

Análise e Síntese de Algoritmos

Complexidade Computacional

CLRS Cap. 35

Instituto Superior Técnico

2022/2023



Resumo

Motivação

Problemas Resolúveis em Tempo Polinomial

Problemas Verificáveis em Tempo Polinomial

Redutibilidade e Completude-NP

Contexto

Revisão [CLRS, Cap.1-13]

Fundamentos; notação; exemplos

Algoritmos em Grafos [CLRS, Cap.21-26]

Algoritmos elementares

Caminhos mais curtos

Fluxos máximos

Árvores abrangentes

Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]

Algoritmos greedy

Programação dinâmica

Programação Linear [CLRS, Cap.29]

Algoritmos e modelação de problemas com restrições lineares

Tópicos Adicionais [CLRS, Cap.32-35]

Emparelhamento de Cadeias de Caracteres

Complexidade Computacional

Análise e Síntese de Algoritmos - 2022/2023

1/70

Motivação

A empresa onde trabalham vai entrar no competitivo mercado dos gambozinómetros

O vosso chefe, chama-vos ao gabinete e pede-vos para elaborarem um algoritmo que permita determinar as especificações ótimas para o primeiro modelo do gambozinómetro

Após consultarem o departamento responsável pelo desenho dos gambozinómetros, para se inteirarem dos contornos do problema, colocam mãos à obra ...

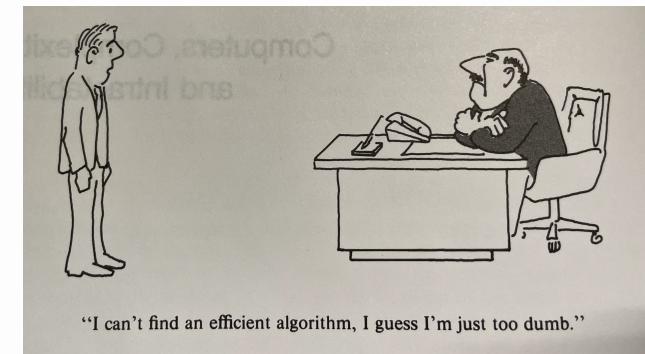
A empresa onde trabalham vai entrar no competitivo mercado dos gambozinómetros

O vosso chefe, chama-vos ao gabinete e pede-vos para elaborarem um algoritmo que permita determinar as especificações ótimas para o primeiro modelo do gambozinómetro

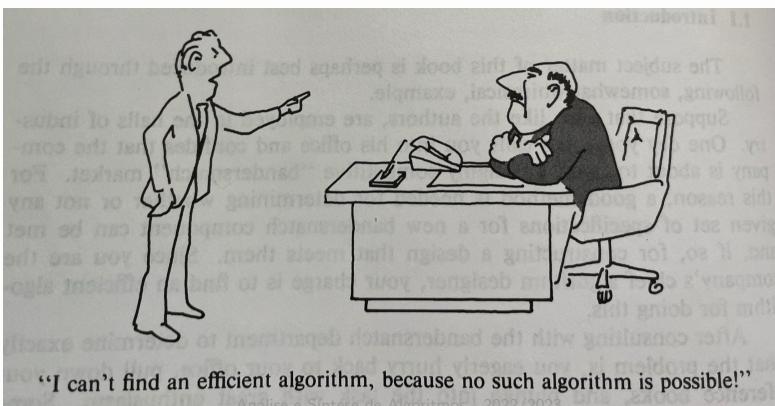
Após consultarem o departamento responsável pelo desenho dos gambozinómetros, para se inteirarem dos contornos do problema, colocam mãos à obra ...

Várias semanas depois, já com o vosso gabinete atolado de pilhas de papel, e muitas noites mal dormidas, o vosso entusiasmo diminuiu ...

Até agora ainda não descobriram um algoritmo melhor do que percorrer e avaliar todos os desenhos possíveis de gambozinómetros, para determinar qual o melhor – isso provavelmente envolverá vários anos de computação. Não querem voltar ao gabinete do vosso chefe e dar-lhe a má notícia ...



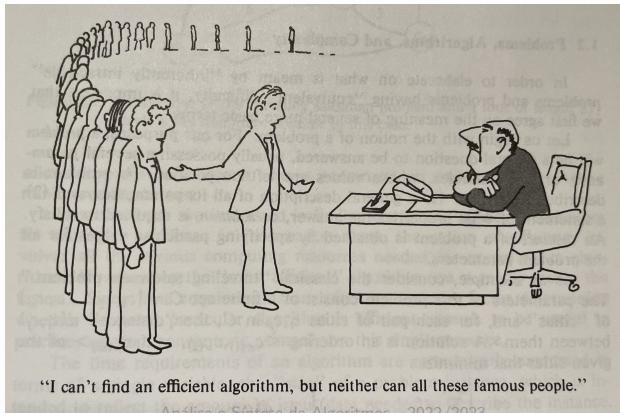
Para evitar e mancharem a vossa reputação como engenheiros informáticos do Técnico, seria preferível se conseguissem provar que o problema dos gambozinómetros é *intrinsecamente intratável*, ou seja, que não existe um algoritmo eficiente para sua solução



Infelizmente, provar que um problema é intrinsecamente intratável pode ser tão difícil quanto encontrar um algoritmo eficiente para o resolver

A teoria da completude-NP (*NP-completeness*) fornece-nos um conjunto de técnicas para provar que um dado problema computacional é tão difícil quanto um conjunto de outros problemas amplamente reconhecidos como sendo difíceis de resolver

Utilizando estas técnicas poderão conseguir provar ao vosso chefe que o problema em questão é NP-completo, e que portanto é equivalente a um conjunto de outros problemas já estudados por cientistas ilustres, e para a solução dos quais não se conseguiu descobrir um algoritmo eficiente



Análise e Síntese de Algoritmos - 2022/2023

Algoritmos Polinomiais

Complexidade em $O(n^k)$

Quase todos os algoritmos estudados em ASA (até agora)

Exemplo exceção: Simplex

Existem algoritmos polinomiais para qualquer problema ? Não !

Existem problemas não resolúveis

Existem problemas não resolúveis em tempo $O(n^k)$ para qualquer k

Problemas intratáveis: requerem tempo superpolinomial

Problemas NP-completos (desde 1971)

Não provado que são tratáveis ou que são intratáveis

Conjectura: problemas NP-completos são intratáveis

Perspectiva

Problemas de decisão

Resposta sim(1)/não(0)

Classe de complexidade P

Polynomial time

Problemas **resolúveis** em tempo polinomial

Codificação de problemas

Linguagens formais

Algoritmos de verificação

Classe de complexidade NP Nondeterministic Polynomial time

Problemas **verificáveis** em tempo polinomial

Redutibilidade entre problemas de decisão

Problemas NP-completos

Problemas Resolúveis em Tempo Polinomial vs. NP-completos

Caminhos mais curtos vs. caminhos mais longos

Mesmo com arcos com peso negativo é possível determinar caminhos mais curtos em tempo $O(VE)$

Determinar o caminho mais longo entre dois vértices é NP-completo

2-CNF SAT vs. 3-CNF SAT

Determinar valores de x_1, x_2, x_3, x_4 que satisfazem $(\bar{x}_1 + x_2)(x_2 + x_3)(x_1 + \bar{x}_4) = 1$ pode ser feito em tempo polinomial

Determinar valores de x_1, x_2, x_3, x_4 que satisfazem $(\bar{x}_1 + x_2 + \bar{x}_4)(x_2 + \bar{x}_3 + x_4) = 1$ é um problema NP-completo

Classes de Problemas P, NP e NPC

Classe **P**: problemas resolúveis em tempo polinomial

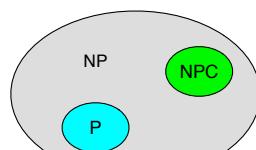
Classe **NP**: problemas verificáveis em tempo polinomial

Dado um certificado de uma solução, é possível verificar que o certificado é correcto, em tempo polinomial na dimensão do problema

Classe **NPC**: problemas NP-completos

Problemas tão difíceis como qualquer problema em NP

Se *algum* problema NP-completo puder ser resolvido em tempo polinomial, então *todos* os problemas NP-completos podem ser resolvidos em tempo polinomial



Análise e Síntese de Algoritmos - 2022/2023

11/70

Problemas Resolúveis em Tempo Polinomial

Porquê admitir problemas resolúveis em tempo polinomial como tratáveis?

Algoritmos polinomiais são normalmente limitados em $O(n^k)$, com k "baixo"

Para modelos de computação usuais, algoritmo polinomial num modelo é polinomial noutros modelos

Serial random-access machine (habitual), computador paralelo

Propriedades de fecho dos algoritmos polinomiais (soma, multiplicação e composição)

Se o resultado de um algoritmo polinomial for utilizado como input de outro algoritmo polinomial, o algoritmo resultante também é polinomial

Análise e Síntese de Algoritmos - 2022/2023

13/70

Exemplos de Problemas NPC

Satisfação de fórmulas proposicionais - SAT

Coloração de grafos

Instalação de pacotes de software

Problemas em lógica

Problemas em grafos / redes

Problemas de autómatos

Problemas em geração de código / compiladores

Problemas em redes de computadores

Problemas em bases de dados

Problemas em investigação operacional

.... (largas centenas)

Análise e Síntese de Algoritmos - 2022/2023

12/70

Problemas Resolúveis em Tempo Polinomial

Problema Abstracto Q

Relação binária entre conjunto I de instâncias e conjunto S de soluções

Exemplo: Problema SHORTEST-PATH

Instância: $i = \langle G, u, v \rangle$, grafo G , vértice origem u e destino v

Solução: sequência de vértices do caminho mais curto

O problema é a relação que associa a cada instância uma ou mais soluções

Análise e Síntese de Algoritmos - 2022/2023

14/70

Problemas de Decisão

Problemas cuja resposta/solução é sim/não (1/0), $Q(i) \in \{0, 1\}$

Problemas de optimização:

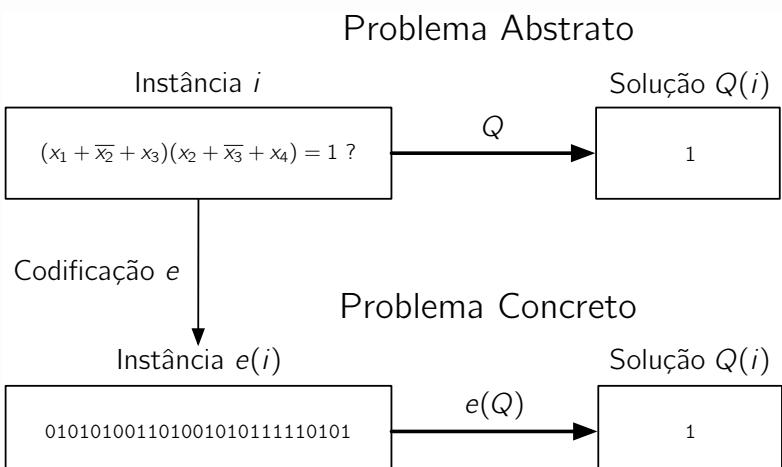
Reformulados como problemas de decisão

Se problema de optimização é tratável, então reformulação como problema de decisão também é tratável

Exemplo: Problema PATH

Instância: $i = \langle G, u, v, k \rangle$, número máximo de arcos k

Solução: 1/0, se um caminho mais curto entre u e v tem ou não no máximo k arcos



Codificação de Problemas

Codificação:

Dado conjunto abstracto de objectos S , uma codificação e é uma função que mapeia os elementos de S em strings binárias

Problema concreto:

Problema com conjunto de instâncias I representadas como strings binárias

Uma codificação e permite mapear um problema abstracto, Q , num problema concreto, $e(Q)$

Exemplo

Codificação dos números naturais $\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$ nas strings binárias $\{0, 1, 10, 11, 100, \dots\}$

Utilizando esta codificação, $e(17) = 10001$

Codificação de Problemas

Problema resolúvel em tempo polinomial

Solução para instância $i \in I$, $n = |i|$, pode ser encontrada em tempo $O(n^k)$, com k constante

Classe de complexidade **P**

Conjunto de problemas de decisão concretos resolúveis em tempo polinomial

Codificação de Problemas

A codificação utilizada tem impacto na eficiência com que é possível resolver um problema

Para codificações “razoáveis” de problemas abstractos, a codificação utilizada não afecta se um dado problema pode ser resolúvel em tempo polinomial

Função $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ é computável em tempo polinomial se existe um algoritmo de tempo polinomial A que, dado $x \in \{0, 1\}^*$, calcula $f(x)$

Codificações e_1 e e_2 são relacionadas polynomialmente se existem duas funções polynomialmente computáveis, f_{12} e f_{21} , tal que para $i \in I$, $f_{12}(e_1(i)) = e_2(i)$ e $f_{21}(e_2(i)) = e_1(i)$

Utilização de Linguagens Formais

Utilização de Linguagens Formais

Alfabeto Σ : conjunto finito de símbolos

Linguagem L definida em Σ : conjunto de strings de símbolos de Σ

Linguagem Σ^* : todas as strings de Σ

String vazia: ϵ

Linguagem vazia: \emptyset

Qualquer linguagem L em Σ é um subconjunto de Σ^*

Operações sobre linguagens: união, intersecção, complemento, concatenação, fecho

Exemplo

Se $\Sigma = \{0, 1\}$, então $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$ é o conjunto de todas as strings binárias

Codificação de Problemas

Seja Q um problema de decisão abstracto com conjunto de instâncias I , e sejam e_1 e e_2 codificações relacionadas polynomialmente.

Então, $e_1(Q) \in P$ se e só se $e_2(Q) \in P$

Admitir que $e_1(Q)$ é resolúvel em tempo $O(n^k)$ (k constante) $e_1(i)$ calculável a partir de $e_2(i)$ em tempo $O(n^c)$, com $n = |e_2(i)|$

Para resolver o problema $e_2(Q)$ sobre a instância $e_2(i)$
Calcular $e_1(i)$ a partir de $e_2(i)$

Resolver o problema $e_1(Q)$ sobre a instância $e_1(i)$
Complexidade polinomial $O(n^{ck})$

Conversão de codificações: $O(n^c)$ (c constante)
 $|e_1(i)| = O(n^c)$, a saída é limitada pelo tempo de execução
Tempo para resolver $e_1(i)$: $O(|e_1(i)|^k) = O(n^{ck})$

Polinomial por c e k serem constantes

Utilização de Linguagens Formais

Utilização de Linguagens Formais

Para qualquer problema de decisão Q , o conjunto de instâncias é Σ^* , com $\Sigma = \{0, 1\}$

Q é completamente caracterizado pelas instâncias que produzem solução 1 (sim)

Q pode ser interpretado como linguagem L definida em $\Sigma = \{0, 1\}$

$$L = \{x \in \Sigma^* : Q(x) = 1\}$$

Exemplo

$$L = \{x \in \Sigma^* : \text{Primo}(x) = 1\}$$

$$L = \{10, 11, 101, 111, 1011, 1101, 10001, \dots\}$$

O problema Primo é completamente caracterizado pelo conjunto (a linguagem) de representação binária dos números primos

Aceitação/Rejeição

Algoritmo A aceita $x \in \{0,1\}^*$ se $A(x) = 1$

Algoritmo A rejeita $x \in \{0,1\}^*$ se $A(x) = 0$

Linguagem aceite por A : $L = \{x \in \{0,1\}^* : A(x) = 1\}$

Decisão

Mesmo que L seja aceite por A , o algoritmo não tem que necessariamente rejeitar $x \notin L$

Pode, por exemplo, entrar em loop infinito

L é decidida por A se qualquer string $x \in \{0,1\}^*$ é aceite ou rejeitada

L é decidida em tempo polinomial se A decide corretamente se $x \in L$ em tempo $O(n^k)$, com $n = |x|$ e k constante

Aceitação vs. Decisão

Para aceitar uma linguagem L , um algoritmo A apenas tem que produzir uma resposta para cada string $x \in L$

Para decidir uma linguagem L , um algoritmo A tem que corretamente aceitar ou rejeitar qualquer string $\{0,1\}^*$

Classe de Complexidade P

Conjunto das linguagens para as quais existe um algoritmo que as decide em tempo polinomial

Classe das linguagens decididas por algoritmos em tempo polinomial é um subconjunto da classe das linguagens aceites por algoritmos em tempo polinomial

Se decide, então aceita (e rejeita)

Se aceita, pode ou não decidir (ex: loop infinito)

Definições Alternativas para a Classe de Complexidade P

$P = \{L \in \{0,1\}^* :$
existe um algoritmo A que decide L em tempo polinomial}
 $P = \{L \in \{0,1\}^* :$
 L é aceite por um algoritmo de tempo polinomial}

Prova

Conjunto das linguagens decididas em tempo polinomial é subconjunto das linguagens aceites em tempo polinomial

Basta provar que se L é aceite por algoritmo polinomial, implica que L é decidida por algoritmo polinomial

Definições Alternativas para a Classe de Complexidade P

$P = \{L \in \{0,1\}^* :$
existe um algoritmo A que decide L em tempo polinomial}
 $P = \{L \in \{0,1\}^* :$
 L é aceite por um algoritmo de tempo polinomial}

Prova (cont.)

Basta provar que se L é aceite por algoritmo polinomial, implica que L é decidida por algoritmo polinomial

A aceita L em $O(n^k)$

Utilizar A' que executa A e observa resultado após $T = cn^k$

Se A aceita, A' aceita; se A não aceita (ainda), A' rejeita

Logo A' decide L em tempo $O(n^k)$

Verificação

Objectivo é **verificar** se uma instância pertence a uma dada linguagem utilizando um certificado (i.e. uma possível solução)

Exemplo

Problema 3-CNF SAT:

$$(x_1 + \bar{x}_2 + x_3)(\bar{x}_1 + x_2 + x_4)(\bar{x}_1 + x_3 + x_4) = 1 ?$$

Certificado:

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$$

Verificação:

$$(1 + 0 + 0)(0 + 1 + 1)(0 + 0 + 1) = 1 !$$

Classe de Complexidade NP

Classe de Complexidade NP

Classe das linguagens que podem ser **verificadas** em tempo polinomial

Uma linguagem L pertence a NP se existe um algoritmo A e uma constante c , tal que:

$$L = \{x \in \{0,1\}^*: \text{ existe um certificado } y \in \{0,1\}^*, \text{ com } |y| = O(|x|^c), \text{ tal que } A(x,y) = 1\}$$

A **verifica** L em tempo polinomial

Classe co-NP

Linguagens L tal que $\bar{L} \in \text{NP}$

Exemplo: CNFUNSAT

Algoritmo de Verificação A

Argumentos:

string x : entrada

string y : certificado

Diz-se que A **verifica** uma string the entrada x , se existe um certificado y , tal que $A(x,y) = 1$

Certificado permite provar que $x \in L$

A linguagem **verificada** por A é:

$$L = \{x \in \{0,1\}^*: \text{ existe } y \in \{0,1\}^* \text{ tal que } A(x,y) = 1\}$$

Exemplo (Problema 3-CNF SAT)

entrada x : formula CNF

certificado y : atribuições às variáveis

$A(x,y)$: substitui atribuição de cada variável na formula e verifica se a formula é verdadeira

Problemas Verificáveis em Tempo Polinomial

Relações entre Classes de Complexidade

$$P \subseteq NP$$

Poder decidir implica poder verificar

$$P \subseteq NP \cap co-NP$$

P fechado quanto a complemento

Questões em aberto:

$$P = NP ??$$

$$P = NP \cap co-NP ??$$

$$\text{Existe } L \text{ em } (NP \cap co-NP) - P ??$$

Relações entre Classes de Complexidade

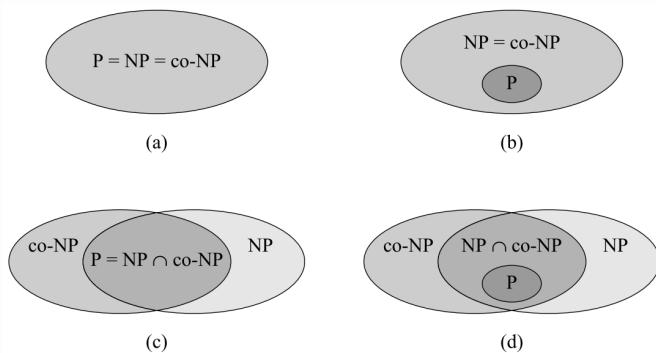


Figure 34.3 Four possibilities for relationships among complexity classes. In each diagram, one region enclosing another indicates a proper-subset relation. **(a)** $P = NP = \text{co-NP}$. Most researchers regard this possibility as the most unlikely. **(b)** If NP is closed under complement, then $NP = \text{co-NP}$, but it need not be the case that $P = NP$. **(c)** $P = NP \cap \text{co-NP}$, but NP is not closed under complement. **(d)** $NP \neq \text{co-NP}$ and $P \neq NP \cap \text{co-NP}$. Most researchers regard this possibility as the most likely.

Redutibilidade e Completude-NP

Redução

Um problema Q pode ser [reduzido](#) a outro problema Q' , se qualquer instância de Q pode ser facilmente reescrita como uma instância de Q'

A solução da instância de Q' fornece uma solução para a instância correspondente de Q

Redutibilidade e Completude-NP

Redutibilidade em Tempo Polinomial

Z é redutível em tempo polinomial a X , $Z \leq_P X$, se existir uma função, $f : Z \rightarrow X$, calculável em tempo polinomial, tal que para qualquer $z \in Z$:

$$Z(z) = 1 \text{ se e só se } X(f(z)) = 1$$

f é designada por [função de redução](#), e o algoritmo F de tempo polinomial que calcula f é designado por [algoritmo de redução](#)

Se Z , X são problemas de decisão, com $Z \leq_P X$, então $X \in P$ implica $Z \in P$

Completude-NP

Um problema de decisão X diz-se **NP-completo** se:

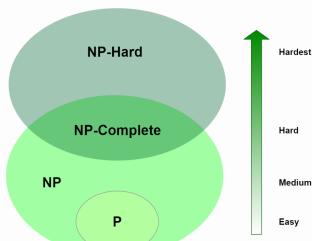
$$X \in NP$$

$$Z \leq_P X \text{ para qualquer } Z \in NP$$

Um problema de decisão X diz-se **NP-difícil** (*NP-hard*) se:

$$Z \leq_P X \text{ para qualquer } Z \in NP$$

NPC: classe de complexidade dos problemas de decisão NP-completos



Completude-NP

Se existir problema NP-completo X , resolúvel em tempo polinomial, então $P = NP$

Todos os problemas em NP redutíveis a X (em tempo polinomial)

Logo, resolúveis em tempo polinomial

Se existir problema X em NP não resolúvel em tempo polinomial, então todos os problemas NPC-completos não são resolúveis em tempo polinomial

Se existisse Y em NPC resolúvel em tempo polinomial, dado que $X \leq_P Y$, então X seria resolúvel em tempo polinomial

Completude-NP

Provar Problemas NP-Completos

Seja X um problema de decisão:

$$Y \leq_P X \wedge Y \in NPC \wedge X \in NP \Rightarrow X \in NPC$$

Prova

$$Y \in NPC \Rightarrow \forall Z \in NP, Z \leq_P Y$$

$$\leq_P \text{ é transitiva} \wedge Y \leq_P X \Rightarrow \forall Z \in NP, Z \leq_P X$$

$$X \in NP \wedge \forall Z \in NP, Z \leq_P X \Rightarrow X \in NPC !$$

Completude-NP

Provar Problemas NP-Completos

Abordagem para provar $X \in NPC$

Provar que $X \in NP$

Escolher $Y \in NPC$

Redução $Y \leq_P X$

Descrever um algoritmo que calcula função f , a qual converte qualquer instância de Y numa instância de X
Provar que $x \in Y$ se e só se $f(x) \in X$, para qualquer instância x

Provar que algoritmo que calcula f tem tempo de execução polinomial

Como definir $Y \in NPC$ inicial ?

Problema SAT

Problema de decisão

Fórmula proposicional ϕ :

variáveis proposicionais, x_1, \dots, x_n

conectivas proposicionais: $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$

parêntesis

Atribuição de verdade: atribuir valores proposicionais (0 ou 1) às variáveis

Atribuição de satisfação: valor da fórmula é 1

Se atribuição de satisfação existe, ϕ é satisfeita

Problema: determinar se uma instância ϕ é satisfeita

$$\text{SAT} = \{\langle \phi \rangle : \phi \text{ é uma fórmula proposicional satisfeita}\}$$

Teorema de Cook

O problema SAT é NP-completo

Prova

$\text{SAT} \in \text{NP}$

O certificado consiste numa atribuição de valores às variáveis

Substituir valores e analisar fórmula resultante

Tempo de execução é polinomial no tamanho da fórmula

SAT é NP-difícil [Cook, 1971]

(ver CLRS, Lema 34.6)

$\therefore \text{SAT}$ é NP-completo

Problema CNFSAT

Problema de decisão

Fórmula CNF (conjunctive normal form) ϕ :

variáveis proposicionais, x_1, \dots, x_n

conectivas proposicionais: \wedge, \vee, \neg

fórmula é conjunção (\wedge) de disjunções (\vee) de literais

literal: variável (x_i) ou o seu complemento ($\neg x_i$)

Atribuição de verdade: atribuir valores proposicionais (0 ou 1) às variáveis

Atribuição de satisfação: valor da fórmula é 1

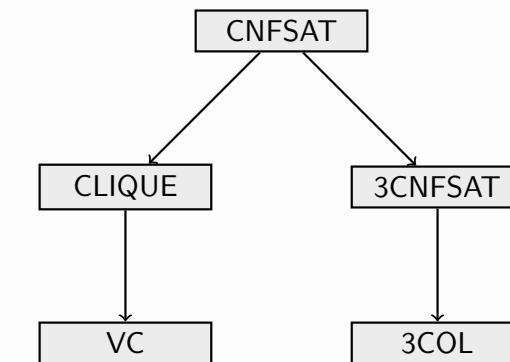
Se atribuição de satisfação existe, ϕ é satisfeita

Problema: determinar se uma instância ϕ é satisfeita

$$\text{CNFSAT} = \{\langle \phi \rangle : \phi \text{ é uma fórmula CNF satisfeita}\}$$

CNFSAT é NP-Completo ($\text{SAT} \leq_P \text{CNFSAT}$)

Provar Problemas NP-Completos



Problema 3CNFSAT

3CNFSAT é uma restrição do problema CNFSAT em que cada cláusula contém exactamente 3 literais

Teorema: o problema 3CNFSAT é NP-completo

Problema 3CNFSAT

Prova: $3\text{CNFSAT} \in \text{NP}$

φ : instância de 3CNFSAT

$(x_1, v_1), \dots, (x_n, v_n)$: certificado / atribuição de valores

Calcular valor de cada disjunção e da conjunção

Complexidade: $O(|\varphi|)$

Polynomial no tamanho da instância

Problemas NP-Completos

Problema 3CNFSAT

Prova: 3CNFSAT é NP-Difícil: $\text{CNFSAT} \leq_P 3\text{CNFSAT}$

Redução (definição de f):

Dada instância $\varphi \in \text{CNFSAT}$, derivar instância $\varphi_3 \in 3\text{CNFSAT}$

Por cada cláusula unitária $w = (l_1)$:

Criar cláusulas:

$(l_1 \vee y_1 \vee y_2) \wedge (l_1 \vee \neg y_1 \vee y_2) \wedge (l_1 \vee y_1 \vee \neg y_2) \wedge (l_1 \vee \neg y_1 \vee \neg y_2)$,

com variáveis adicionais y_1 e y_2

Por cada cláusula binária $w = (l_1 \vee l_2)$:

Criar cláusulas: $(l_1 \vee l_2 \vee y_1) \wedge (l_1 \vee l_2 \vee \neg y_1)$, com variável adicional y_1

Por cada cláusula com mais que 3 literais

$w = (l_1 \vee l_2 \vee l_3 \vee \dots \vee l_k)$:

Criar cláusulas: $(l_1 \vee l_2 \vee y_1) \wedge (\neg y_1 \vee l_3 \vee y_2) \wedge \dots \wedge$

$(\neg y_{k-4} \vee l_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee l_{k-1} \vee l_k)$, com

variáveis adicionais y_1, y_2, \dots, y_{k-3}

Problemas NP-Completos

Problema 3CNFSAT

Prova: 3CNFSAT é NP-Difícil: $\text{CNFSAT} \leq_P 3\text{CNFSAT}$

Complexidade da função de redução f :

Número de variáveis adicionais é $O(|\varphi|)$

Número de cláusulas adicionais é $O(|\varphi|)$

Complexidade da redução é $O(|\varphi|)$

Problema 3CNFSAT

$\text{CNFSAT}(x) = 1$ se e só se $\text{3CNFSAT}(f(x)) = 1$

Cláusulas unitárias e binárias: prova é simples

Considerar cláusulas com mais que 3 literais

Se $\varphi = 1$:

Para cada cláusula w , existe $l_r = 1, 1 \leq r \leq k$

Atribuir valor 1 às variáveis y_s , $1 \leq s \leq \min(r-1, k-3)$

Atribuir valor 0 às variáveis y_t ,

$\min(r-1, k-3) + 1 \leq t \leq k-3$

Todas as cláusulas satisfeitas, pelo que $\varphi_3 = 1$

Se $\varphi_3 = 1$:

Um dos literais de $(l_1 \vee l_2 \vee l_3 \vee \dots \vee l_k)$ tem de ter valor 1

Caso contrário

$(y_1) \wedge (\neg y_1 \vee y_2) \wedge \dots \wedge (\neg y_{k-4} \vee y_{k-3}) \wedge (\neg y_{k-3})$ teria que ser satisfeita; impossível

Pelo que $\varphi = 1$

Análise e Síntese de Algoritmos - 2022/2023

47/70

Problema 2CNFSAT

3CNFSAT é NP-Completo, mas $\text{2CNFSAT} \in P$

2CNFSAT é uma restrição do problema CNFSAT em que cada cláusula contém exactamente 2 literais

Teorema: o problema $\text{2CNFSAT} \in P$

Prova:

Existe algoritmo para decidir 2CNFSAT com tempo de execução linear no tamanho de $|\varphi|$, $\varphi \in \text{2CNFSAT}$

Cada cláusula binária corresponde a dois arcos (implicações) num grafo

Identificar SCCs no grafo

Se existe SCC com x e $\neg x$ então instância não é satisfeita

Análise e Síntese de Algoritmos - 2022/2023

48/70

Problema HornSAT

HornSAT é uma restrição do problema CNFSAT em que cada cláusula contém não mais do que 1 literal não complementado

Teorema: o problema $\text{HornSAT} \in P$

Prova:

Existe algoritmo para decidir HornSAT com tempo de execução linear no tamanho de $|\varphi|$, $\varphi \in \text{HornSAT}$

Repetidamente satisfazer cláusulas com apenas 1 literal x_i não complementado (i.e. atribuir valor 1 (TRUE) a x_i)

Reducir cláusulas com literal complementado

Terminar quando identificada cláusula vazia (UNSAT) ou todas as cláusulas com literais complementados

Atribuir valor 0 (FALSE) às restantes variáveis; cláusulas satisfeitas !

Análise e Síntese de Algoritmos - 2022/2023

49/70

Problema HornSAT

HornSAT(φ)

```

while  $\exists$  cláusulas com literal positivo  $x_i$  do
     $x_i = 1$ 
    satisfazer cláusulas com  $x_i$ 
    reduzir cláusulas com  $\neg x_i$ 
    if  $\exists$  cláusula vazia then
        eliminar atribuições
        return FALSE
    end if
end while
atribuir  $x_j = 0$  às variáveis ainda não atribuídas
return TRUE

```

Análise e Síntese de Algoritmos - 2022/2023

50/70

Problema CLIQUE

Seja $G = (V, E)$ grafo não dirigido e k um inteiro positivo

O problema CLIQUE consiste em decidir a existência de um sub-grafo completo com k vértices em G

Teorema: o problema CLIQUE é NP-completo

Problema CLIQUE

Prova: CLIQUE ∈ NP

Seja $G = (V, E)$, grafo não dirigido

$V' \subseteq V$, com $|V'| = k$

Para cada $u \in V'$, verificar se existe arco $(u, v) \in E$ para qualquer $v \in V' - \{u\}$

Verificar se V' é sub-grafo completo com tempo de execução em $O(V+E)$

Problema CLIQUE

Prova: CLIQUE é NP-Difícil, CNFSAT \leq_P CLIQUE

Redução (definição de f):

Instância $\varphi = \omega_1 \wedge \dots \wedge \omega_m$

Gerar instância de CLIQUE $< G = (V, E), k >$

Vértices de G organizados por colunas

Cada coluna associada a uma cláusula

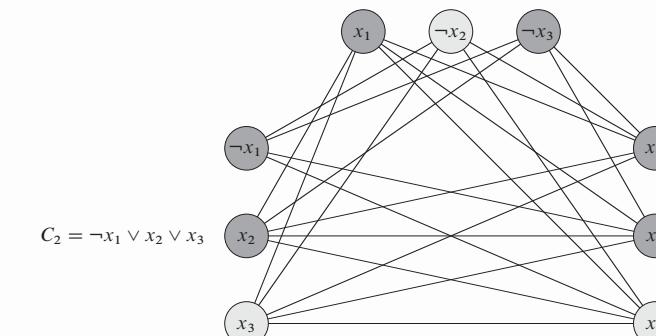
Número de vértices por coluna igual ao número de literais na cláusula correspondente

Arcos de G :

Vértices na mesma coluna não ligados por arcos

Vértices em colunas diferentes ligados por arcos, excepto se par de vértices corresponder a x e à respectiva negação $\neg x$

$$C_1 = x_1 \vee \neg x_2 \vee \neg x_3$$



$$C_2 = \neg x_1 \vee x_2 \vee x_3$$

$$C_3 = x_1 \vee x_2 \vee x_3$$

Figure 34.14 The graph G derived from the 3-CNF formula $\phi = C_1 \wedge C_2 \wedge C_3$, where $C_1 = (x_1 \vee \neg x_2 \vee \neg x_3)$, $C_2 = (\neg x_1 \vee x_2 \vee x_3)$, and $C_3 = (x_1 \vee x_2 \vee x_3)$, in reducing 3-CNF-SAT to CLIQUE. A satisfying assignment of the formula has $x_2 = 0$, $x_3 = 1$, and x_1 either 0 or 1. This assignment satisfies C_1 with $\neg x_2$, and it satisfies C_2 and C_3 with x_3 , corresponding to the clique with lightly shaded vertices.

Problema CLIQUE

$\text{CNFSAT}(x) = 1$ se e só se $\text{CLIQUE}(f(x)) = 1$

G tem sub-grafo completo de dimensão $k = m$ se e só se fórmula φ é satisfeita

Se φ é satisfeita:

Em cada coluna seleccionar vértice ao qual o literal respectivo tem atribuído o valor 1

Para este vértice u existe arco para vértice v em qualquer outra coluna, tal que v associado com literal com valor 1 atribuído

Conclusão: existe sub-grafo completo com dimensão m

Se G tem sub-grafo completo de dimensão m :

Um vértice tem de ser seleccionado em cada coluna

Atribuir valor 1 a cada vértice seleccionado (x e $\neg x$ não seleccionados simultaneamente)

Cada cláusula com 1 literal atribuído valor 1; φ é satisfeita

Análise e Síntese de Algoritmos - 2022/2023

55/70

Problema CLIQUE

Prova: CLIQUE é NP-Difícil, $\text{CNFSAT} \leq_P \text{CLIQUE}$

Complexidade da função de redução f :

Número de colunas de vértices igual a número de cláusulas

Em cada coluna um vértice associado com cada literal

Para n variáveis e m cláusulas:

$O(nm)$ vértices

Para cada vértice: $O(nm)$ arcos

Total de arcos: $O(n^2m^2)$

Redução realizada em tempo polinomial

Análise e Síntese de Algoritmos - 2022/2023

Problemas NP-Completos

Problemas NP-Completos

Problema Vertex Cover (VC)

Seja $G = (V, E)$ um grafo não dirigido e um k inteiro positivo

Cobertura de vértices (VC): conjunto de vértices V' tal que qualquer arco em G é incidente em pelo menos um dos vértices de V'

O problema VC consiste em decidir se G tem cobertura de vértices com k vértices

Teorema: o problema VC é NP-completo

Análise e Síntese de Algoritmos - 2022/2023

57/70

Problema Vertex Cover (VC)

Prova: $\text{VC} \in \text{NP}$

Dado $V' \subseteq V$, $|V'| = k$, analisar cada arco $(u, v) \in E$ e verificar se pelo menos um dos vértices u ou v está em V'

Se sim, V' é cobertura de vértices

Processo de verificação: $O(V + E)$

Análise e Síntese de Algoritmos - 2022/2023

58/70

Problema Vertex Cover (VC)

Prova: VC é NP-Difícil, CLIQUE \leq_P VC

Redução (definição de f):

$G = (V, E)$, não dirigido

$G^C = (V, E^C)$, $E^C = V \times V - E$, grafo complementar

G tem sub-grafo completo com k vértices se e só se G^C tem cobertura de vértices com $|V| - k$ vértices

Complexidade (de f):

Redução tem tempo de execução polinomial; basta identificar G^C

Problemas NP-Completos

Problema Vertex Cover (VC)

$\text{CLIQUE}(x) = 1$ se e só se $\text{VC}(f(x)) = 1$

G tem sub-grafo completo com k vértices, dado por V'

Como V' identifica um sub-grafo completo, em G^C não existem arcos entre os vértices de V'

Todos os arcos de G^C incidentes em pelo menos um vértice de $V - V'$

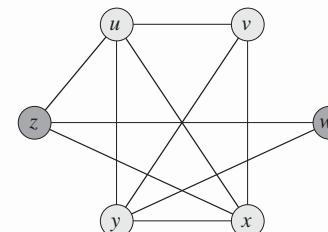
$V - V'$ é cobertura de vértices de G^C , com $|V| - k$ vértices

G^C tem cobertura $V - V'$, com $|V| - k$ vértices

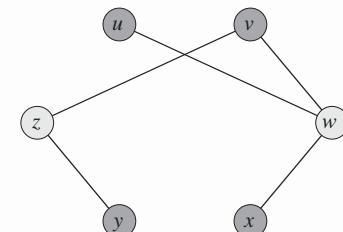
Se $u, v \in V'$, então $(u, v) \notin E^C$; caso contrário $V - V'$ não seria cobertura de vértices

Pelo que $(u, v) \in E$

V' é sub-grafo completo em G , com $|V'| = k$



(a)



(b)

Figure 34.15 Reducing CLIQUE to VERTEX-COVER. (a) An undirected graph $G = (V, E)$ with clique $V' = \{u, v, x, y\}$. (b) The graph \bar{G} produced by the reduction algorithm that has vertex cover $V - V' = \{w, z\}$.

Problemas NP-Completos

Problemas NP-Completos

Problema 3COL

Seja $G = (V, E)$ um grafo não dirigido

Coloração válida para G : atribuição de cores aos vértices de G tal que vértices adjacentes têm cores diferentes; G diz-se colorido

O problema 3COL consiste em decidir se G pode ser colorido com 3 cores

Teorema: o problema 3COL é NP-completo

Problema 3COL**Prova:** 3COL ∈ NPConsiderar a atribuição de uma de três cores a cada vértice de V Se existir $(u, v) \in E$ em que u e v têm a mesma cor, coloração não é válida

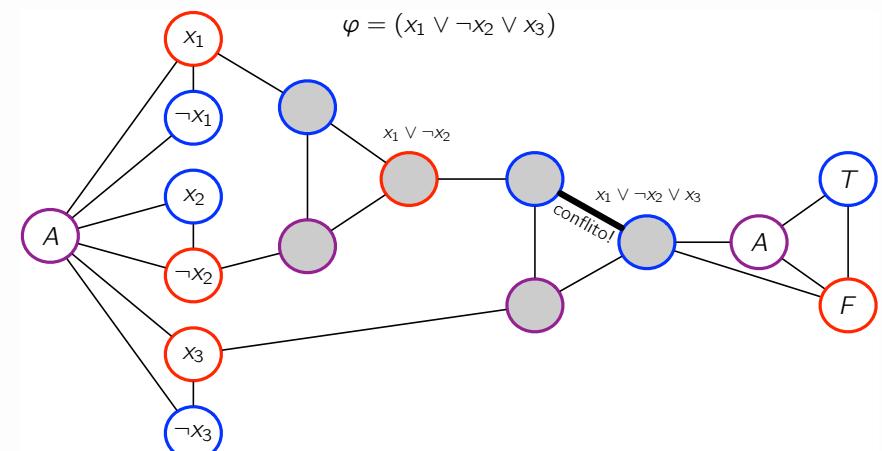
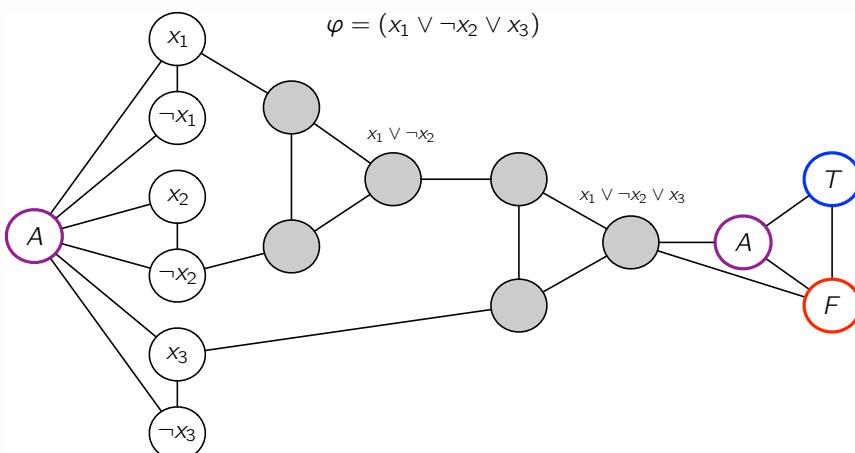
Caso contrário coloração é válida

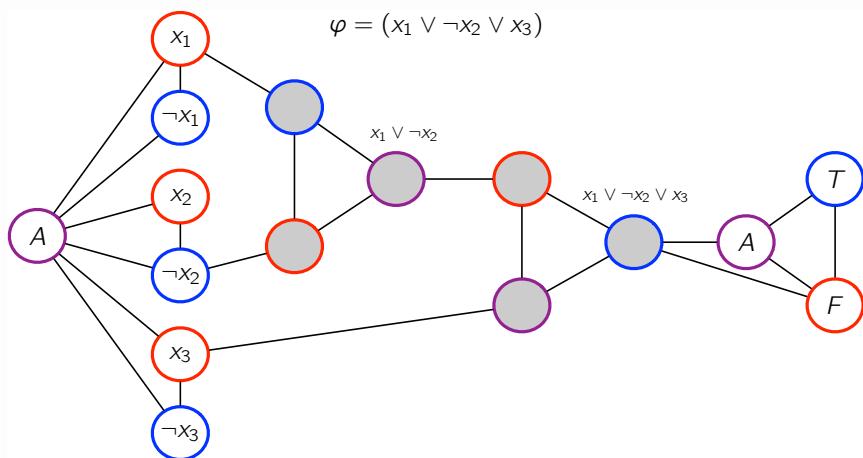
Tempo de execução: $O(V + E)$ **Problema 3COL****Prova:** 3COL é NP-Difícil: $3CNFSAT \leq_P 3COL$ Redução (definição de f):

$$\varphi = \omega_1 \wedge \dots \wedge \omega_m$$

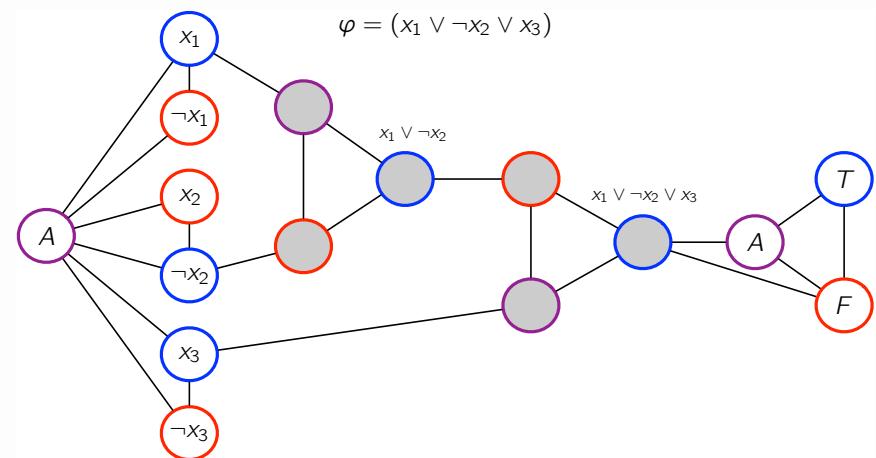
Construção de $G = (V, E)$: cores utilizadas são T, F, A Triângulo M com vértices T, F e A Para cada variável x , construir triângulo com vértices $x, \neg x$ e A
Vértices x e $\neg x$ com cores diferentes entre si e diferentes de A

Para cada cláusula ternária incluir um subgrafo com 5 novos vértices:

um triângulo com nós internos I_1, I_2, I_3 nós internos de I_1 e I_2 ligam a dois vértices da cláusulanó interno I_3 está ligado a nó externo X nó externo X forma triângulo com T e outro nó externo Y nó externo Y ligado a literal e a nó T Se todos os literais F , então impossível saída T



Se algum literal T , então possível saída T



Se todos os literais T , então também é possível saída T

Problema 3COL

Complexidade da função de redução f :

Para cada variável é criado um triângulo

Para cada cláusula é criado um número fixo de arcos e vértices

Construção de G é polinomial (linear) no tamanho da fórmula original φ

Problema 3COL

$3\text{CNFSAT}(x) = 1$ se e só se $3\text{COL}(f(x)) = 1$

φ é satisfeita

Para cada cláusula ω existe literal l com valor T

Atribuir a vértice O associado a l o valor F

Aos restantes vértices O atribuir cor A

Triângulo interno pode ser colorido com 3 cores

G tem uma coloração válida com 3 cores

G pode ser colorido com 3 cores

Admitir φ não satisfeita

Existe ω em que todos os literais têm valor 0

Vértices em G com cor F

Vértices O todos com cor A

Triângulo interno não pode ser colorido com 3 cores; uma contradição