se você comprasse um computador 10 vezes mais rápido?
7. Dado um grafo  $G$  não dirigido, dizemos que um subconjunto  $X$  de vértices de  $G$  é
  - (a) *independente* se para quaisquer  $v, u \in X$ , não existe uma aresta ligando  $v$  e  $u$ .
  - (b) *clique* se para quaisquer  $v, u \in X$ , existe uma aresta ligando  $v$  e  $u$ .
  - (c) *vertex-cover* se para toda aresta  $(u, v)$  de  $G$  ao menos uma das extremidades está em  $X$ .

Considere o grafo não-dirigido  $G$  da figura abaixo. Pergunta-se: o grafo  $G$  tem um circuito hamiltoniano? Qual a clique de maior tamanho? Qual o conjunto independente de maior tamanho? Qual o vertex-cover de menor tamanho?



8. Até o momento, definimos os conceitos de *decidibilidade* e *complexidade* para problemas de *decisão*, isto é, problemas cujo output é “sim” ou “não”. Um problema de *otimização* é um problema que pede como resposta um objeto minimal ou maximal, isto é, um objeto que minimize ou maximize uma função. Por exemplo, a versão *otimização* do problema do caixeiro viajante é a seguinte:

*Instância (input)*: um grafo  $G$ , onde a cada aresta é associado um número inteiro positivo  $d$  (distância entre os vértices da aresta).

*Pergunta*: qual o comprimento minimal de um ciclo no grafo, passando por todas os vértices ?

Pede-se: Resolva o problema de otimização do caixeiro viajante para o grafo  $G = (V, E)$ , onde  $V = \{A, B, C, D, E\}$  e  $E = \{(A,B,8), (A,C,4), (A,D,5), (A,E,9), (B,C,1), (B,D,7), (B,E,3), (C,D,6), (C,E,2), (D,E,5)\}$ . Resolva o problema de decisão do Caixeiro viajante para este mesmo grafo e para  $\alpha = 18$ .

9. Exercício 7.8 do Livro M. Sipser, página 272.
10. Exercício 7.9 do Livro M. Sipser, página 272.
11. Exercício 7.10 do Livro M. Sipser, página 272.
12. Exercício 7.12 do Livro M. Sipser, página 272.
13. Exercício 7.13 do Livro M. Sipser, página 272.
14. O objetivo deste exercício é mostrar que a classe  $EXP \neq$  da classe  $P$ , isto é, que existe um problema de decisão que não pode ser decidido por um algoritmo polinomial e que pode ser decidido por um algoritmo exponencial.

Considere o problema  $E = \{ \langle M, w \rangle \mid M \text{ executada em } w \text{ pára no estado } q_a \text{ até } 2^{|w|} \text{ passos de cálculo} \}$ .

- Mostre que  $E$  pertence à classe  $EXP = \{L \mid L \text{ é decidida por uma M.T. determinista simples de complexidade } O(2^n)\}$ . Lembre-se que mostramos em sala de aula que este problema não está na classe  $\mathcal{P}$ .
  - Mostre que  $E$  não pertence à classe  $P$ . Sugestão: Suponha que  $E$  pertencesse à classe  $P$ . Considere o problema  $E' = \{ \langle M \rangle \mid M \text{ executada no string } \langle M \rangle \text{ pára no estado } q_a \text{ até } 2^{|\langle M \rangle|} \text{ passos de cálculo} \}$ . Este problema seria polinomial. Seja  $M'$  a máquina de Turing determinista polinomial que decide  $E'$ . O que acontece quando se executa  $E'$  em  $\langle M' \rangle$  ? Se você tiver dúvida, consulte o livro Elements of the Theory of Computation, 2a edição, H.R. Lewis, C.H. Papadimitriou, Teorema 6.1.2, página 277.
15. Exiba uma redução polinomial de  $SAT \rightarrow$  CLIQUE (Sugestão: Generalize a idéia da demonstração feita em sala de aula para a redução  $3SAT \rightarrow$  CLIQUE).
16. Mostre que CLIQUE se reduz a INDSET e vice-versa (exiba redução polinomial  $f$ : CLIQUE  $\rightarrow$  INDSET e redução polinomial  $g$ : INDSET  $\rightarrow$  CLIQUE).

Problema CLIQUE:

Input: Grafo não-dirigido  $G = (V,E)$ , número natural  $K$ ,  $0 < K \leq |V|$

Pergunta: Existe um subconjunto de vértices  $C \subseteq V$  tal que  $|C| \geq K$  e para quaisquer  $a, b \in C$  tem-se  $(a, b) \in E$  ?

Problema INDSET:

Input: Grafo não-dirigido  $G = (V,E)$ , número natural  $K$ ,  $0 < K \leq |V|$

Pergunta: Existe um subconjunto de vértices  $I \subseteq V$  tal que  $|I| \geq K$  e para quaisquer  $a, b \in I$  tem-se  $(a, b) \notin E$  ?

17. Descubra o erro na seguinte “demonstração” de que  $P \neq NP$  :

Considere o seguinte algoritmo para o problema SAT : “No input  $\phi$ , tente todas as possíveis avaliações das variáveis proposicionais de  $\phi$ . Responda ‘sim’ se uma delas satisfaz  $\phi$ .” Este algoritmo é evidentemente exponencial em tempo. Assim SAT tem complexidade exponencial. Portanto, SAT não está em P. É fácil mostrar que SAT está em NP. Logo, acabamos de exibir um problema que está em NP e não está em P. Portanto,  $P \neq NP$ .

18. Dois grafos  $G$  e  $H$  são ditos *isomorfos* se os vértices de  $G$  podem ser renomeados de modo que o grafo resultante seja idêntico a  $H$ . Por exemplo, os dois grafos  $G = (V,E)$  e  $H = (V',E')$  onde  $V = \{1,2,3,4\}$ ,  $E = \{(1,3), (3,4), (2,3)\}$ ,  $V' = \{a,b,c,d\}$ ,  $E' = \{(a,c), (c,d), (b,c)\}$  são isomorfos, pois se fizermos a seguinte renomeação nos vértices de  $G$  :  $1 \rightarrow a$ ,  $2 \rightarrow b$ ,  $3 \rightarrow c$  e  $4 \rightarrow d$ , obtemos o grafo  $H$ . Considere o seguinte problema:

$ISO = \{ \langle G, H \rangle \mid G \text{ e } H \text{ são isomorfos} \}$ .

Mostre que  $ISO$  é NP.

19. Exercício 7.16, página 272 do livro texto M. Sipser.
20. Exercício 7.21, página 273 do livro texto M. Sipser. Neste exercício, fala-se da noção de grafo *completo*. Um grafo é dito *completo* se quaisquer dois de seus vértices  $v_1, v_2$  estão ligados diretamente por uma aresta no grafo, isto é,  $(v_1, v_2) \in E_G$ , onde  $E_G =$  conjunto das arestas de  $G$ . Repare que um grafo completo é uma clique de tamanho  $n$ , onde  $n =$  número de vértices do grafo.
21. Exercício 7.22, página 273 do livro texto M. Sipser.
22. O Teorema de Ladner (1975) tem o seguinte enunciado : Seja  $B$  uma linguagem recursiva (decidível), não polinomial. Então existe uma linguagem polinomial  $D$  tal que : (1)  $D \cup B$  não é polinomial, (2)  $D \cup B$  é redutível a  $B$  e (3)  $B$  não é redutível a  $D \cup B$ .

Utilizando o teorema de Ladner, e supondo que  $P \neq NP$ , mostre que existe uma linguagem NP que é “intermediária”, isto é, que não é nem P, nem NP-completa. *Sugestão : aplique o teorema de Ladner para  $B =$  problema SAT. Mostre que a linguagem  $D \cup B$  produzida pelo teorema de Ladner é “intermediária”.*

23. Aplicando o teorema de Ladner diversas vezes e supondo  $P \neq NP$ , mostre que existe uma hierarquia de problemas intermediários entre NPC e P, isto é: existem linguagens (problemas de decisão)  $L_0, L_1, \dots, L_n$ , tais que:

- $L_0$  é NP-completo
  - Para todo  $i > 0$ ,  $L_i$  se reduz a  $L_{i-1}$  mas  $L_{i-1}$  não se reduz a  $L_i$
  - Para todo  $i \geq 0$ ,  $L_i$  não é polinomial.
24. Um problema é dito co-NP se o seu complementar é NP. Mostre que se  $NP \neq co-NP$  então  $P \neq NP$ . (Até hoje, ninguém mostrou a conjectura  $NP \neq co-NP$ ). Repare que  $P = co-P$ , mas o problema análogo para NP ( $NP = co-NP$ ) ainda está em aberto. Reflita por que não é óbvio que o complementar de um problema NP é também NP (veja que é óbvio que o complementar de um problema P é P).
25. Mostre que se existe um problema NP-completo tal que seu complementar é NP então  $NP = co-NP$ .
26. Em 2004, Manindra Agrawal, Neeraj Kayal, Nitin Saxena provaram que o problema PRIMOS está na classe  $\mathcal{P}$  (é decidido por um algoritmo polinomial) (Ver referência: Manindra Agrawal, Neeraj Kayal, Nitin Saxena: PRIMES is in P, Annals of Mathematics 160 (2004), no. 2, pp. 781-793.). Antes disto, ninguém tinha conseguido exibir uma prova de que PRIMOS fosse polinomial e nem que fosse NP-completo. Acreditava-se que PRIMOS, bem como o seu complementar COMPOSTOS fossem problemas intermediários entre  $\mathcal{P}$  e NPC (= conjunto dos problemas NP-completos). Esta *crença* de que COMPOSTOS fosse um problema *intermediário* entre  $\mathcal{P}$  e NPC baseava-se no raciocínio abaixo. Pede-se que você esclareça este raciocínio, respondendo às perguntas propostas:
- Em 1975, Pratt mostrou que o problema PRIMOS é NP. Baseado neste resultado, mostre que se o problema COMPOSTOS fosse NP-completo então  $NP = co-NP$ .
  - Até aquele momento, ninguém havia exibido um algoritmo polinomial para COMPOSTOS e ninguém havia mostrado a conjectura  $NP \neq co-NP$  (ou a conjectura  $NP = co-NP$ ). Conclua que COMPOSTOS era um candidato para ser um problema “intermediário”.
  - Podemos utilizar o mesmo raciocínio acima para concluir que o problema *PRIMOS* era um candidato possível para ser um problema “intermediário” ?

**Observação:** Hoje, 30 anos depois, com o resultado de M. Agrawal et al., sabemos que todo este raciocínio é inútil, já que tanto *PRIMOS* quanto *COMPOSTOS* são polinomiais, e portanto não estão em classes intermediárias entre  $\mathcal{P}$  e NPC.