

PROJETO E ANÁLISE DE ALGORITMOS

PARTE V

TEORIA DA COMPLEXIDADE

1. Introdução

- A maioria dos problemas conhecidos e estudados se divide em dois grupos:
 - Problemas cuja solução é limitada por um polinômio de grau pequeno
 - Pesquisa binária: $\Theta(\log n)$
 - Ordenação: $\Theta(n \log n)$
 - Multiplicação de matriz: $\Theta(n^{2.81})$
 - Problemas cujo melhor algoritmo conhecido é não-polinomial
 - Problema do Caixeiro Viajante: $\Theta(n^2 2^n)$
 - Knapsack Problem: $\Theta(2^{n/2})$
- Algoritmos polinomiais são obtidos através de um entendimento mais profundo da estrutura do problema
- Um problema é considerado **tratável** quando existe um algoritmo polinomial para resolvê-lo
- Algoritmos exponenciais são, em geral, simples variação de pesquisa exaustiva
- Um problema é considerado **intratável** se ele é tão difícil que não existe um algoritmo polinomial para resolvê-lo
- Entretanto,
 - Um algoritmo $\Theta(2^n)$ é mais rápido que um algoritmo $\Theta(n^5)$ para $n \leq 20$
 - Existem algoritmos exponenciais que são muito úteis na prática
 - ▶ Algoritmo Simplex para programação linear é exponencial mas, executa muito rápido na prática
 - Na prática os algoritmos polinomiais tendem a ter grau 2 ou 3 no máximo e não possuem coeficientes muito grandes n^{100} ou $10^{99}n^2$ NÃO OCORREM

2. Decisão x Otimização

- Em um problema de **otimização** queremos determinar uma solução possível com o melhor valor.
- Em um problema de **decisão** queremos responder “sim” ou “não”.
- Para cada problema de otimização podemos encontrar um problema de decisão equivalente a ele.

Exemplo: Problema do Caixeiro Viajante

- **Dados:** Um conjunto de cidades $C=\{c_1, c_2, \dots, c_n\}$, uma distância $d(c_i, c_j)$ para cada par de cidades $c_i, c_j \in C$ e uma constante K .
- **Questão:** Existe um roteiro para todas as cidades em C cujo comprimento total seja menor ou igual a K ?

3. Classe P e NP

- Classe P:
 - Um algoritmo está na Classe P se a complexidade do seu pior caso é uma função polinomial do tamanho da entrada de dados
- Classe NP:
 - Classe de problemas “Sim/Não” para os quais uma dada solução pode ser verificada facilmente
 - Existe uma enorme quantidade de problemas em NP para os quais não se conhece um único algoritmo polinomial para resolver qualquer um deles

Exemplo: Ordenação está em P e NP

VOrdenacao (A, n)

inicio

ordenado \leftarrow verdadeiro

para i \leftarrow 1 **até** n-1 **faça**

se A[i] > A[i+1] **então**

 ordenado \leftarrow falso

fim se

fim para

se ordenado = falso **então**

escreva "NAO"

senão

escreva "SIM"

fim se

fim

- Complexidade: $\Theta(n)$

Exemplo: Coloração de Grafos está em NP

VColoracao(G,C,K)

inicio

colorido \leftarrow verdadeiro

para i \leftarrow 1 **até** |E| **faça**

se C[Ei.V1] = C[Ei.V2] **então**

 colorido \leftarrow falso

fim se

fim para

se colorido = verdadeiro **e** |C| \leq K **então**

escreva "SIM"

senão

escreva "NAO"

fim se

fim

- Complexidade: $\Theta(n^2)$, onde n é o número de vértices

4. Algoritmos Não-deterministas

- Um computador não-determinista, quando diante de duas ou mais alternativas, é capaz de produzir cópias de si mesmo e continuar a computação independentemente para cada alternativa
- Um algoritmo não-determinista é capaz de escolher uma dentre as várias alternativas possíveis a cada passo (o algoritmo é capaz de adivinhar a alternativa que leva a solução)
- Utilizam
 - a função `escolhe(C)`: escolhe um dos elementos de C de forma arbitrária.
 - **SUCESSO**: sinaliza uma computação com sucesso
 - **INSUCESSO**: sinaliza uma computação sem sucesso
- Sempre que existir um conjunto de opções que levam a um término com sucesso então, este conjunto é sempre escolhido
- A complexidade da função `escolhe` é $\Theta(1)$

Exemplo: Pesquisa

- Pesquisar o elemento x em um conjunto de elementos $A[1..n]$, $n \geq 1$

```
j ← escolhe(1..n)
se A[j] = x então
    SUCESSO
senão
    INSUCESSO
fim se
```

- Complexidade: $\Theta(1)$
- Para um algoritmo determinista a complexidade é $\Theta(n)$

Exemplo: Ordenação

- Ordenar um conjunto A contendo n inteiros $n \geq 1$

```
NDOrdenacao (A,n)
inicio
  para i←1 até n faça B[i]=0;
  para i←1 até n faça
    inicio
      j ← escolhe(1..n);
      se B[j] = 0 então
        B[j] = A[i];
      senão
        INSUCESSO
      fim se
    fim para
  SUCESSO
fim
```

- B contém o conjunto ordenado
- A posição correta em B de cada inteiro de A é obtida de forma não-determinista
- Complexidade do algoritmo não-determinista: $\Theta(n)$
- Complexidade do algoritmo determinista: $\Theta(n \log n)$

5. Algoritmos Deterministas X Algoritmos Não-deterministas

- Classe P (*Polynomial-time Algorithms*)
 - Conjunto de todos os problemas que podem ser resolvidos por algoritmos deterministas em tempo polinomial
- Classe NP (*Nondeterministic Polynomial Time Algorithms*)
 - Conjunto de todos os problemas que podem ser resolvidos por algoritmos não-deterministas em tempo polinomial

Como Mostrar que um Determinado Problema está em NP?

- Basta apresentar um algoritmo não-determinista que execute em tempo polinomial para resolver o problema

Ou

- Basta encontrar um algoritmo determinista polinomial para verificar que uma dada solução é válida

Exemplo: Problema de decisão do Caixeiro Viajante está em NP

Algoritmo não-determinista em tempo polinomial

```
NDPCV(G,n,k)
  Soma ← 0
  para i ← 1 até n faça
    A[i] ← escolhe(G,n)
  fim para
  A[n+1] ← A[1]
  para i ← 1 até n faça
    Soma ← Soma + distancia entre A[i] e A[i+1]
  fim para
  se Soma ≤ k então
    SUCESSO
  senão
    INSUCESSO
  fim se
```

- Complexidade do algoritmo não-determinista: $\Theta(n)$
- Complexidade do algoritmo determinista: $O(n^2 2^n)$

Algoritmo determinista polinomial para verificar a solução

```
DPCVV(G,S,n,k)
  Soma ← 0
  para i ← 1 até n faça
    Soma ← Soma + distancia entre S[i] e S[i+1]
  se Soma ≤ k então
    escreva "SIM"
  senão
    escreva "NAO"
```

- Complexidade: $\Theta(n)$

6. Relação entre P e NP

$P = NP$ ou $P \neq NP$?

- Como algoritmos deterministas são apenas um caso especial de algoritmos não-deterministas, podemos concluir que

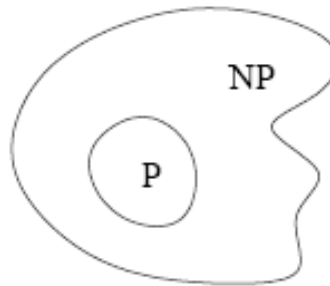
$$P \subseteq NP$$

- O que não sabemos é se

$$P = NP \text{ ou } P \neq NP$$

- Será que existem algoritmos polinomiais deterministas para todos os problemas em NP?
- Por outro lado, a prova de que $P \neq NP$ parece exigir técnicas ainda desconhecidas

Descrição tentativa de NP



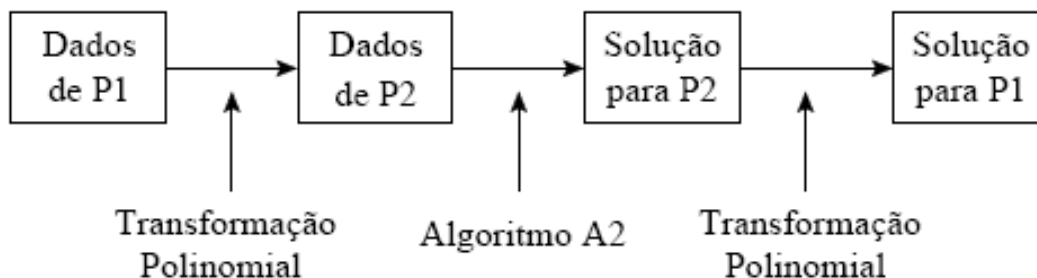
- P está contida em NP
- Acredita-se que NP seja muito maior que P

Consequências

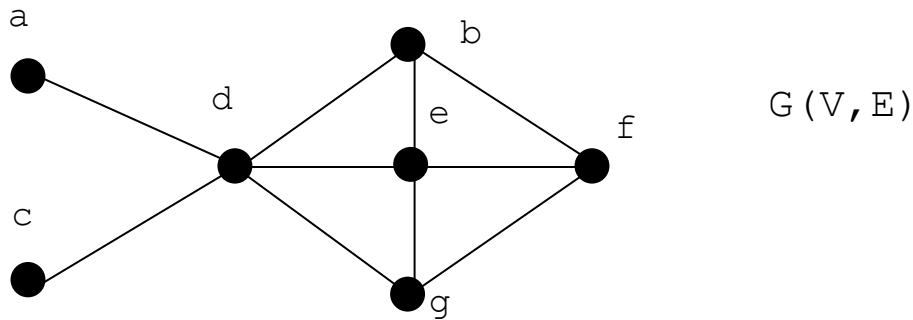
- Existem muitos problemas práticos em NP que podem ou não pertencer a P (não conhecemos nenhum algoritmo eficiente que execute em uma máquina determinista)
- Se conseguirmos provar que um problema não pertence a P, então não precisamos procurar por uma solução eficiente para ele
- Como não existe tal prova sempre há esperança de que alguém descubra um algoritmo eficiente
- Quase ninguém acredita que $NP = P$
- Existe um esforço considerável para provar o contrário: MAS O PROBLEMA CONTINUA EM ABERTO!

7. Redução Polinomial

- Sejam Π_1 e Π_2 dois problemas “sim/não”.
- Suponha que exista um algoritmo A2 para resolver Π_2 . Se for possível transformar Π_1 em Π_2 e sendo conhecido um processo de transformar a solução de Π_2 numa solução de Π_1 então, o algoritmo A2 pode ser utilizado para resolver Π_1
- Se essas duas transformações puderem ser realizadas em tempo polinomial então, Π_1 é polinomialmente redutível a Π_2 ($\Pi_1 \propto \Pi_2$)



Exemplo de Transformação Polinomial



- Conjunto independente de vértices
 - $V' \subseteq V$ tal que todo par de vértices de V' é não adjacente, ou seja, se $v, w \in V' \Rightarrow (v, w) \notin E$
 - a, c, b, g é um exemplo de um conjunto independente de cardinalidade 4
- Clique
 - $V' \subseteq V$ tal que todo par de vértices de V' é adjacente, V' é um subgrafo completo, ou seja, se $v, w \in V' \Rightarrow (v, w) \in E$
 - d, b, e é um exemplo de um clique de cardinalidade 3
- Instância I do Clique
 - **Dados:** Grafo $G(V, E)$ e um inteiro $K > 0$
 - **Decisão:** G possui um clique de tamanho $\geq k$?
- Instância f(I) do Conjunto Independente
 - Considere o grafo complementar G' de G e o mesmo inteiro K, f é uma transformação polinomial porque:
 - G' pode ser obtido a partir de G em tempo polinomial
 - G possui clique de tamanho $\geq k$ se e somente se G' possui conjunto independente de vértices de tamanho $\geq k$

Clique \propto Conjunto Independente

Se existir um algoritmo que resolva o conjunto independente em tempo polinomial, este algoritmo pode ser utilizado para resolver o clique também em tempo polinomial

8. Satisfabilidade

- Definir se uma expressão booleana E contendo produto de adições de variáveis booleanas é satisfatível

– Exemplo: $(x_1 + x_2) * (x_1 + \overline{x_3} + x_2) * (x_3)$

onde x_i representa variáveis lógicas

+ representa OR

* representa AND

- representa NOT

- Problema: Existe uma atribuição de valores lógicos (V ou F) às variáveis que torne a expressão verdadeira (“satisfaça”)?

$x_1=F, x_2=V, x_3=V$ Satisfaz!

Exemplo: $(x_1) * (\overline{x_1})$ não é satisfatível

```

NDAval(E,n)
inicio
  para i<-1 até n faça
    x[i] <- escolhe(true,false)
    se E(x[1],x[2],...,x[n]) = true então
      SUCESSO
    senão
      INSUCESSO
  fim se
fim

```

- O algoritmo obtém uma das 2^n atribuições possíveis de forma não-determinista em $O(n)$

SAT \in NP

Teorema de Cook

- S. Cook (1971) provou que todo problema de NP se reduz à satisfabilidade, enunciando o seguinte teorema:

Teorema de Cook: Satisfabilidade está em P se, e somente se, $P = NP$

- Isto é: Se existe um algoritmo polinomial determinista para a satisfabilidade então, todos os problemas em NP poderiam ser resolvidos em tempo polinomial (estariam também em P, implicando que $P=NP$)

9. Conjunto NP-completo

- Um problema de decisão Π é denominado NP-Completo se
 1. $\Pi \in \text{NP}$
 2. Todo problema de decisão $\Pi' \in \text{NP}$ satisfaz $\Pi' \leq \Pi$
- Apenas problemas de decisão (sim/não) podem ser NP-Completo

Como Provar que um Problema é NP-Completo

1. Mostre que o problema está em NP
 2. Mostre que um problema NP-Completo conhecido pode ser polinomialmente transformado para ele
- Porque:
 - Cook apresentou uma prova direta de que SAT é NP-Completo
 - Transitividade da redução polinomial

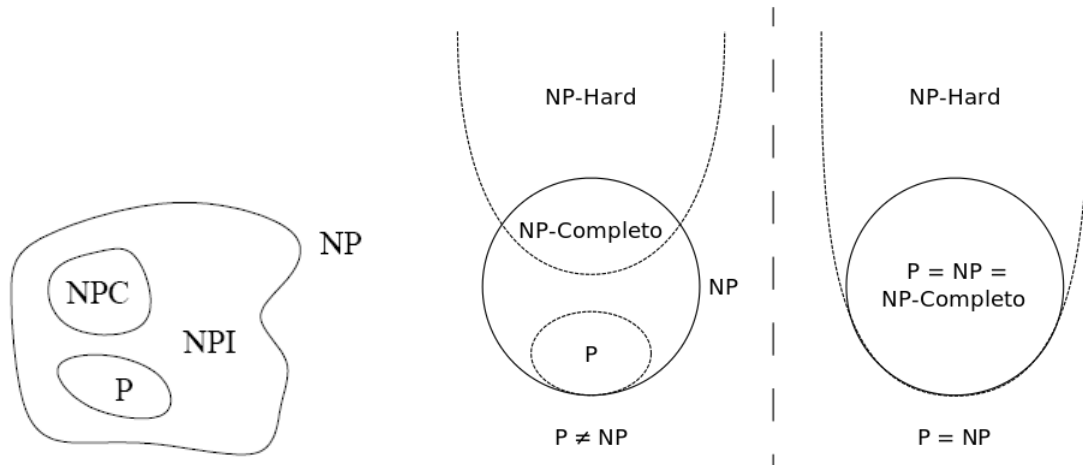
$$\text{SAT} \leq \Pi_1 \text{ e } \Pi_1 \leq \Pi_2 \Rightarrow \text{SAT} \leq \Pi_2$$

10. Conjunto NP-difícil

- Contém os problemas NP-completos, mas não só eles.
- Cuidado: um problema de otimização para o qual não existe algoritmo de complexidade polinomial é NP-difícil mas não pertence a NP!
- Entretanto, todo problema dessa natureza possui um correspondente de decisão em NP-completo.

Descrição tentativa de NP

Assumindo que $P \neq NP$



- Se alguém encontrar um algoritmo polinomial que resolva algum problema NP-Completo então, todos os problemas em NP também terão solução polinomial, ou seja, **P será igual a NP.**
- Se alguém provar que um determinado problema em NP não tem solução polinomial então, todos os problemas em NP-Completo também não terão solução polinomial, ou seja, **P será diferente de NP.**

Qual é a Contribuição Prática da Teoria de NP-Completo?

- Fornece um mecanismo que permite descobrir se um novo problema tem solução eficiente (“fácil”) ou não.
- Se encontrarmos um algoritmo polinomial para o problema, então ele será “fácil”. Senão, uma prova de que o problema é NP-Completo nos diz que possivelmente ele não terá solução eficiente.

Resumo

- Redução polinomial: poder transformar um problema A em outro B e transformar a solução de B em solução de A em tempo polinomial (Não é simétrica!).
- Classe P: Problemas para os quais existem soluções deterministas polinomiais. Como são um subconjunto da classe NP (uma vez que algoritmos determinísticos são subconjunto dos não-determinísticos), esses problemas são sempre associados a um problema correspondente de decisão. Exemplo: O problema de ordenação está associado ao problema de se decidir se uma determinada sequência está ordenada.
- Classe NP: Problemas de decisão que incluem a classe P, a classe NPI e a classe NP-completo. Problemas que não estão na classe P, mas pertencem à NP não têm solução polinomial conhecida. Todos eles devem poder ser resolvidos por um algoritmo não-determinista polinomial ou poder ter sua solução verificada por um algoritmo determinista polinomial.
- Classe NP-completo: Subconjunto de problemas de NP que são também NP-difíceis. Assim, todos os problemas em NP-completo devem ser redutíveis entre si, uma vez que SAT pertence ao conjunto. Contém os problemas de decisão sem solução polinomial determinística conhecida. Por estarem em NP, devem poder ser resolvidos por um algoritmo não-determinista polinomial ou poder ter sua solução verificada por um algoritmo determinista polinomial.

- Classe NP-difícil: Conjunto que contém, além de outros, os problemas que não são resolvidos pela transformação polinomial a um outro problema que está em NP-completo. Contém problemas de otimização sem solução polinomial determinística conhecida e seus correspondentes problemas de decisão, mas também outros problemas difíceis, como os indecidíveis (estes ficam fora de NP). Os problemas de decisão correspondentes (NP-completos) ficam na interseção entre NP-difícil e NP. Os que não são de decisão ficam fora de NP, para os quais não se conhece solução polinomial determinística nem não-determinística.
- Classe NPI: Problemas de decisão associados aos problemas que não são de otimização, mas para os quais também não se conhece solução polinomial determinística. Exemplo: fatoração de grandes números e isomorfismo de grafos não são problemas de otimização. Também não são de decisão mas possuem um correspondente de decisão e podem ser verificados por algoritmos determinísticos polinomiais, logo estão em NP.
- Satisfabilidade (SAT) é um problema de decisão pertencente a NP (provavelmente apenas em NP-completo)

Reduções:

- Todo problema em NP se reduz à SAT (Cook)
- NP-completo se reduz a NP-difícil, por definição (logo todos os NP-completos se reduzem entre si). No entanto, NP-difícil não se reduz a NP-completo.

- SAT se reduz a todo problema NP-difícil (logo também a todo problema NP-completo)
- O problema da parada é NP-difícil pois é indecidível (logo não pode pertencer a NP) mas SAT, que pertence a NP-completo se reduz a ele.
- Não se conhece redução polinomial de NP-completo para NPI, nem para P.
- Se existir uma redução de NP-completo para P, teremos que $P=NP$

11. Algoritmos Aproximados

- Problemas de otimização que estão na classe NP não possuem soluções gerais eficientes.
- Algoritmos aproximados podem ser usados para se encontrarem soluções próximas do ótimo, em tempo polinomial.
- Estes algoritmos possuem diferentes níveis de sofisticação, entretanto, a maioria deles constitui-se de algoritmos gulosos baseados em alguma heurística.
- Algoritmos aproximados:
 - Algoritmos usados normalmente para resolver problemas para os quais não se conhece uma solução polinomial
 - Devem executar em tempo polinomial dentro de limites “prováveis” de qualidade absoluta ou assintótica

- Heurísticas:
 - Heurísticas são regras de bom senso, baseadas na experiência e na forma como resolvemos problemas naturalmente.
 - Algoritmos que têm como objetivo fornecer soluções sem um limite formal de qualidade, em geral avaliado empiricamente em termos de complexidade (média) e qualidade de soluções
 - É projetada para obter ganho computacional ou simplicidade conceitual, possivelmente ao custo de precisão

A) Exemplo: Problema do Caixeiro Viajante: visitar n cidades, passando por cada uma única vez, saindo e chegando de uma mesma cidade. Todas as cidades estão interligadas e deseja-se a solução de menor custo.

- Heurística do vértice mais próximo primeiro
- Heurística do vértice mais distante primeiro

B) Exemplo: Problema do Empacotamento (Bin Packing): Utilizar o menor número possível de caixas para empacotar n objetos de tamanhos variados.

- Heurística do maior primeiro
- Heurística do melhor primeiro

C) Exemplo: Mochila

- Colocar os objetos com maior valor/tamanho primeiro

D) Exemplo: Coloração

- Atribuir cor ao vértice com maior grau primeiro