Análise e Técnicas de Algoritmos

Jorge Figueiredo

NP-Completude

Agenda

- Conceitos básicos
- Classes de Complexidade
  - P
  - NP
- Redução
- Problemas NPC

Introdução

- Existem alguns problemas computacionais que são difíceis de serem resolvidos.
- Impossível de se provar que não existe solução *eficiente*.
- Que conclusões tirar da tabela abaixo?

Funções de Complexidade	Tamanho da Entrada (n)					
	10	20	30	40	50	60
n	.00001s	.00002s	.00003s	.00004s	.00005s	.00006s
n^2	.0001s	.0004s	.0009s	.0016s	.0025s	.0036s
n^3	.001s	.008s	.027s	.064s	.125s	.216s
n^5	.1s	3.2s	24.3s	1.7min	5.2min	13.0min
2^n	.001s	1.0s	17.9min	12.7dias	35.7anos	366sec
3^n	0.059s	58min	6.5anos	2855sec	$2x10^8$ sec	$1.3x10^{13}$sec

Eficiência

- Como definir a eficiência de uma solução?
  - Vimos em nosso curso a conveniência de se utilizar medidas de complexidade como medida de eficiência.
- Um algoritmo é eficiente quando a sua complexidade for polinomial em relação ao tamanho de sua entrada.
  - Um algoritmo é dito ser de tempo polinomial se for $O(n^k)$, para alguma constante $k > 0$.
  - Qualquer outro algoritmo que não for polinomial é dito ser exponencial.
  - Classificação não é absoluta.
  - Algumas vezes pode ser insatisfatória mas, na maioria dos casos, é aceitável.

Intratabilidade de Problemas

- Um problema é dito **tratável** se ele apresenta uma solução polinomial.
- Um problema é **intratável** se ele for tão difícil que nenhum algoritmo polinomial pode resolvê-lo.
- Alguns algoritmos polinomiais podem não ser muito úteis. Por exemplo, se for $O(n^{100})$. Na prática, porém, quase sempre os polinômios são de grau 2 ou 3.
- Alguns problemas são tão difíceis que são indecidíveis. Por exemplo, o problema da parada.
- Por outro lado, alguns problemas são decidíveis mas, intratáveis.

NP-Completude

- Vamos estudar certos problemas que são, de fato, *difíceis* (computacionalmente) de se resolver.
- Esse é a idéia central da teoria de **NP-Completude**.
- Vamos mostrar que encontrar uma solução eficiente para um certo problema é tão difícil quanto encontrar soluções eficientes para todos os problemas definidos em uma classe de problemas que chamamos **NP**.

Problema Algorítmico

- Caracterizado por:
  - Conjunto de dados.
  - Objetivo do problema.
  - Solução.
- Exemplo: Encontrar um clique de tamanho k em um grafo G.
  - Conjunto de dados: um grafo G e um inteiro k > 0.
  - Objetivo do problema: a própria definição.

Classes de Problemas

- **Problemas de Decisão:** Problemas em que a saída (solução) é *SIM* ou *NÃO*.
- **Problemas de Localização:** Determinar uma certa estrutura que satisfaça um conjunto de propriedades dadas.
- **Problemas de Otimização:** Problemas de localização em que as propriedades satisfazem critérios de otimização.
- Todos os problemas que vamos utilizar no estudo de NP-Completude são problemas de decisão.

Classes de Problemas

- É possível relacionar problemas de otimização e localização com problemas de decisão. Por exemplo:

Problema de Decisão:
Dados: Um grafo G e um inteiro k>0.
Objetivo: Verificar se G possui um percurso de caixeiro viajante de peso ≤k.

Problema de Localização:
Dados: Um grafo G e um inteiro k>0.
Objetivo: Localizar, em G, um percurso de caixeiro viajante de peso ≤k.

Problema de Otimização:
Dados: Um grafo G.
Objetivo: Localizar, em G, um percurso de caixeiro viajante ótimo.

Classes de Complexidade P e NP

- A **Classe de Complexidade P** (polynomial time) é o conjunto de todos os problemas de decisão que são resolvíveis em tempo polinomial em um computador determinístico.
- A **Classe de Problemas NP** (nondeterministic polynomial time) é o conjunto de problemas resolvíveis em tempo polinomial em um computador não-determinístico. A Classe NP também pode ser definida como a classe de problemas em que é possível verificar em tempo polinomial, se um determinada solução proposta satisfaz o problema de decisão.

Não Determinismo

- Veja um computador não determinístico como sendo um computador que "magicamente adivinha" uma solução:
  - Se a solução existe, o computador sempre adivinha.
- Outra forma de definir: Um computador paralelo que executa infinito número de processos
  - Um processador para cada solução possível

Classes de Complexidade Co-NP

O *complemento* de um problema de decisão D é um problema de decisão cujo objetivo é o complemento da decisão de D.

A **Classe Co-NP** compreende exatamente os complementos dos problemas da classe NP.

Relação entre Classes de Complexidade

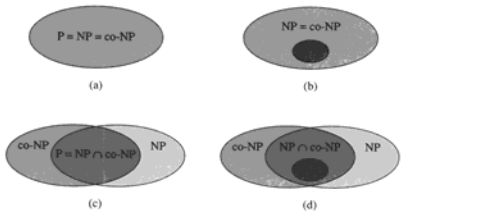


Figure 34.3 Four possibilities for relationships among complexity classes. In each diagram, one region enclosing another indicates a proper-subset relation. **(a)** P = NP = co-NP. Most researchers regard this possibility as the most unlikely. **(b)** If NP is closed under complement, then NP = co-NP,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

Relação entre Classes de Complexidade

- **P** = conjunto de problemas que podem ser resolvidos em tempo polinomial
- **NP** = conjunto de problemas cuja solução pode ser verificada em tempo polinomial
- **P** ⊆ **NP**
- Questão não resolvida: **P = NP**? ou **P ≠ NP**?
- Se **P ≠ NP ,** NP – P são problemas intratáveis.

Alguns Problemas de Decisão em NP

Problema 1: Satisfiability

- O problema da *satisfiability* (SAT) é um problema de lógica que envolve expressões booleanas. Pode ser definido da seguinte forma:
  - Dados: Uma expressão booleana E na sua forma normal conjuntiva (FNC).
  - Objetivo: Verificar se E é satisfeita, ou seja, verificar se existe uma atribuição de valores às variáveis da expressão de tal modo que a expressão seja avaliada verdadeira.

Uma expressão é FNC quando for uma conjunção de cláusulas. Por exemplo, $(x2 \vee \neg x1) \wedge (x1 \vee \neg x3 \vee \neg x2) \wedge (x3) \wedge (x1 \vee x3 \vee x2)$.

Alguns Problemas de Decisão em NP

Problema 2: Conjunto Independente de Vértices

- Dados: Um grafo G e um inteiro k>0.
- Objetivo: Verificar se G possui um conjunto independente de vértices de tamanho ≥k.

Dado um grafo G= (V,E), um conjunto independente de vértices é um subconjunto $V_{IND} \subseteq V$ tal que qualquer par de vértices de V_{IND} não é adjacente.

Alguns Problemas de Decisão em NP

Problema 3: Clique

- Dados: Um grafo G e um inteiro k>0.
- Objetivo: Verificar se G possui um clique de tamanho ≥k.

Dado um grafo G= (V,E), um clique é um subconjunto $V_{CLI} \subseteq V$ tal que qualquer par de vértices de V_{CLI} é adjacente.

Alguns Problemas de Decisão em NP

Problema 4: Cobertura de Vértices

- Dados: Um grafo G e um inteiro k>0.
- Objetivo: Verificar se G possui uma cobertura de vértices de tamanho ≤k.

Dado um grafo G= (V,E), uma cobertura de vértices é um subconjunto $V_{COB} \subseteq V$ tal que qualquer aresta de G é incidente a um vértice de V_{COB}.

Alguns Problemas de Decisão em NP

Problema 5: Coloração

- Dados: Um grafo G e um inteiro k>0.
- Objetivo: Verificar se G possui uma coloração com um número de cores ≤k.

Dado um grafo G= (V,E), uma coloração é atribuída a G de tal forma que dois vértices adjacentes tenham cores distintas.

Problemas NP-Completos

•A classe de problemas NP captura o conjunto de problemas que acreditamos que sejam difíceis de se tratar.

•Existem, entretanto, problemas que podem ser considerados pelo menos tão difíceis como qualquer outro em NP.

•Essa classe de problemas mais difíceis em NP é chamada de NP-Completo ou NPC.

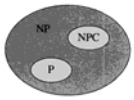


Figure 34.6 How most theoretical computer scientists view the relationships among P, NP, and NPC. Both P and NPC are wholly contained within NP, and $P \cap NPC = \emptyset$.

Redução em Tempo Polinomial

- Um problema P pode ser *reduzido* a um outro problema Q se qualquer instância de P pode ser refraseada (transformada) para uma instância de Q.
- Intuitivamente: se P é redutível em tempo polinomial a Q, P não é mais difícil de resolver do que Q.

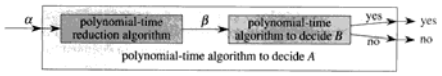


Figure 34.1 Using a polynomial-time reduction algorithm to solve a decision problem A in polynomial time, given a polynomial-time decision algorithm for another problem B. In polynomial time, we transform an instance α of A into an instance β of B, we solve B in polynomial time, and we use the answer for β as the answer for α.

Teorema de Cook

- SAT é NP-Completo.
- SAT tem a propriedade que todos os problemas em NP podem ser reduzidos a ele, em tempo polinomial.
- Existem outros problems em NP que têm as mesmas características de SAT.
- Se D1 ∈ NPC e D2 ∈ NPC, então D1 $\leq_p$ D2 e D2 $\leq_p$ D1.
- É possível concluir:
  - Se um problema NPC pode ser resolvido em tempo polinomial, então todos os problemas de NP podem sê-lo.
  - Se um problema de NP é intratável, todos os problemas de NPC são intratáveis.

Como Determinar se um Problema é NPC?

- Se P, Q ∈ NP, P é NPC e P $\leq_p$ Q, então Q é NP-Completo.
- Para provar se um problema de decisão D é NP-Completo, devemos seguir os seguintes passos:
  1. Mostra que D ∈ NP.
  2. Selecionar, D1, um problema NPC conhecido.
  3. Achar uma redução de D1 $\leq_p$ D para .
  4. Mostrar que a redução foi feita em tempo polinomial.

Exemplos de Problemas NPC: 3SAT

- O problema 3SAT consiste em determinar o resultado de uma expressão booleana E que está escrita em sua FNC é satisfeita.
- Cada cláusula de E tem exatamente três literais. Por exemplo:
  - (v1 ∨ ¬v2 ∨ v7)∧(v3 ∨ v5 ∨ ¬v6)∧(v1 ∨ ¬v5 ∨ ¬v8) é uma instância de 3SAT.
- Aplicando os passos para determinar se 3SAT é NPC temos:
  - 3SAT é claramente NP.
  - SAT é NPC. Se SAT $\leq_p$ 3SAT. (vamos mostrar como a redução é feita)
  - Se a redução é feita em tempo polinomial, 3SAT é NPC.

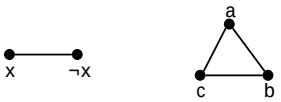
3SAT

Para fazer a redução, devemos substituir cada cláusula Ci em E por:

- Se Ci=(a) , devemos trocar por Si = $(a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (a \vee b \vee \neg c) \wedge (a \vee \neg b \vee \neg c)$.
- Se Ci=(a∨b), devemos trocar por Si = $(a \vee b \vee c) \wedge (a \vee b \vee \neg c)$
- Se Ci= (a∨b∨c), não fazemos nada.
- Se Ci= $(a_1 \vee a_2 \vee ... \vee a_k)$, k>3, devemos trocar por Si = $(a_1 \vee a_2 \vee b_1) \wedge (\neg b_1 \vee a_3 \vee b_2) \wedge (\neg b_2 \vee a_4 \vee b_3) \wedge ... \wedge (\neg b_{k-3} \vee a_{k-1} \vee a_k)$.
- As variáveis adicionadas são novas variáveis que não estão sendo usadas.

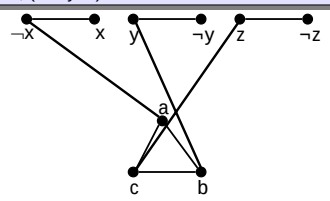
Cobertura de Vértices é NPC

- Reduzir 3SAT para Cobertura de Vértices:
  - Para cada variável x crie um nó para x e ¬x e conecte os dois nós.
  - Para cada cláusula (a∨b∨c), crie um triângulo e conecte os 3 nós.



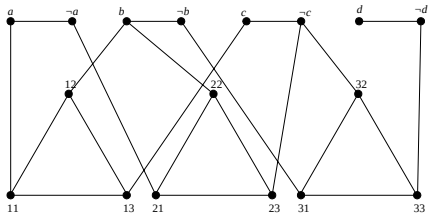
Cobertura de Vértices é NPC

- Complete a construção:
  - Conecte cada literal em um triângulo para o seu correspondente (no par).
  - Seja n o número de variáveis, m o número de cláusulas
  - Fazer k = n + 2m
  - Por exemplo, (¬x∨y∨z)



Cobertura de Vértices é NPC

- Exemplo: (a∨b∨c)∧(¬a∨b∨¬c)∧(¬b∨¬c∨¬d)
- O Grafo tem cobertura de vértice de tamanho k = 4+6 = 10 sss a fórmula é satisfatível



Clique é NPC

- Reduzir a partir de cobertura de vértices.
- O grafo G tem cobertura de vértice de tamanho k sss seu complemento tem um clique de tamanho n-k.