Análise e Síntese de Algoritmos

Complexidade Computacional [CLRS, Cap. 34]

2011/2012

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Árvores abrangentes
  - Caminhos mais curtos
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional
  - Algoritmos de Aproximação

Resumo

Motivação

Prespectiva

- Problemas de decisão
    - Resposta sim(1)/não(0)
- Classe de complexidade P
    - Problemas resolúveis em tempo polinomial
- Codificação de problemas
- Linguagens formais
- Algoritmos de verificação
- Classe de complexidade NP
    - Problemas verificáveis em tempo polinomial
- Redutibilidade entre problemas de decisão
- Problemas NP-completos

Motivação

Algoritmos Polinomiais

- Complexidade em $O(n^k)$
- Todos os algoritmos estudados em ASA (até agora)
- Excepção: algoritmo para o problema da mochila: $O(nW)$; 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

Motivação

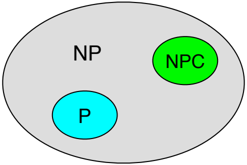
Problemas Resolúveis em Tempo Polinomial vs. NP-completos

- 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 $(\overline{x}_1 + x_2)(x_2 + x_3)(x_1 + \overline{x}_4) = 1$ pode ser feito em tempo polinomial
  - Determinar valores de x_1, x_2, x_3, x_4 que satisfazem $(\overline{x}_1 + x_2 + \overline{x}_4)(x_2 + \overline{x}_3 + x_4) = 1$ é um problema NP-completo

Motivação

Classes de Problemas P, NP e NPC

- Classe **P**: problemas resolúveis em tempo polinomial
- Classe **NP**: problemas verificáveis em tempo polinominal
  - 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



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

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 vértice destino v
- Solução: sequência de vértices do caminho mais curto
- O problema é a relação que associa a cada instância uma ou mais soluções

Problemas Resolúveis em Tempo Polinomial

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

Problemas Resolúveis em Tempo Polinomial

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 $I\!N = \{0, 1, 2, 3, 4, \ldots\}$ nas strings binárias $\{0, 1, 10, 11, 100, \ldots\}$
- Utilizando esta codificação, $e(17) = 10001$

Problemas Resolúveis em Tempo Polinomial

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

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

Problemas Resolúveis em Tempo Polinomial

Codificação de Problemas

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

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

Problemas Resolúveis em Tempo Polinomial

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

Problemas Resolúveis em Tempo Polinomial

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

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
que consiste em, no máximo, k arcos$\}$

Problemas Resolúveis em Tempo Polinomial

Utilização de Linguagens Formais

- 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: $L = \{x \in \{0,1\}^* : A(x) = 1\}$
- L é decidida por A se qualquer string $x \in \{0,1\}^*$ é aceite ou rejeitada
- L aceite/decidida em tempo polinomial se A tem tempo de execução em $O(n^k)$, com $n = |x|$

Problemas Resolúveis em Tempo Polinomial

Definições Alternativas para a Classe 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$\}$
    - 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
    - A aceita L em $O(n^k)$, pelo que A aceita L em tempo não superior a $T = cn^k$
    - Utilizar A' que executa A e observa resultado após $T = cn^k$
        - Se A aceita, A' aceita; se A não aceita (ainda), A' rejeita

Problemas Verificáveis em Tempo Polinomial

Problemas Verificáveis em Tempo Polinomial

- Objectivo é aferir se uma instância pertence a uma dada linguagem utilizando um certificado (i.e. uma possível solução); não é decidir se uma instância pertence a essa linguagem

Algoritmos de Verificação

Algoritmos de Verificação

Algoritmo de verificação A:

- Argumentos:
  - string x: entrada
  - string y: certificado
- O algoritmo A verifica, para uma entrada x e certificado y, se $A(x,y) = 1$
  - 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: CNFSAT

Classe de complexidade NP

Classe NP

- Classe de complexidade NP:
  - Linguagens que podem ser verificadas por um algoritmo de tempo polinomial A
    - $L = \{x \in \{0,1\}^* :$ existe um certificado $y \in \{0,1\}^*$, com $|y| = O(|x|^c)$, tal que $A(x,y) = 1\}$
    - $L \in NP$
    - A verifica L em tempo polinomial
- Classe co-NP:
  - Linguagens L tal que $\bar{L} \in NP$
- Exemplo: CNFUNSAT

Classe de complexidade NP

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

Redutibilidade e Completude-NP

Relações entre classes de complexidade

- Noção de redução entre problemas
- Definição de problemas NP-Completos
- Um problema NP-completo
- Provar problemas NP-completos

Redutibilidade e Completude-NP

Redutibilidade

- 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 $z \in Z$:
    - $Z(z) = 1$ se e só se $X(x) = X(f(z)) = 1$
- 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

Completude-NP

- Um problema de decisão X diz-se NP-completo se:
  - $X \in NP$
  - $Z \leq_P X$ para qualquer $Z \in NP$
- Um problema de decisão X diz-se NP-difícil se:
  - $Z \leq_P X$ para qualquer $Z \in NP$
- NPC: classe de complexidade dos problemas de decisão NP-completos

Completude-NP

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 NP-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 tal que $Y \leq_P X$, em que $Y \in NPC$. Se $X \in NP$, então $X \in NPC$
  - $Y \in NPC$
    - $\forall Z \in NP, Z \leq_P Y$
  - Notando que $\leq_P$ é transitiva e que $Y \leq_P X$, obtemos:
    - $\forall Z \in NP, Z \leq_P X$
  - Deste modo:
    - $X \in NP$
    - $\forall Z \in NP, Z \leq_P X$
  - Pelo que $X \in NPC$!

Completude-NP

Provar Problemas NP-Completos

- Abordagem para provar $X \in NPC$:
  - Provar que $X \in NP$
  - Escolher $Y \in NPC$
  - Descrever um algoritmo que calcula função f, a qual converte qualquer instância de Y numa instância de X, $Y \leq_P X$
  - 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 ?

Completude-NP

Problema NP-Completo

- Problema de decisão: SAT
  - Fórmula proposicional ϕ:
    - variáveis proposicionais, $x_1, \ldots, x_n$
    - 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 SAT: determinar se uma instância ϕ é satisfeita
    - SAT = $\{\langle\phi\rangle : \phi$ é uma fórmula proposicional satisfeita$\}$

Completude-NP

Problema NP-Completo

- $SAT \in NP$:
  - 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]
- $\therefore$ SAT é NP-completo

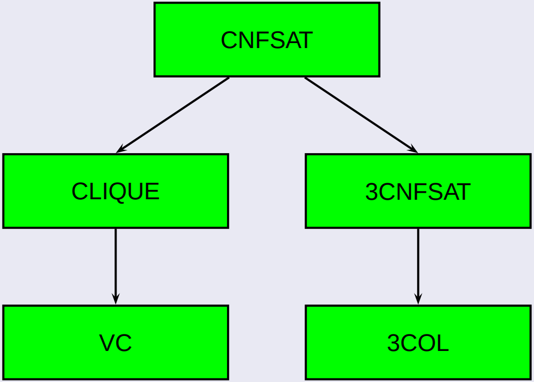
Completude-NP

Problema CNFSAT

- Problema de decisão: CNFSAT
  - Fórmula CNF (conjunctive normal form) ϕ:
    - variáveis proposicionais, $x_1, \ldots, x_n$
    - conectivas proposicionais: $\wedge, \vee, \urcorner$
    - fórmula é conjunção $\wedge$ de disjunções $(\vee)$ de literais
    - literal: variável (x_i) ou o seu complemento $(\urcorner x_i)$
  - 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 CNFSAT: determinar se uma instância ϕ é satisfeita
    - CNFSAT = $\{\langle\phi\rangle : \phi$ é uma fórmula CNF satisfeita$\}$
- CNFSAT é NP-Completo ($SAT \leq_P CNFSAT$)

Completude-NP

Provar Problemas NP-Completos



Completude-NP

Problema 3CNFSAT

- Definição:
  - 3CNFSAT é uma restrição do problema CNFSAT em que cada cláusula contém exactamente 3 literais
- Teorema:
  - O problema 3CNFSAT é NP-completo

Completude-NP

Problema 3CNFSAT

3CNFSAT $\in NP$

- φ: instância de 3CNFSAT
  - Atribuição de valores:
- $(x_1, v_1), \ldots, (x_n, v_n)$
- Calcular valor de cada disjunção e da conjunção
- Complexidade: $O(|\varphi|)$
  - Polinomial no tamanho da instância

Completude-NP

Problema 3CNFSAT

3CNFSAT é NP-Difícil: CNFSAT $\leq_P$ 3CNFSAT

- Redução (definição de f):
  - Dada instância $\varphi \in$ CNFSAT, derivar instância $\varphi_3 \in$ 3CNFSAT
  - 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 $w = (l_1 \vee l_2 \vee l_3 \vee \ldots \vee l_k)$:
    - Criar cláusulas: $(l_1 \vee l_2 \vee y_1) \wedge (\neg y_1 \vee l_3 \vee y_2) \wedge \ldots \wedge (\neg y_{k-4} \vee l_{k-2} \vee y_{k-3}) \wedge (\neg y_{k-3} \vee l_{k-1} \vee l_k)$, com variáveis adicionais $y_1, y_2, \ldots, y_{k-3}$

Completude-NP

Problema 3CNFSAT

Complexidade da função de redução f:

- Número de variáveis adicionais é $O(|\varphi|)$
- Número de cláusulas adicionais é $O(|\varphi|)$
- Complexidade da redução é $O(|\varphi|)$

Completude-NP

Problema 3CNFSAT

3CNFSAT(x) = 1 se e só se 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 $(l_1 \vee l_2 \vee l_3 \vee \ldots \vee l_k)$ tem de ter valor 1
  - Caso contrário $(y_1) \wedge (\neg y_1 \vee y_2) \wedge \ldots \wedge (\neg y_{k-4} \vee y_{k-3}) \wedge (\neg y_{k-3})$ teria que ser satisfeita; impossível
  - Pelo que $\varphi = 1$

Completude-NP

Problema 2CNFSAT

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

- Definição:
  - 2CNFSAT é uma restrição do problema CNFSAT em que cada cláusula contém exactamente 2 literais
- Teorema:
  - O problema 2CNFSAT $\in P$
- Prova:
  - Existe algoritmo para decidir 2CNFSAT com tempo de execução linear no tamanho de $|\varphi|, \varphi \in 2CNFSAT$
  - 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

Completude-NP

Problema HornSAT

HornSAT $\in P$

- Definição:
  - HornSAT é uma restrição do problema CNFSAT em que cada cláusula contém não mais do que 1 literal não complementado
- Teorema:
  - O problema HornSAT $\in P$
- Prova:
  - Existe algoritmo para decidir HornSAT com tempo de execução linear no tamanho de $|\varphi|, \varphi \in$ HornSAT
  - Repetidamente satisfazer cláusulas com apenas 1 literal x_i não complementado (i.e. atribuir valor 1 (TRUE) a x_i)
  - Reduzir 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 !

Completude-NP

Problema HornSAT

HornSAT(φ)

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

Completude-NP

Problema CLIQUE

- Definição:
  - $G = (V, E)$, grafo não dirigido; k 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

Completude-NP

Problema CLIQUE

CLIQUE $\in NP$

- 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)

Completude-NP

Problema CLIQUE

CLIQUE é NP-Difícil: CNFSAT $\leq_P$ CLIQUE

- Redução (definição de f):
  - Instância $\varphi = \omega_1 \wedge \ldots \wedge \omega_m$
  - 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 $\neg x$

Completude-NP

Problema 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

Completude-NP

Problema CLIQUE

CNFSAT(x) = 1 se e só se 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

Completude-NP

Problema Vertex Cover (VC)

- Definição:
  - $G = (V, E)$, grafo não dirigido; 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

Completude-NP

Problema Vertex Cover (VC)

VC $\in NP$

- 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: $O(V + E)$

Completude-NP

Problema Vertex Cover (VC)

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

Completude-NP

Problema Vertex Cover (VC)

CLIQUE(x) = 1 se e só se 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 $|V'| = k$

Completude-NP

Problema 3COL

- Definição:
  - $G = (V, E)$, 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
  - Problema 3COL: decidir se G pode ser colorido com 3 cores
- Teorema:
  - O problema 3COL é NP-completo

Completude-NP

Problema 3COL

3COL $\in NP$

- Considerar 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: $O(V + E)$

Completude-NP

Problema 3COL

3COL é NP-Difícil: 3CNFSAT $\leq_P$ 3COL

- Redução (definição de f):
- $\varphi = \omega_1 \wedge \ldots \wedge \omega_m$
- Construção de $G = (V, E)$
  - Cores utilizadas são: T, F, A
- Triângulo M com vértices T, F e A
- Para cada variável x, construir um 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:
  - um triângulo com nós internos I
  - cada nó interno ligado a nó externo O
  - nó externo ligado a literal e a nó T

Completude-NP

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

Completude-NP

Problema 3COL

CNFSAT(x) = 1 se e só se 3COL($f(x)$) = 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