**INF 02779 Análise de Algoritmos - 2002/2** 8/01/03

# Aula 11, 12 e 13: Introdução à Complexidade Computacional

Instrutor: Berilhes Borges Garcia Escriba: William Flávio Rigo, Alberico Pereira de Souza Junior e Eduardo Glicério

**DRAFT**

## 1 Introdução

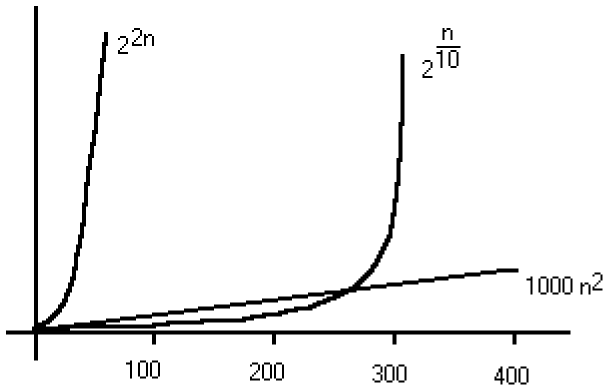
O computador do ano que vem será 2 vezes mais Rápido. Se eu posso resolver, hoje, um problema de tamanho $n_0$, qual é o tamanho do problema que eu posso resolver no próximo ano com o mesmo tempo? (Ver tabela na próxima página)
Nós diremos que qualquer algorítmo cujo tempo de execução é:
- polinimial é bom
- maior que polinomial é ruim.
Note, é claro, que existe execções:
- $n^{100}$ é maior que $(1,001)^n$ para muitos valores práticos de $n$, mas usualmente tais tempos de execução não ocorrem.
- Existem algoritmos que possuem tempo de execução do tipo $O(2^{n/22})$ e estes podem ser úteis para entradas pequenas e médias, contudo este tipo de algoritmo não é muito comum.

## 2 Alguma História

1. 1930 - O que é (não é) computável.
2. 1960 - 1970 -
   (a) O que é (não é) factivelmente computável.
   (b) Meta - Uma teoria do tempo requerido pelos algoritmos que seja independente da tecnologia.
   (c) Suposições de modelagem chaves:
      i. Comportamento Assintótico (Big O, Big theta , . . . ), pior caso é relevante.
      ii. Tempo exponencial e tempo polinomial são qualitativamente diferentes.

## 3 Problemas de Decisão e Classe P

Complexidade computacional é usualmente analizada utilizando-se problemas de decisão , ou seja problemas com resposta 1(sim) ou 0 (não).




| *Complexidade* | *Aumento* | *Exemplo* | $T = 10^{12}$ |
|---|---|---|---|
| $O(n)$ | $2n_0$ | $10^{12}$ | $2X10^{12}$ |
| $O(n^2)$ | $\sqrt[2]{2}n_0$ | $10^6$ | $1,4X10^6$ |
| $O(n^3)$ | $\sqrt[3]{2}n_0$ | $10^4$ | $1,25X10^4$ |
| $2^{n/10}$ | $n_0 + 10$ | 400 | 410 |
| $2^n$ | $n_0 + 1$ | 40 | 41 |

Por que?
Porque são muitos mais simples de serem manipuladas.
Decidir se existe um caminho de s para t no grafo G , é certamente não mais difícil que encontrar um caminho de s para t me G, dessa forma um limite inferior sob a complexidade de um problema de decisão é também um limite inferior sob o problema de encontrar um caminho , por exemplo.
Outro ponto menos importante é que se você tem um bom algoritmo de decisão você pode frequentemente utilizá-lo para obter um algoritmo de construção da solução.
Complexidade computacional tem por objetivo classificar os problemas de acordo com o montante de recursos computacionais utilizado que resolve estes problemas.
Relembre que o tempo de execução no pior caso de um determinado algoritmo que resolve estes problemas.
Relembre que o tempo de execução no pior caso de um determinado algoritmo é o número máximo de passos que o algoritmo demanda sob qualquer entrada de tamano n.

**Definição 1.** *(Tempo $f(n)$) Tempo f(n) é o conjunto de todos os problemas de decisão resolvidos por algoritmos tendo tempo de execução no pior caso de $O(f(n))$.*

**Definição 2.** *(Tempo Polinomial )P(tempo polinomial) é o conjunto de todos os problemas de de-*

*cisão resolvíveis por algoritmos cujo tempo de execução no pior caso é limitado por algum polinômio no tamanho da instância de entrada.*

$$P = \bigcup_{k \geq 0} Time(n^k) \tag{1}$$

**Notação 1.** *1. Problema. O caso geral de uma tarefa computacional . Exemplo: Dado um grafo G e nós s e t em G, G contém um caminho de s para t?*

*2. Instância do Problema. Uma dada entrada específica para um problema.*

*3. Problemas de Decisão - são problemas cujas as respostas são Sim / Não.*

Além de P?
Existem muitos problemas naturais e práticos para os quais nós não conhecemos nehum algoritmo de tempo polinomial.
*Example:* Versão de decisão do TSP.
Dado um grafo ponderado G e um inteiro $k$, existe algum caminho que visite todos os vértices em G tendo peso total de no mínimo $k$? ⊠

# 4 Satisfabilidade

Variáveis booleanas $x_1, \ldots, x_n$ assumem o valor 0 ou 1. Literais são variáveis booleanas $x_i$ ou sua negação $\neg x_i$ para i = 1, ... , n. Uma cláusula é uma disjunção de um ou mais literais, por exemplo $(x_1 \vee x_2 \vee x_7 \vee \neg x_{12})$. Uma fórmula CNF é uma conjunção de uma ou mais cláusulas. Por exemplo, a fórmula $\phi = (x_1 \vee x_3 \vee \neg x_7 \vee x_{12}) \wedge (x_2 \vee \neg x_4 \vee x_7 \vee x_5)$, é uma fórmula CNF. Se existe alguma atribuição de zeros e uns às variáveis que tornam uma fórmula verdadeira então nós dizemos que a fórmula é satisfatível.

*Example:*
a) Fórmula satisfatível: $\phi = (x_1 \vee x_3 \vee \neg x_7 \vee x_{12}) \wedge (x_2 \vee \neg x_4 \vee x_7 \vee x_5)$.
b) Fórmula não satisfatível: $\psi = x_1 \wedge (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_3) \wedge \neg x_3$.

⊠

**Definição 3.** *(Satisfabilidade) Dada uma fórmula CNF $\phi$, esta é satisfatível?*

Muitos dos problemas acima têm sido estudados por décadas. Todos têm aplicações reais e práticas. Contudo nenhum deles tem até hoje um algoritmo de tempo polinomial, algoritmos exponenciais são o que de "melhor" existem para eles [aspas do autor]. Todos estes problemas possuem a propriedade comum que um certificado curto (ou prova) existe, de modo que você pode eficientemente verificar (em tempo polinomial) se uma resposta Sim é correta. Note que este pedaço especial de informação pode ser difícil de encontrar.

1. Versão de decisão do TSP: o próprio tour é o certificado.

2. Conjunto Independente: o conjunto U é o certificado.

3. Clique: o conjunto U é o certificado.

4. Satisfabilidade: uma atribuição que faz uma fórmula $\phi$ verdade é o certificado.

**Definição 4.** *(Classe NP)*

*A classe NP consiste de todos os problemas onde você pode verificar as respostas Sim eficientemente (em tempo polinomial) dado um certificado curto (de tamanho polinomial).*