Problemas
Classes de problemas

Teoria da Computação

Problemas vs Computação

- O que os computadores podem fazer?
  - Quais problemas eles podem resolver eficientemente?
- O que os computadores não podem fazer?
  - Existem problemas que não podem ser resolvidos por computadores, não importando quão poderosos eles possam vir a ser?
- Importa como eles são implementados?

Origens da Computação

[1900] *Entscheidungsproblem*: 23 problemas-desafios propostos por David Hilbert, visando investigar se:

- Existe um procedimento mecânico (algoritmo) capaz de decidir a veracidade ou falsidade de qualquer conjectura matemática?
- Existe um sistema formal capaz de derivar uma prova ou uma refutação de qualquer conjectura matemática?

Alan Turing

- Em 1936 Turing, em Cambridge, UK, desenvolveu um modelo do que ele achava ser o processo que um matemático realiza quando tenta provar alguma conjectura matemática.

- A partir desse modelo (máquina de Turing) ele demonstrou que o *Entscheidungsproblem* era equivalente ao *Halting Problem* e, portanto, a resposta à questão de Hilbert era NÃO.

O que estuda a Teo.Comp.?

- O que é passível de solução por algoritmos?
  - *Decidibilidade e computabilidade*
- O que não é passível de solução por algoritmos?
  - *Indecidibilidade*
- O que pode ser efetivamente solúvel por algoritmos?
  - *Complexidade*

Taxa de crescimento

- Sejam *f* e *g* funções de N em N. Então, dizemos que $f(n) = O(g(n))$ se existem números c e n_o tais que $|f(n)| \leq c.|g(n)|$, para todo $n \geq n_o$.

- Se $f(n) = O(g(n))$ e $g(n) = O(f(n))$
  - então dizemos que *f* e *g* têm a mesma taxa de crescimento.
- Se $f(n) = O(g(n))$ mas $g(n) \neq O(f(n))$
  - então dizemos que g cresce mais rápido que f.

Teorema do limite

Sejam *f* e *g* funções de N em N. Assim:

- Se $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \beta$ onde $\beta \in R^{+}$

  então, ***f(n)* = O(*g(n)*)** e ***g(n)* = O(*f(n)*)**

- Se $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$

  então, ***f(n)* ≠ O(*g(n)*)** mas ***g(n)* = O(*f(n)*)**

Taxa de crescimento iguais?

Exemplo: $n^2=O(3n^2-6n+5)$ e $3n^2-6n+5=O(n^2)$

Assim, temos: $$\frac{f(n)}{g(n)} = \frac{n^2}{3n^2-6n+5}$$

Simplificando a equação ...

$$\frac{n^2 \div n^2}{3n^2-6n+5 \div n^2} = \frac{1}{3-\frac{6}{n}+\frac{5}{n^2}}$$

Taxa de crescimento iguais? (cont)

Calculando o limite, quando n tende ao ∞:

$$\lim_{n\to\infty} \frac{1}{3 - \frac{6}{n} + \frac{5}{n^2}} = \frac{1}{3}$$

Portanto, existe um n_o tal que para todo $n \geq n_o$ seja verdadeira a inequação:

$$\frac{n^2}{3n^2 - 6n + 5} \leq 1$$

Polinômios vs Polinômios

- Seja $p(n)$ um polinômio de grau d. Então:

$$p(n) = O(n^d)$$

- Supondo $p(n) = a_0 + a_1 n + \ldots + a_d n^d$, então se dividirmos esse polinômio por n^d, teremos:

$$p(n)/n^d = a_0/n^d + a_1/n^{d-1} + \ldots + a_d$$

  que tende para a_d quando n tende para ∞.
- Assim, existe n_0 tal que p/ todo $n \geq n_0$ temos $p(n)/n^d \leq a_d+1$, o que implica $p(n) \leq (a_d+1) \cdot n^d$ e portanto, $p(n) = O(n^d)$

Polinômios vs Exponenciais

- Seja $p(n)$ um polinômio de grau d. Então:

$$p(n) = O(n^d)$$

- Em outras palavras, a taxa de crescimento de um polinômio pode ser identificada segundo o seu grau.

$$n^d = O(n^{d+1}) \text{ mas } n^{d+1} \neq O(n^d)$$

- Seja p um polinômio e $k > 1$. Então:

$$p(n) = O(k^n) \text{ mas } k^n \neq O(p(n))$$

ou seja, a função exponencial k^n sempre cresce mais rápido que qualquer polinômio.

Classe P, NP e NP-Completo

Classes de problemas

Categorias dos problemas

Os problemas de decisão que têm solução algorítmica são classificados em:

- **tratáveis**: os que são teórica e realisticamente computáveis; e
- **intratáveis**: os que são teórica, mas não realisticamente computáveis.

Os demais problemas, são ditos **indecidíveis**.

Problemas Solucionáveis 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 fechamento dos algoritmos polinomiais (soma, multiplicação e composição)

Problemas Verificáveis em tempo polinomial

- O objetivo é conferir (verificar) se uma instância pertence a uma dada linguagem utilizando um certificado (i.e. uma possível solução); não implica em decidir se uma instância pertence a essa linguagem

Problemas Abstratos

- Problema abstrato Q:
  - Relação binária entre conjunto de instâncias I e conjunto S de soluções
- Problemas de decisão:
  - Problemas cuja resposta/solução é: sim(1) ou não(0), $Q(i) \in \{0,1\}$
  - Problemas de otimização:
    - Reformulados como problemas de decisão
    - se problema de otimização é tratável, então reformulação como problema de decisão também é tratável

Utilização de Linguagens Formais

- Definições:
  - Alfabeto Σ: conjunto finito de símbolos
  - Linguagem L: conjunto de strings de símbolos de Σ
  - Linguagem Σ*: todas as strings de Σ
    - String vazia: ε
    - Linguagem vazia: ∅
  - Operações sobre linguagens:
    - união, intersecção, complemento, concatenação, fecho
- Para um problema de decisão Q, conjunto de instâncias é Σ*, com Σ= {0,1}
  - Q interpretado como linguagem L definida em Σ
    - $L = \{ x \in \Sigma^* : Q(x) = 1 \}$

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 aceitada por A: $L=\{ x \in \{0,1\}^* : A(x)=1 \}$
L é decidida por A se qualquer string $x \in \{0,1\}^*$ é aceita ou rejeitada

L é aceitada/decidida em tempo polinomial se A tem tempo de execução em $O(n^k)$, com $n = |x|$

Polinomialmente Decidível

- Uma linguagem L é dita ser decidível em tempo polinomial se existem uma MT determinística M de k-fitas que decide L e um polinômio p tal que o nº de passos da computação de M para a entrada w é $\leq p(|w|)$.
- Uma função f é dita ser computável em tempo polinomial se existem uma MT determinística M de k-fitas que computa f e um polinômio p tal que o nº de passos da computação de M para entrada x é $\leq p(|x|)$.

Classe *P*

Definição

Classe P: é uma classe de problemas de decisão que podem ser resolvidos em tempo polinomial por algoritmos determinísticos.

Essa categoria de problemas também é chamada de Polinomial

Outras definições para a Classe *P*

- P = { L $\subseteq$ {0,1}* : existe um algoritmo A que decide L em tempo polinomial }

- P = { L $\subseteq$ {0,1}* : L é aceitada por um algoritmo de tempo polinomial }

  - Conjunto das linguagens decididas em tempo polinomial é subconjunto das linguagens aceitadas em tempo polinomial
  - Basta provar que se L é aceitada 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 $T=c.n^k$
  - Utilizar A' que executa A e observa resultado após T=cnk
    - Se A aceita, A' aceita; se A não aceita (ainda), A' rejeita

Classe *P*

- Todo problema de decisão pode ser resolvido em tempo polinomial?

  - Na realidade, alguns problemas de decisão não podem ser resolvidos por um algoritmo. Estes problemas são chamados indecidíveis.

  - Exemplo: *Halting Problem*
    - Dado um programa de computador e uma entrada para ele, determinar se o programa irá parar (halt) naquela entrada ou se continuará trabalhando sobre ela indefinidamente.

Outros problemas ...

- Existe um grande número de problemas importantes para os quais:
  - ainda não foi descoberto um algoritmo de tempo polinomial; e
  - nem foi provada a impossibilidade de existir tal algoritmo.

Classe *NP*

Definição

Classe NP: problemas de decisão que são "verificáveis" em tempo polinomial, isto é, se tivéssemos uma "solução" (de algum modo), poderíamos verificar se a solução é correta em tempo polinomial.

Problemas de decisão desta categoria podem ser resolvidos por algoritmos polinomiais não-determinísticos

Classe *NP*

- São exemplos de problemas desta classe:
  - Circuito Hamiltoniano
  - Caixeiro Viajante
  - Problema da Mochila
  - Problema da Partição
  - Problema do Empacotamento
- Esses problemas têm em comum um crescimento exponencial em função do tamanho da entrada

Algoritmos Não Determinístico

- Um algoritmos não determinístico é um procedimento que tem como entrada uma instância I de um problema de decisão e é composto por dois estágios:
  - Estágio não determinístico (suposição):
    - produz aleatoriamente um conjunto S de valores que podem ser considerados uma possível solução para a instância I.
  - Estágio determinístico (verificação):
    - tendo I e S como entrada, o algoritmo responde *sim* se S for uma solução para a instância I.

Algoritmos Não Determinístico Polinomial

- Dizemos que um algoritmos não determinístico resolve um problema de decisão se, para cada instância *sim* do problema, ele retorna *sim* em alguma execução.
- Um algoritmos não determinístico é dito ser polinomial não determinístico se a eficiência de seu estágio de verificação for polinomial.

P vs NP

- $P \subseteq NP$
  A prova é imediata visto toda MT determinística também é não-deterministica.
- $NP \subseteq P$
  A grande questão!?
  Ninguém até hoje conseguiu provar que algum problema de *NP* não pertence a *P*.
  Por outro lado, muito esforço tem sido feito para encontrar um algoritmo polinomial para alguns problemas *NP* sem sucesso.

Classe *NP-Completo*

- Certos problemas em *NP* tem uma característica interessante:
  - se um algoritmo polinomial existe para qualquer desses problemas, todos os problemas em *NP* seriam computáveis em tempo polinomial, ou seja, *P* seria igual a *NP*.
- Esses problemas são chamados de NP-Completos.

Classe *NP - Completo*

Definição

Classe NP-Completo: um problema NPC é um problema que está em NP e é tão difícil quanto qualquer outro problema NP. Por definição, qualquer problema NP pode ser reduzido a ele em tempo polinomial.

Se qualquer problema NP - Completo pode ser resolvido em tempo polinomial, então, todo problema NP - Completo tem um algoritmo de tempo polinomial

Resumo

- Os problemas de decisão são aqueles cuja resposta é sim ou não.
- O *halting problem* é um problema de decisão não decidível, i.e., ele não pode ser resolvido por um nenhum algoritmo.
- P é a classe de problemas de decisão que podem ser resolvidos em tempo polinomial.

Resumo

- NP é a classe de todo os problemas de decisão cujas soluções geradas aleatoriamente podem ser verificadas em tempo polinomial.
- Muitos problemas em NP são também NP - Completos, todos os outros problemas em NP são redutíveis a tal problema em tempo polinomial.
- A primeira prova de um problema NP - Completo foi publicada em 1971 por Cook, para o problema da satisfabilidade.

Resumo

- Não é conhecido se P = NP ou P é somente um subconjunto apropriado de NP. Esta é a mais importante questão aberta na teoria da ciência da computação.
- A descoberta de um algoritmo de tempo polinomial para qualquer um dos milhares de problemas NP - Completos implicaria em que P = NP