Projeto e Análise de Algoritmos

Prof. Hamilton José Brumatto

Bacharelado em Ciência da Computação - UESC

18 de novembro de 2010

Classes de Problemas

Algoritmos Eficientes

- Foram trabalhados vários problemas, e para muitos deles obtivemos algoritmos eficientes:
  - Localização de um valor em um vetor (ordenado ou não);
  - Ordenação de vetores;
  - Multiplicação de matrizes;
  - Árvore Geradora Mínima de um Grafo;
  - Caminhos mais curtos em grafos;...
- Um algoritmo é eficiente se ele é executado na complexidade de tempo da cota inferior do problema.
- Ou seja, um problema tem um algoritmo eficiente se sua cota inferior for da complexidade de tempo de sua cota superior.

Problemas Tratáveis e Intratáveis

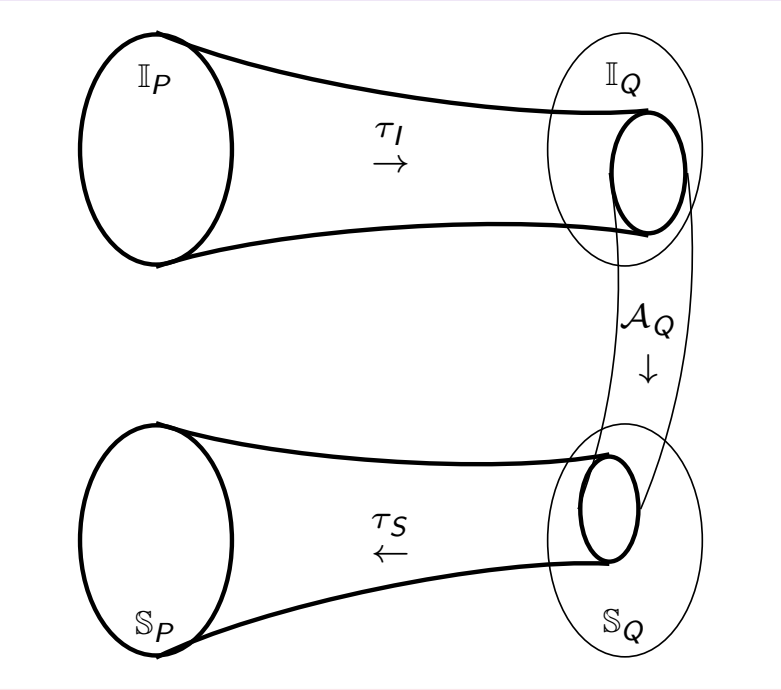
- Os problemas acima, e muitos outros puderam ser resolvidos em tempo polinomial.
- Um algoritmo que resolve um problema em tempo polinomial é $O(n^k)$ para alguma constante k.
- Os problemas resolvidos em tempo polinomial são considerados problemas tratáveis.
- No entanto existem problemas que podem ser resolvidos, mas não no tempo polinomial, estes problemas são considerados intratáveis, ou difíceis.
- Existem até mesmo problemas que não podem ser resolvidos, como o *Problema de parada (halting) de Turing*.

Classes de problemas

- Existe um conjunto de problemas que não se conhece algoritmos eficientes.
- É difícil encontrar um algoritmo polinomial que resolva o problema, mas existe um algoritmo polinomial que **verifica** se uma proposta de solução resolve de fato o problema.
- Costuma-se catalogar os problemas como estando em pelo menos duas classes:
  - A classe dos problemas para os quais se conhece um algoritmo eficiente para resolução
  - A classe dos problemas para os quais se conhece um algoritmo eficiente para verificação
- O estudo é feito tradicionalmente para **Problemas de Decisão**

Tipos de reduções

- Vamos tratar somente com Redução de Karp, que será usada em provas de pertinência de problemas de decisão às diferentes classes de problemas.



$\mathcal{A}_Q$ deve responder *SIM* para $\mathbb{I}_Q$ se e somente se $\mathbb{I}_P$ for uma instância *SIM* para o problema P

Problemas de decisão × Problemas de otimização

- Apesar de tratarmos apenas problemas de decisão, muitos problemas de interesse são problemas de otimização. Por exemplo, o caminho mais curto entre dois pontos.
- Existe um relacionamento conveinente entre problemas de decisão e problemas de otimização. Por exemplo, um problema de decisão é identificar se existe um caminho cujo comprimento no máximo é L.
- Assim, uma vez identificado um problema de decisão em uma classe de problemas, é possível abstrair quão difícil (ou fácil) é um problema de otimização (Por Quê?)
- Lembre-se que é possível reduzir um problema de decisão em um problema de otimização (relacionado àquele problema de decisão)!

Problemas de decisão × Problemas de otimização

- Apesar de tratarmos apenas problemas de decisão, muitos problemas de interesse são problemas de otimização. Por exemplo, o caminho mais curto entre dois pontos.
- Existe um relacionamento conveinente entre problemas de decisão e problemas de otimização. Por exemplo, um problema de decisão é identificar se existe um caminho cujo comprimento no máximo é L.
- Assim, uma vez identificado um problema de decisão em uma classe de problemas, é possível abstrair quão difícil (ou fácil) é um problema de otimização (Por Quê?)
- Lembre-se que é possível reduzir um problema de decisão em um problema de otimização (relacionado àquele problema de decisão)!

Problemas de Tempo Polinomial

- Apesar de enunciado que problemas com cota superior $O(n^k)$ são tratáveis, parece incorreto pensar que um algoritmo $O(n^{100})$ seja algo tratável.
- A experiência mostra que uma vez encontrado um algoritmo polinomial, outros mais eficientes são encontrados.
- Um algoritmo de computador resolve um problema concreto no tempo $O(T(n))$ se, quando é fornecida uma instância de problema i de comprimento $n = |i|$, o algoritmo pode produzir a solução no tempo máximo $O(T(n))$.
- Um problema concreto pode ser resolvido em tempo polinomial se existe um algoritmo para resolvê-lo no tempo $O(n^k)$ para alguma constante k.
- Um problem que possua um algoritmo *determinístico* de tempo polinomial pertence à classe de complexidade $\mathcal{P}$.

Algoritmos não-determinísticos

- Até o momento somente trabalhamos com algoritmos determinísticos. Uma operação por vez e o resultado da operação é interpretado de maneira única.
- No modelo de computação *não-determinístico* existe o comando *Escolha*(S) que retorna um elemento do conjunto S.
- *Escolha* define de forma aleatória um elemento de $|S|$ possíveis, e *Escolha*$(S) \in O(1)$.
- Um algoritmo não-determinístico é construído em duas partes:
  - Construção: irá de forma **não**-determinística construir uma proposta de solução (conhecida como Certificado) para o problema
  - Verificação: irá de forma determinística verificar a validade do Certificado. Se o Certificado resolve o problema, o algoritmo define o resultado Aceitar, caso contrário Rejeitar.

Exemplo de algoritmo não-determinístico

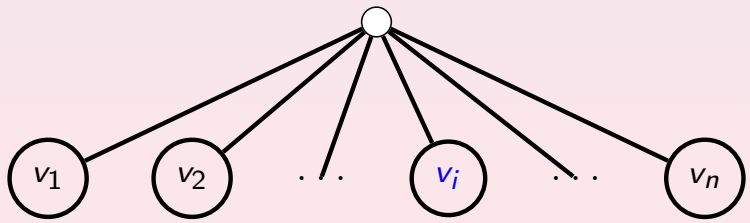
- Determinar se um valor x pertence a um vetor A de n posições.

Algoritmo *BuscaND*(A, x)
1. ▷ Construção
2. $i \leftarrow$ *Escolha*$(1, \ldots, n)$
3. ▷ Verificação
4. **se** $A[i] = x$
5. **então retorne** ACEITAR
6. **senão retorne** REJEITAR

- Como interpretar a execução de um algoritmo não-determinístico?

"Entendendo" uma máquina não-determinística

- Considere um computador com *infinitos* processadores, todos se comunicando com tempo 0 (zero).
- *Escolha*, na Construção, gera um certificado para cada processador.
- Na Verificação, cada processador verifica um certificado, se um certificado resolve o problema, este processador avisa (em tempo 0) os demais e a resposta é ACEITAR



Complexidade de um algoritmo não-determinístico

- Um algoritmo não-determinístico tem complexidade $O(f(n))$ se existem constantes positivas c e n_0 tais que para toda instância de tamanho $n \geqslant n_0$ para o qual ele resulta em ACEITAR, o tempo de execução é limitado a $cf(n)$.
- *Escolha* é $O(1)$.
- A comparação também é $O(1)$, logo este algoritmo é $O(1)$.
- Observe que qualquer algoritmo determinístico para este problema é $\Omega(n)$.
- Como seria um algoritmo não-determinístico para o problema de ordenação?

Complexidade de um algoritmo não-determinístico

- Um algoritmo não-determinístico tem complexidade $O(f(n))$ se existem constantes positivas c e n_0 tais que para toda instância de tamanho $n \geqslant n_0$ para o qual ele resulta em ACEITAR, o tempo de execução é limitado a $cf(n)$.
- *Escolha* é $O(1)$.
- A comparação também é $O(1)$, logo este algoritmo é $O(1)$.
- Observe que qualquer algoritmo determinístico para este problema é $\Omega(n)$.
- Como seria um algoritmo não-determinístico para o problema de ordenação?

Exemplo de algoritmo não-determinístico para partição

- Determinar se um vetor S de n posições pode ser particionado em dois vetores S_1 e S_2 tal que $\sum_{v_i \in S_1} v_i = \sum_{v_i \in S_2} v_i$.

Algoritmo *ParticaoND*(S)
1. ▷ Construção
2. $S_2 \leftarrow S$; $k \leftarrow Escolha(1, \ldots, n-1)$
3. **para** $i \leftarrow 1$ **até** k
4. **faça** $s \leftarrow Escolha(S_2)$; $S_1 \leftarrow S_1 \cup \{s\}$; $S_2 \leftarrow S_2 - \{s\}$
5. ▷ Verificação
6. $s_1 \leftarrow 0$; $s_2 \leftarrow 0$;
7. **para** $i \leftarrow 1$ **até** $tamanho[S_1]$
8. **faça** $s_1 \leftarrow s_1 + S_1[i]$
9. **para** $i \leftarrow 1$ **até** $tamanho[S_2]$
10. **faça** $s_2 \leftarrow s_2 + S_2[i]$
11. **se** $s_1 = s_2$
12. **então retorne** ACEITAR
13. **senão retorne** REJEITAR

- Observe que a complexidade deste algoritmo é $O(n^2)$

As classes $\mathcal{P}$ e $\mathcal{NP}$

- Definição: $\mathcal{P}$ é o conjunto de problemas que podem ser resolvidos por um algoritmo determinístico polinomial.
- Definição: $\mathcal{NP}$ é o conjunto de todos os problemas que podem ser resolvidos por um algoritmo não-determinístico polinomial.
- Todo algoritmo determinístico é um caso particular de um algoritmo não-determinístico, de forma que:

$$\mathcal{P} \subseteq \mathcal{NP}$$

$\mathcal{P} = \mathcal{NP}$?

- Esta é uma questão aberta da Teoria da Computação.
- Até o momento não há prova de que algum problema que pertença à classe $\mathcal{NP}$ não possa ser resolvido por nenhum algoritmo determinístico.
- Também não é possível mostrar que todos os problemas na classe $\mathcal{NP}$ possua algoritmo polinomial.
- No entanto, a tendência é interpretar que a proposição é falsa.

$\mathcal{NP}$-difícil

- Um problema é $\mathcal{NP}$-difícil se ele for tão difícil quanto qualquer problema da classe $\mathcal{NP}$.
- Ou seja, é possível realizar uma redução polinomial (τ_I e τ_S são polinomiais) de qualquer problema da classe $\mathcal{NP}$ a este problema.
- Um problema que é $\mathcal{NP}$-difícil não precisa estar necessariamente na classe $\mathcal{NP}$.

$\mathcal{NP}$-completo

- Um problema A é $\mathcal{NP}$-completo, se:
  - $A \in \mathcal{NP}$ e
  - $A \in \mathcal{NP}$-difícil
- Se um problema é $\mathcal{NP}$-completo, e houver um algoritmo determinístico polinomial que resolve o problema, então todos os problemas da classe $\mathcal{NP}$ podem ser resolvidos por um algoritmos determinísticos de tempo polinomial, ou seja: $\mathcal{P} = \mathcal{NP}$.

Primeiro problema $\mathcal{NP}$-completo: SAT

Problema da Satisfatibilidade de uma Expressão Booleana

Dada uma Fórmula lógica booleana construída com os seguintes elementos:

- Variáveis: $x_1, \ldots, x_n$ (e suas negações: $\bar{x}_i$ para todo i).
- Operadores lógicos: "+" e "." (OU e E lógicos)
- Cláusulas: $C_1, C_2, \ldots, C_m$ da forma $C_i = (x_{i1} + x_{i2} + \ldots)$
- Fórmula: $F = C_1.C_2.\cdots.C_m$

Existe uma instância de entradas nas variáveis de forma que a fórmula retorne *VERDADE*?

- O SAT é $\mathcal{NP}$. Por quê?
- Resta provar que o SAT é $\mathcal{NP}$-difícil.

Primeiro problema $\mathcal{NP}$-completo: SAT

- Como provar que um problema é $\mathcal{NP}$-completo? É preciso saber a dificuldade de todos os problemas da classe $\mathcal{NP}$, conhecidos ou não.
- Cook provou que o problema de satisfatibilidade SAT é $\mathcal{NP}$-completo.
  - De uma forma muito breve, uma característica dos problemas da classe $\mathcal{NP}$ é que existe um algoritmo verificador determinístico polinomial para analisar o certificado.
  - Todo algoritmo eficiente pode ser descrito por instruções da máquina de Turing, logo a máquina de Turing descreve o algoritmo verificador para um problema A de $\mathcal{NP}$.
  - Uma instância x do problema A tem resposta SIM *se e somente se* uma fórmula F tem resposta SIM para *SAT*.
- Será que existem outros problemas $\mathcal{NP}$-completo? ou SAT é o único. Como provar?

Provas de $\mathcal{NP}$-completude

- Depois que Cook (1971) provou que SAT estava em $\mathcal{NP}$-completo, Karp (1972) mostrou que 24 problemas famosos também estavam em $\mathcal{NP}$-completo.
- Para provar que um problema A está em $\mathcal{NP}$-completo é necessário:
  1. Provar que A está em $\mathcal{NP}$.
  2. Provar que A está em $\mathcal{NP}$-difícil: pode ser feito encontrando-se uma redução polinomial de um problema B qualquer em $\mathcal{NP}$-difícil para A.

CLIQUE é um problema $\mathcal{NP}$-completo ?

O problema CLIQUE

Dado um grafo não-orientado $G = (V, E)$ e um valor inteiro $k \in \{1, \ldots, n\}$, onde $n = |V|$, pergunta-se: G possui uma *clique*[a]

[a] Clique de k vértices é um subgrafo completo que possua k vértices

- TEOREMA: CLIQUE $\in \mathcal{NP}$-completo
  1. CLIQUE $\in \mathcal{NP}$. Por quê?
  2. SAT $\propto_{poli}$ CLIQUE

CLIQUE é um problema $\mathcal{NP}$-completo ?

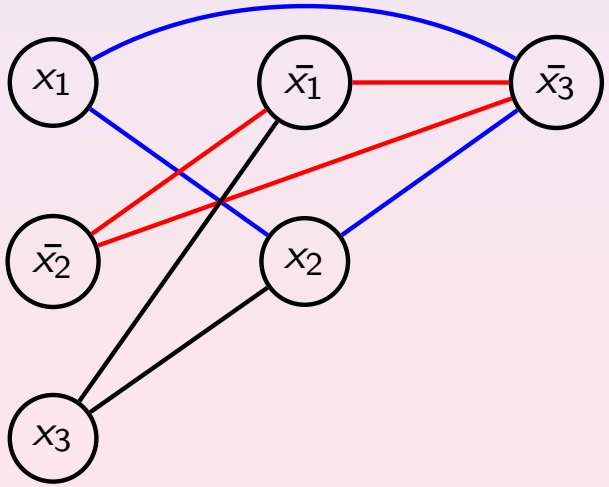
Grafo t-partido

DEFINIÇÃO: um grafo $G = (V, E)$ é t-partido se o conjunto de vértices pode ser particionado em t subconjuntos $V_1, V_2, \ldots, V_t$ tal que não existam arestas em E ligando dois vértices em um mesmo subconjunto V_i, $i \in \{1, \ldots, t\}$.

- Transormação de uma instância SAT em uma instância CLIQUE:
  - Seja $F = C_1.C_2.\cdots.C_c$ uma fórmula booleana nas variáveis $x_1, x_2, \ldots, x_v$. Construa o grafo c-partido $G = ((V_1, V_2, \ldots, V_c), E)$ tal que:
    1. Em um subconjunto V_i existe um vértice associado a cada variável que aparece na cláusual C_i de F.
    2. A aresta (a, b) está em E **se e somente se** a e b estão em subconjuntos distintos **e**, além disso, a e b não representam simultaneamente uma variável e a sua negação.

CLIQUE é um problema $\mathcal{NP}$-completo ?

- O grafo possui $O(cv)$ vértices e $O(c^2v^2)$ arestas, a redução é em tempo polinomial
- SAT tem resposta SIM *se e somente se* o grafo possui uma CLIQUE de c vértices.
- Exemplo: $F = (x_1 + \bar{x_2} + x_3)(\bar{x_1} + x_2)(\bar{x_3})$



CLIQUE é um problema $\mathcal{NP}$-completo ✓

- A prova de que a redução é correta segue nos seguintes argumentos:
- Uma clique só acontece quando cada vértice é adjacente a todos os demais vértices.
- Só um vértice de uma partição participa da clique.
- Vértice que representam contradições (x_i e $\bar{x}_i$) não são adjacente.
- Uma clique representa um conjunto de variáveis que podem ser todas VERDADE ao mesmo tempo.
- Tendo uma variável em cada cláusula (partição) como VERDADE indica que a fórmula é verdade.
- Só existe a clique *se e somente se* SAT é satisfeita.

Outros problemas $\mathcal{NP}$-completos

- Em breve.

Atividades

1. Em breve.