Teoria da Complexidade

Sumário

Classificação de Problemas

- A maioria dos problemas estudados são polinomiais, isto é, possuem complexidade de tempo $O(n^k)$ para algum k

Questão:

É possível resolver qualquer problema em tempo polinomial?

– Não.

– Exemplo: geração de permutações, ou *the halting problem*

- Dada a descrição de um programa de computador, decidir se o programa terminar a execução ou continua a executar para sempre.
- Isto é equivalente ao problema de decidir, dado um programa e uma entrada, se o programa irá eventualmente parar quando executado com que a entrada, ou vai executar para sempre.

Introdução

Classificação dos Problemas:

- Tratáveis ("fáceis"): podem ser resolvidos por um algoritmo de tempo polinomial;
- Intratáveis ("difíceis"): só podem ser resolvidos por algoritmos superpolinomiais (exponenciais, fatoriais, etc.)
- A complexidade de tempo da maioria dos problemas é polinomial ou exponencial.

Introdução

Polinomial:

Função de complexidade é O(p(n)), em que p(n) é um polinômio.

- Ex.: algoritmos com pesquisa binária ($O(logn)$), pesquisa sequencial ($O(n)$), ordenação por inserção ($O(n^2)$), e multiplicação de matrizes ($O(n^3)$).

Exponencial:

Função de complexidade é $O(c^n), c > 1$.

- Ex.: problema do caixeiro-viajante (PCV) ($O(n!)$).
- Mesmo problemas de tamanho pequeno a moderado não podem ser resolvidos por algoritmos não-polinomiais.

Classe NP

Problemas NP:

- A teoria de complexidade a ser apresentada não mostra como obter algoritmos polinomiais para problemas que demandam algoritmos exponenciais, nem afirma que não existem.
- É possível mostrar que os problemas para os quais não há algoritmo polinomial conhecido são computacionalmente relacionados.
- Formam a classe conhecida como NP.
- Propriedade: um problema da classe N P poderá ser resolvido em tempo polinomial se e somente se todos os outros problemas em N P também puderem.
- Este fato é um indício forte de que dificilmente alguém será capaz de encontrar um algoritmo eficiente para um problema da classe NP.

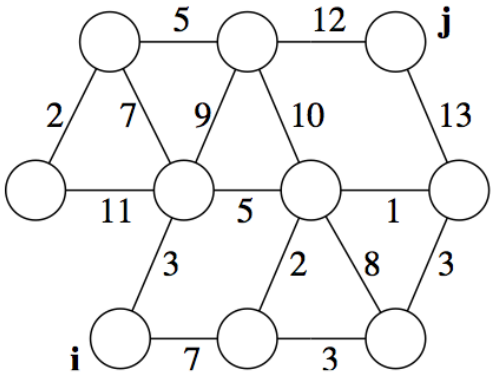
Classe NP

Problemas "Sim/Não":

- Para o estudo teórico da complexidade de algoritmos considera-se problemas cujo resultado da computação seja "sim" ou "não".
- Versão do Problema do Caixeiro-Viajante (PCV) cujo resultado é do tipo "sim/não":
  – Dados: uma constante k, um conjunto de cidades $C = c_1, c_2, \cdots, c_n$ e uma distância $d(c_i, c_j)$ para cada par de cidades $c_i, c_j \in C$.
  – Questão: Existe um "roteiro" para todas as cidades em C cujo comprimento total seja menor ou igual a k?
- Característica da classe NP: problemas "sim/não" para os quais uma dada solução pode ser verificada facilmente.
- A solução pode ser muito difícil ou impossível de ser obtida, mas uma vez conhecida ela pode ser verificada em tempo polinomial.

Classe NP

Considere um grafo com peso nas arestas, dois vértices i, j e int $k > 0$.



Problemas "Sim/Não":

- Fácil: Existe um caminho de i até j com peso $\leq k$?
  – Há um algoritmo eficiente com complexidade de tempo $O(AlogV)$, sendo A o número de arestas e V o número de vértices (algoritmo de Dijkstra).
- Difícil: Existe um caminho de i até j com peso $\geq k$?
  – Não existe algoritmo eficiente. É equivalente ao PCV em termos de complexidade.

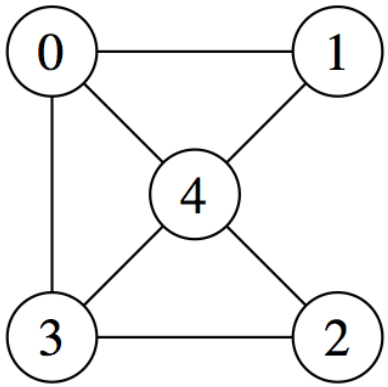
Classe NP

Problema da Coloração de um Grafo:

- O problema é produzir uma coloração ótima, que é a que usa apenas $X(G)$ cores (número cromático).
- Formulação do tipo "sim/não":
  – Dados G e um inteiro positivo k, existe uma coloração de G usando k cores?
  - Fácil: k = 2.
  - Difícil: k > 2.
- Aplicação: modelar problemas de agrupamento (*clustering*) e de horário (*scheduling*).

Classe NP

Considere um grafo abaixo. Existe um ciclo de hamilton no grafo?

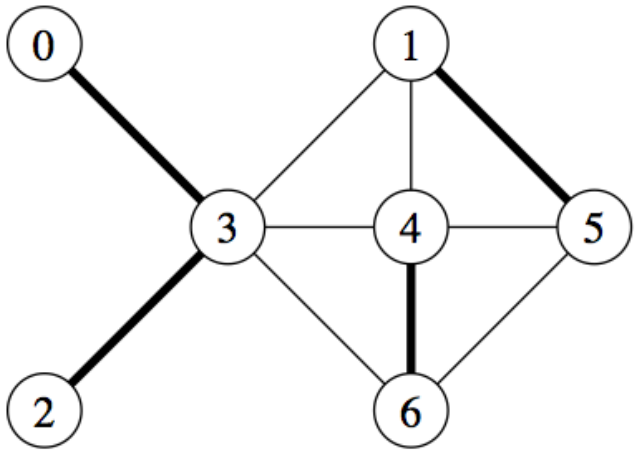


Problemas "Sim/Não":

- Fácil: Grafos com grau máximo = 2 (vértices com no máximo duas arestas incidentes).
- Difícil: Grafos com grau > 2.
- É um caso especial do PCV.
  - Pares de vértices com uma aresta entre eles tem distância 1
  - Pares de vértices sem aresta entre eles têm distância infinita.

Classe NP

Considere um grafo abaixo e um inteiro $k > 0$.



Problemas "Sim/Não":

- Cobertura de k vértices ou k arestas.
  – C. Arestas: para $k = 4$ é $A' = \{(0,3),(2,3),(4,6),(1,5)\}$ – C. Vértices: para $k = 3$ é $V' = \{3,4,5\}$
- Fácil: há uma cobertura de arestas $\leq k$?
- Difícil: há uma cobertura de vértices $\leq k$?

Problemas NP-Completos

- Ainda não foi descoberto um algoritmo polinomial para qualquer problema NP-Completo
- Ninguém até hoje foi capaz de demonstrar formalmente que os problemas NP-Completos são intratáveis
- Muitos problemas NP-Completos são muito parecidos com problemas polinomiais conhecidos:

Classes P, NP e NPC

Classe P

Está na classe P se puder ser resolvido em tempo polinomial.

Classe NP

Está na classe NP se puder ser verificado em tempo polinomial.

Classe NP-Completo

Um problema P está na classe NP-Completo se:

- $P \in NP$
- Qualquer problema da classe NPC puder ser resolvido em tempo polinomial se e somente se P puder ser resolvido em tempo polinomial.

- Um dos principais problemas em aberto na Computação é determinar se $P = NP$.

Decisão × Otimização

Problemas de Otimização:

Dado um conjunto de soluções viáveis, encontrar a solução mínima ou a solução máxima.

Exemplos:

- Encontrar um menor caminho de v_1 para v_2
- Encontrar um conjunto dominante mínimo
- Encontrar um conjunto independente máximo

Problemas de Decisão:

Respostas possíveis são SIM ou NÃO.

Exemplos:

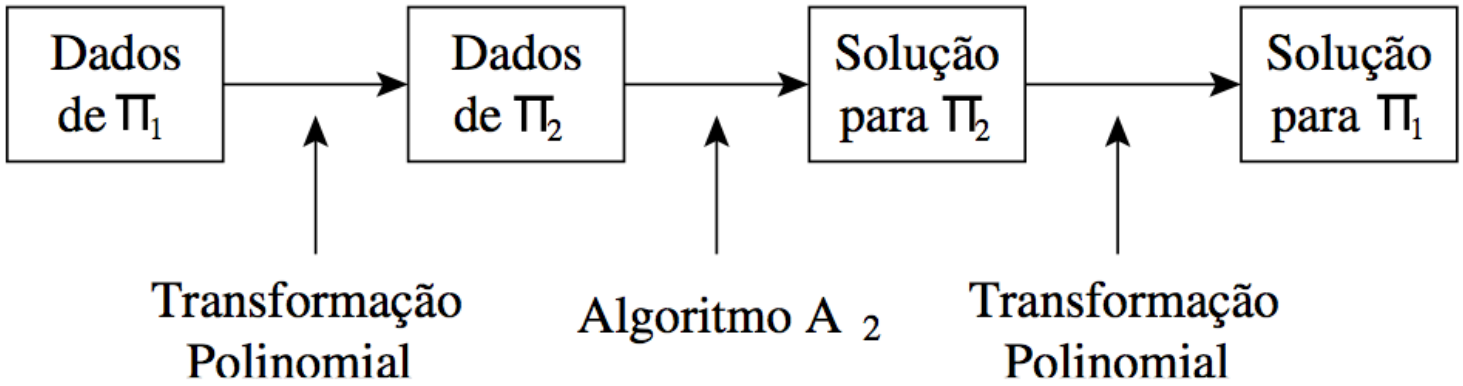
- Existe um caminho de v_1 para v_2 com distância menor que k?
- Existe um conjunto dominante com até k elementos?
- Existe um conjunto independente com pelo menos k elementos?

Reduções

- Sejam A e B problemas de decisão
- Sejam α uma instância de A e β uma instância de B
- Suponha que B possa ser resolvido em tempo polinomial
- Suponha um algoritmo P que transforme α em β, com as seguintes propriedades:
  - a transformação é feita em tempo polinomial
  - as respostas são as mesmas, isto é, a resposta de β é SIM se e somente se a resposta para α for SIM
- Um algoritmo polinomial para resolver α consiste em:
  - aplique P para transformar α em β
  - resolva β
  - use a resposta de β como resposta para α

Reduções

- Suponha que existam dois problemas de "sim/não" Π_1 e Π_2
- Suponha que um algoritmo A_2 resolva Pi_2
- Se for possível transformar Pi_1 em Pi_2 e a solução de Pi_2 em solução de Pi_1, então A_2 pode ser utilizado para resolver Pi_1.
- Se pudermos realizar as transformações nos dois sentidos em tempo polinomial, então Pi_1 é polinomialmente transformável em Pi_2.



- Esse conceito é importante para definir a classe NP-completo

Reduções

- Considere Pi_1 o problema clique e Pi_2 o problema conjunto independente de vértices.
- A instância I de clique consiste de um grafo $G = (V, A)$ e um inteiro $k > 0$.
- A instância $f(I)$ de conjunto independente pode ser obtida considerando-se o grafo complementar $\bar{G}$ de G e o mesmo inteiro k.
- $f(I)$ é uma transformação polinomial:
  1. $\bar{G}$ pode ser obtido a partir de G em tempo polinomial.
  2. G possui clique de tamanho $\geq k$ se e somente se G possui conjunto independente de vértices de tamanho $\geq k$.

Reduções

- Se existe um algoritmo que resolve o conjunto independente em tempo polinomial, ele pode ser utilizado para resolver clique também em tempo polinomial.
- Diz-se que clique $\leq_p$ conjunto independente.
- Denota-se $\Pi_1 \leq_p \Pi_2$ para indicar que Π_1 é polinomialmente transformável em Π_2.
- A relação $\leq_p$ é transitiva:

$$\Pi_1 \leq_p \Pi_2 \text{ e } \Pi_2 \leq_p \Pi_3 \Rightarrow \Pi_1 \leq_p \Pi_3)$$

NP-Completo

Satisfabilidade de Circuitos

- O primeiro problema que provou-se ser NP-Completo foi o problema da Satisfabilidade de Circuitos, ou simplesmente SAT
- Dado um circuito com n entradas, verificar se algum conjunto de entradas faz com que a resposta do circuito seja 1
- Esboço da técnica utilizada:
  - SAT $\in$ NP
  - Dado um problema NP-Completo, provou-se que pode-se criar em tempo polinomial um circuito que o resolva
  - A resposta para o problema é exatamente a resposta dada pelo circuito

CNF-SAT e 3-CNF SAT

- Dizemos que uma expressão lógica $\phi(x_1, \cdots, x_n)$ está na Forma Normal Conjuntiva (ou CNF) se tivermos:

$$\phi(x_1, \cdots, x_n) = C_1 \wedge \cdots \wedge C_m$$

onde cada C_i é da forma $C_i = a_{i_1} \vee \cdots \vee a_{ik_i}$ e cada a_{ij} é da forma x_p ou $\neg x_p$

Exemplo:

$$(x_1 \vee x_4) \wedge (\neg x_1 \vee x3 \vee x_4 \vee \neg x_5) \wedge (\neg x3 \vee x_4 \vee x_5)$$

CNF-SAT e 3-CNF SAT

- Uma expressão lógica é 3-CNF se cada C_i contiver exatamente 3 variáveis

Exemplo:

$(x_1 \vee x_4 \vee \neg x_5) \wedge (\neg x_1 \vee x2 \vee \neg x_5) \wedge (\neg x3 \vee x_4 \vee x_5)$

Proposição:

Se ϕ for 3-CNF, então verificar se existe $X \in \{0,1\}^n$, tal que $\phi(X)$ é verdadeiro, é um problema NP-Completo (3-CNF-SAT).

Teorema de Cook

TEOREMA:

Satisfabilidade (SAT) está em $P \Leftrightarrow P = NP$

- Ou seja, se existisse um algoritmo polinomial determinista para satisfabilidade, então todos os problemas em N P poderiam ser resolvidos em tempo polinomial.
- A prova considera os dois sentidos:
  1. SAT está em NP (basta apresentar um algoritmo não-determinista que execute em tempo polinomial). Logo, se $P = NP$, então SAT está em P.
  2. Se SAT está em P, então $P = NP$. A prova descreve como obter de qualquer algoritmo polinomial não determinista de decisão A, com entrada E, uma fórmula $Q(A, E)$ de modo que Q é satisfatível se e somente se A termina com sucesso para E. O tempo necessário para construir Q é $O(p^3(n)log(n))$, em que n é o tamanho de E e $p(n)$ é a complexidade de A.

Teorema de Cook

Prova:

- A prova, bastante longa, mostra como construir Q a partir de A e E.
- A expressão booleana Q é longa, mas pode ser construída em tempo polinomial no tamanho de E.
- Estabelece uma correspondência entre todo problema em NP (expresso por um programa na MTnd) e alguma instância de SAT.
- Uma instância de SAT corresponde à tradução do programa em uma fórmula booleana.
- A solução de SAT corresponde à simulação da máquina executando o programa em cima da fórmula obtida, o que produz uma solução para uma instância do problema inicial dado.

Reduções

Técnica utilizada para provar que Π_2 é NP-Completo:

- Prove que $\Pi_2 \in$ NP
- Encontre um problema A que seja NP-Completo
- Mostre que Π_1 pode ser transformado em Π_2 em tempo polinomial
- É possível porque Cook apresentou uma prova direta de que SAT é NP-completo, além do fato de a redução polinomial ser transitiva ($SAT \leq_p \Pi_1 \& \Pi_1 \leq_q \Pi_2 \Leftarrow SAT \leq_p \Pi_2$).
- Para ilustrar como um problema Π pode ser provado ser NP-completo, basta considerar um problema já provado ser NP-completo e apresentar uma redução polinomial desse problema para Π.

Reduções

Exemplo:

- Provaremos que determinar se um grafo não dirigido G = (V, E) possui um conjunto independente de tamanho igual a k (CI) é NP-Completo
- Demonstre que CI $\in$ NP
- Sabe-se pela literatura científica que verificar se um grafo não dirigido G = (V, E) possui uma clique de tamanho igual a k (CL) é um problema NP-Completo
- Dada uma instância de CL, isto é, um grafo G qualquer, crie uma instância de CI com C(G), isto é, o complemento de G
- G possui clique de tamanho k se e somente se C(G) possuir um conjunto independente de tamanho k
- Portanto, CL $\leq_p$ CI

Clique é NP-Completo

- CLIQUE: dado um grafo G = (V, E) e um número inteiro k, existe alguma clique em G com tamanho igual a k?
- CLIQUE $\in$ NP: Devemos mostrar um algoritmo polinomial que verifique se $S = \{v_1, \cdots, v_k\}$, $vi \in V$, é uma clique de G
- 3-CNF-SAT $\leq_p$ CLIQUE
  - Seja expressão $\phi(x_i, \cdots, x_n)$ uma expressão lógica 3-CNF contendo k cláusulas
  - construiremos um grafo G = (V, E) que possui uma clique de tamanho k se e somente se ϕ for satisfatível
  - Para cada cláusula $C_r = a_1^r \wedge a_1^r \wedge a_3^r$, inserimos três vértices v_1^r, v_2^r, v_3^r em V
  - a aresta $(v_i^r, v_j^s) \in E$ se $r \neq s$ e v_i^r não é a negação de v_j^s

Clique é NP-Completo

- Se G contiver um clique de tamanho k, então ϕ é satisfatível

Exemplo:

$\phi(x_1, x_2, x_3) =$
$(x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$

- O algoritmo é uma redução, pois uma clique com k elementos garante um elemento verdadeiro em cada uma das cláusulas C_i
- A redução é polinomial:
  - O número de vértices é igual a $3k$ e o número de arestas é no máximo $\frac{9}{2}k(k-1)$
  - Verificar se existe aresta (u, v) demora um tempo O(1)

PCV é NP-Completo

- O Problema do Caixeiro Viajante (PCV) consiste em determinar se existe uma rota do CV em Kn com custo total igual a k
- PCV $\in$ NP
- Sabe-se pela literatura científica que verificar se um grafo G possui um circuito hamiltoniano (CIRC-HAMILTON) é NP-Completo
- CIRC-HAMILTON $\leq_p$ PCV:
  - Seja G = (V, E) um grafo com n vértices
  - Crie um grafo completo contendo os vértices de G, e associe a cada aresta (u, v) o peso 0 se $(u, v) \in E$ ou 1 se $(u, v) \notin E$
  - Existe um circuito hamiltoniano em G se e somente se houver solução para a instância de PCV dada pelo grafo criado acima e k = 0

PPLI-$\{0,1\}$ é NP-Completo

- O Problema de Programação Linear Inteira 0-1 (PPLI-$\{0,1\}$) consiste em determinar se existe algum $X \in \{0,1\}^n$ que satisfaça m restrições da forma:

$$a_{11}x_1 + \cdots + a_{1n}x_n \geq b_1 \quad (1)$$

$$\vdots + \ddots + \vdots \quad (2)$$

$$a_{m1}x_1 + \cdots + a_{mn}x_n \geq b_m \quad (3)$$

- PPLI-$\{0,1\} \in$ NP
  Basta mostrar um algoritmo que verifica se um dado $X \in \{0,1\}^n$ atende todas as m restrições do problema

3-CNF SAT $\leq_p$ PPLI-$\{0,1\}$

- Seja a expressão $\phi(x_i, \cdots, x_n)$ uma expressão lógica 3-CNF contendo m cláusulas
- Seja $C_i = u_1 \vee u_2 \vee u_3$ uma cláusula de ϕ em que cada ui representa x_k ou $\neg x_k$
- Criaremos uma restrição $R_i \equiv w_1 + w_2 + w_3 \geq 1$, em que cada w_i é definido por:

$$w_i = \begin{cases} y_k, & \text{se } u_i = x_k \\ 1 - y_k, & \text{se } u_i = \neg x_k \end{cases} \tag{4}$$

- Seja π um PPLI-$\{0,1\}$ com as restrições $R_1, \cdots, R_m$
- Existe solução para π se e somente se ϕ for satisfatível
- A redução é feita em tempo polinomial

Problemas Relacionados

- Seja G = (V, E) um grafo não dirigido e S um subconjunto de V
- As três proposições abaixo são equivalentes:
  - S é uma clique de G
  - S é um conjunto independente de C(G)
  - V - S é uma cobertura de vértices de C(G)
- Ao provar que algum dos problemas acima é NP-completo, estaremos provando que todos são NP-Completos

Problema da Mochila

- Sejam $S = \{x_1, \cdots, x_n\}$ um conjunto de objetos, cada objeto x_i com peso $p(x_i)$ e com valor $v(x_i)$, e dois números C e L
- O Problema da Mochila consiste em encontrar um subconjunto T de S tal que

$$\sum_{x \in T} p(x_i) \leq C \tag{5}$$

e (6)

$$\sum_{x \in T} v(x_i) \geq L \tag{7}$$

- O Problema da Mochila é NP-Completo

Problema *Bin-packing*

- Sejam $S = \{x_1, \cdots, x_n\}$um conjunto de objetos de peso $p(x_i)$, e dois números C e k
- O Problema Bin-packing consiste em determinar se há uma partição de S, $P = \{S_1, \cdots, S_k\}$, contendo k elementos, tal que para todo $S_i \in P$,

$$\sum_{x \in S_i} p(x_i) \leq M \tag{8}$$

- O Problema Bin-packing é NP-Completo