Universidade Federal de São Carlos
Departamento de Engenharia de Produção

Otimização Linear Contínua e Discreta (Tópicos Avançados em PCSP)

PPGEP, UFSCar - Semestre 01/2022
Prof. Dr. Pedro Munari (munari@dep.ufscar.br)

Tópico 8.1: Complexidade de Algoritmos em Otimização

Objetivos deste tópico

- Estudar os conceitos de complexidade computacional, com enfoque em problemas de otimização;
- Estudar a análise de pior caso baseada em tempo computacional;
- Conhecer alguns exemplos de algoritmos de tempo polinomial e não-polinomial.

Complexidade computacional

Estudo e classificação de algoritmos e problemas;

Complexidade computacional

Estudo e classificação de algoritmos e problemas;

- Ajuda a compreender quão difícil pode ser resolver um dado problema, tipicamente em relação ao tempo computacional;

Complexidade computacional

Estudo e classificação de algoritmos e problemas;

- Ajuda a compreender quão difícil pode ser resolver um dado problema, tipicamente em relação ao tempo computacional;

Dado um modelo que descreva o problema e os dados de suas instâncias, estamos preocupados com o tempo computacional para resolvê-lo;

Complexidade computacional

Estudo e classificação de algoritmos e problemas;

- Ajuda a compreender quão difícil pode ser resolver um dado problema, tipicamente em relação ao tempo computacional;

Dado um modelo que descreva o problema e os dados de suas instâncias, estamos preocupados com o tempo computacional para resolvê-lo;

- Estamos interessados em determinar quando um problema pode ser resolvido em tempo $O(l^k)$, sendo k uma constante fixa e l uma medida do tamanho da entrada necessária para descrever os dados do problema.

Complexidade computacional

- Intuitivamente, relacionamos o tempo computacional ao tamanho do problema, baseando-se no número de variáveis e número de restrições;

Complexidade computacional

- Intuitivamente, relacionamos o tempo computacional ao tamanho do problema, baseando-se no número de variáveis e número de restrições;
- Porém, olharmos apenas para isso pode levar a conclusões equivocadas!

Complexidade computacional

- Intuitivamente, relacionamos o tempo computacional ao tamanho do problema, baseando-se no número de variáveis e número de restrições;
- Porém, olharmos apenas para isso pode levar a conclusões equivocadas!
- Além disso, o ideal é termos uma medida de tempo que seja independente das característica de um computador específico;

Complexidade computacional

- Intuitivamente, relacionamos o tempo computacional ao tamanho do problema, baseando-se no número de variáveis e número de restrições;
- Porém, olharmos apenas para isso pode levar a conclusões equivocadas!
- Além disso, o ideal é termos uma medida de tempo que seja independente das característica de um computador específico;
- Vamos considerar o número de operações elementares (p.ex. adições, multiplicações, comparações, etc.) e assumir que cada operação elementar seja executada em uma unidade de tempo.

Complexidade computacional

▷ Definições e notação

- Nosso enfoque será na complexidade de problemas de programação inteira-mista, que podem ser descritos na forma geral:

Complexidade computacional

▷ Definições e notação

- Nosso enfoque será na complexidade de problemas de programação inteira-mista, que podem ser descritos na forma geral:

$$\max\{cx + hy \mid Ax + Gy \leq b,\ x \in \mathbb{Z}^n_+,\ y \in \mathbb{R}^p_+\},$$

sendo $m, n, p \in \mathbb{Z}^n_+$, com $m > 0$ e $p + n \geq 1$; c, h, A, G, b matrizes com coeficientes inteiros e dimensões $1 \times n$, $1 \times p$, $m \times n$, $m \times p$ e $m \times 1$, respectivamente

Complexidade computacional

▷ Definições e notação

- Nosso enfoque será na complexidade de problemas de programação inteira-mista, que podem ser descritos na forma geral:

$$\max\{cx + hy \mid Ax + Gy \leq b,\ x \in \mathbb{Z}^n_+,\ y \in \mathbb{R}^p_+\},$$

sendo $m, n, p \in \mathbb{Z}^n_+$, com $m > 0$ e $p + n \geq 1$; c, h, A, G, b matrizes com coeficientes inteiros e dimensões $1 \times n$, $1 \times p$, $m \times n$, $m \times p$ e $m \times 1$, respectivamente (obs.: assumimos coeficientes inteiros sem perda de generalidade em relação a coeficientes racionais);

Complexidade computacional

▷ Definições e notação

- Nosso enfoque será na complexidade de problemas de programação inteira-mista, que podem ser descritos na forma geral:

$$\max\{cx + hy \mid Ax + Gy \leq b,\ x \in \mathbb{Z}^n_+,\ y \in \mathbb{R}^p_+\},$$

  sendo $m, n, p \in \mathbb{Z}^n_+$, com $m > 0$ e $p + n \geq 1$; c, h, A, G, b matrizes com coeficientes inteiros e dimensões $1 \times n$, $1 \times p$, $m \times n$, $m \times p$ e $m \times 1$, respectivamente (obs.: assumimos coeficientes inteiros sem perda de generalidade em relação a coeficientes racionais);
- Programação linear e programação inteira-pura são casos particulares.

Complexidade computacional

▷ Definições e notação

- Um problema consiste em um número infinito de instâncias;

Complexidade computacional

▷ Definições e notação

- Um problema consiste em um número infinito de instâncias;
- Uma instância é especificada pela designação de valores numéricos, chamados de **dados**, aos parâmetros do problema;

Complexidade computacional

▷ Definições e notação

- Um problema consiste em um número infinito de instâncias;
- Uma instância é especificada pela designação de valores numéricos, chamados de **dados**, aos parâmetros do problema;
- O tamanho de um problema fica definido como a quantidade de informação necessária para representar a instância;

Complexidade computacional

▷ Definições e notação

- Um problema consiste em um número infinito de instâncias;
- Uma instância é especificada pela designação de valores numéricos, chamados de **dados**, aos parâmetros do problema;
- O tamanho de um problema fica definido como a quantidade de informação necessária para representar a instância;
  - Por exemplo, inteiros m e n, vetores e matrizes b, c, A;

Complexidade computacional

▷ Definições e notação

- Um problema consiste em um número infinito de instâncias;
- Uma instância é especificada pela designação de valores numéricos, chamados de **dados**, aos parâmetros do problema;
- O tamanho de um problema fica definido como a quantidade de informação necessária para representar a instância;
  - Por exemplo, inteiros m e n, vetores e matrizes b, c, A;
  - Em geral, usamos uma representação binária desses dados.

Complexidade computacional

▷ Definições e notação

- Um problema consiste em um número infinito de instâncias;
- Uma instância é especificada pela designação de valores numéricos, chamados de **dados**, aos parâmetros do problema;
- O tamanho de um problema fica definido como a quantidade de informação necessária para representar a instância;
  - Por exemplo, inteiros m e n, vetores e matrizes b, c, A;
  - Em geral, usamos uma representação binária desses dados. Um número inteiro positivo x, tal que $2^k \leq x < 2^{k+1}$, é representado por um vetor $(\delta_0, \delta_1, \ldots, \delta_k)$ tal que:

$$x = \sum_{i=0}^{k} \delta_i 2^i, \ \ \delta_i \in \{0, 1\}.$$

  - Note que $k \leq \log_2 x < k + 1$.

Complexidade computacional

▷ Definições e notação

- Seja X um problema de otimização que consiste em um número infinito de instâncias $d_1, d_2, \ldots$;

Complexidade computacional

▷ Definições e notação

- Seja X um problema de otimização que consiste em um número infinito de instâncias $d_1, d_2, \ldots$;
- Os dados de uma dada instância d_i são descritos por uma cadeia binária de tamanho $l_i = l(d_i)$;

Complexidade computacional

▷ Definições e notação

- Seja X um problema de otimização que consiste em um número infinito de instâncias $d_1, d_2, \ldots$;
- Os dados de uma dada instância d_i são descritos por uma cadeia binária de tamanho $l_i = l(d_i)$;
- Seja A um algoritmo capaz de resolver qualquer instância de X em tempo finito;

Complexidade computacional

▷ Definições e notação

- Seja X um problema de otimização que consiste em um número infinito de instâncias $d_1, d_2, \ldots$;
- Os dados de uma dada instância d_i são descritos por uma cadeia binária de tamanho $l_i = l(d_i)$;
- Seja A um algoritmo capaz de resolver qualquer instância de X em tempo finito;
- O tempo de execução de A é representado pela função $g_A : X \rightarrow \mathbb{R}_+$

Complexidade computacional

▷ Definições e notação

- Seja X um problema de otimização que consiste em um número infinito de instâncias $d_1, d_2, \ldots$;
- Os dados de uma dada instância d_i são descritos por uma cadeia binária de tamanho $l_i = l(d_i)$;
- Seja A um algoritmo capaz de resolver qualquer instância de X em tempo finito;
- O tempo de execução de A é representado pela função $g_A : X \rightarrow \mathbb{R}_+$
- O ideal seria termos uma expressão para g_A que dependesse de l, o tamanho da cadeia binária que descreve os dados;

Complexidade computacional

▷ Definições e notação

- Seja X um problema de otimização que consiste em um número infinito de instâncias $d_1, d_2, \ldots$;
- Os dados de uma dada instância d_i são descritos por uma cadeia binária de tamanho $l_i = l(d_i)$;
- Seja A um algoritmo capaz de resolver qualquer instância de X em tempo finito;
- O tempo de execução de A é representado pela função $g_A : X \to \mathbb{R}_+$
- O ideal seria termos uma expressão para g_A que dependesse de l, o tamanho da cadeia binária que descreve os dados;
- Entretanto, sabemos que duas instâncias de mesmo tamanho, podem não ter o mesmo tempo de execução e, assim, precisamos de uma **medida agregada** para as instâncias de mesmo tamanho;

Complexidade computacional

▷ Definições e notação

- Para isso, vamos usar uma medida de **pior caso**.

Complexidade computacional

▷ Definições e notação

- Para isso, vamos usar uma medida de **pior caso**.
- O tempo de execução associado a todas as instâncias de tamanho k é então definido por:

$$f_A(k) = \max\{g_A(d_i) : l(d_i) = k\}$$

Complexidade computacional

▷ Definições e notação

- Para isso, vamos usar uma medida de **pior caso**.
- O tempo de execução associado a todas as instâncias de tamanho k é então definido por:

$$f_A(k) = \max\{g_A(d_i) : l(d_i) = k\}$$

- Assim, temos a vantagem de ter uma garantia absoluta a respeito do tempo de execução, que é independente de uma distribuição de probabilidades e fácil de analisar;

Complexidade computacional

▷ Definições e notação

- Para isso, vamos usar uma medida de **pior caso**.
- O tempo de execução associado a todas as instâncias de tamanho k é então definido por:

$$f_A(k) = \max\{g_A(d_i) : l(d_i) = k\}$$

- Assim, temos a vantagem de ter uma garantia absoluta a respeito do tempo de execução, que é independente de uma distribuição de probabilidades e fácil de analisar;
- Por outro lado, é altamente conservadora e pode falhar em expressar a realidade, especialmente quando uma grande porcentagem de instâncias de um dado tamanho pode ser resolvida rápida, enquanto apenas algumas poucas levam tempo relativamente grande.

Complexidade computacional

▷ Definições e notação

- Dado que estamos interessados em uma análise de pior caso, vamos caracterizar $f_A(k)$ usando a notação O-grande;

Complexidade computacional

▷ Definições e notação

- Dado que estamos interessados em uma análise de pior caso, vamos caracterizar $f_A(k)$ usando a notação O-grande;
- Assim, dizemos que $f(k)$ é $O(g(k))$ sempre que existir uma constante positiva c e um inteiro positivo k' tais que:

$$f(k) \leq cg(k), \text{ para todo inteiro } k \geq k'.$$

Complexidade computacional

▷ Definições e notação

- Dado que estamos interessados em uma análise de pior caso, vamos caracterizar $f_A(k)$ usando a notação O-grande;
- Assim, dizemos que $f(k)$ é $O(g(k))$ sempre que existir uma constante positiva c e um inteiro positivo k' tais que:

$$f(k) \leq cg(k), \text{ para todo inteiro } k \geq k'.$$

- Por exemplo, o polinômio $k_1 + k_2 + \ldots + k^p$ é $O(k^p)$;

Complexidade computacional

▷ Definições e notação

- Dado que estamos interessados em uma análise de pior caso, vamos caracterizar $f_A(k)$ usando a notação O-grande;
- Assim, dizemos que $f(k)$ é $O(g(k))$ sempre que existir uma constante positiva c e um inteiro positivo k' tais que:

$$f(k) \leq cg(k), \text{ para todo inteiro } k \geq k'.$$

- Por exemplo, o polinômio $k_1 + k_2 + \ldots + k^p$ é $O(k^p)$;
- Atenção para $k \geq k'$ na definição: estamos considerando apenas o comportamento assintótico da função, conforme $k \to \infty$.

Complexidade computacional

▷ Algoritmos de tempo polinomial e problemas na classe $\mathcal{P}$

Complexidade computacional

▷ Algoritmos de tempo polinomial e problemas na classe $\mathcal{P}$

- Um **algoritmo** A é dito ser de **tempo polinomial** para o problema X se $f_A(k)$ é $O(k^p)$ para algum p fixo;

Complexidade computacional

▷ Algoritmos de tempo polinomial e problemas na classe $\mathcal{P}$

- Um **algoritmo** A é dito ser de **tempo polinomial** para o problema X se $f_A(k)$ é $O(k^p)$ para algum p fixo;
- Seja $\mathcal{P}$ a **classe de problemas** que podem ser resolvidos em **tempo polinomial**;

Complexidade computacional

▷ Algoritmos de tempo polinomial e problemas na classe $\mathcal{P}$

- Um **algoritmo** A é dito ser de **tempo polinomial** para o problema X se $f_A(k)$ é $O(k^p)$ para algum p fixo;
- Seja $\mathcal{P}$ a **classe de problemas** que podem ser resolvidos em **tempo polinomial**;
- Assim, um **problema** X está em $\mathcal{P}$ se, e somente se, existe um algoritmo de tempo polinomial para resolver X;

Complexidade computacional

▷ Algoritmos de tempo polinomial e problemas na classe $\mathcal{P}$

- Um **algoritmo** A é dito ser de **tempo polinomial** para o problema X se $f_A(k)$ é $O(k^p)$ para algum p fixo;
- Seja $\mathcal{P}$ a **classe de problemas** que podem ser resolvidos em **tempo polinomial**;
- Assim, um **problema** X está em $\mathcal{P}$ se, e somente se, existe um algoritmo de tempo polinomial para resolver X;
- Um tema de bastante interesse em complexidade computacional, está na diferença (ou não!) entre problemas que se sabe pertencer a $\mathcal{P}$ e aqueles para os quais não se conhece nenhum algoritmo de tempo polinomial.

Complexidade computacional

▷ Algoritmos de tempo exponencial

- Uma função f é dita ser *exponencial* se para constantes $c_1, c_2 > 0$ e $d_1, d_2 > 1$ e um inteiro positivo k', tivermos:

$$c_1 d_1^k \leq f(k) \leq c_2 d_2^k, \text{ para todo inteiro } k \geq k'.$$

Complexidade computacional

▷ Algoritmos de tempo exponencial

- Uma função f é dita ser *exponencial* se para constantes $c_1, c_2 > 0$ e $d_1, d_2 > 1$ e um inteiro positivo k', tivermos:

$$c_1 d_1^k \leq f(k) \leq c_2 d_2^k, \text{ para todo inteiro } k \geq k'.$$

- Um exemplo de algoritmo de tempo exponencial é a enumeração completa de 2^k vetores binários k-dimensionais;

Complexidade computacional

▷ Algoritmos de tempo exponencial

- Uma função f é dita ser *exponencial* se para constantes $c_1, c_2 > 0$ e $d_1, d_2 > 1$ e um inteiro positivo k', tivermos:

$$c_1 d_1^k \leq f(k) \leq c_2 d_2^k, \text{ para todo inteiro } k \geq k'.$$

- Um exemplo de algoritmo de tempo exponencial é a enumeração completa de 2^k vetores binários k-dimensionais;
- A função 2^k não é limitada por nenhum polinômio em k e se torna demasiadamente grande para valores relativamente pequenos de k;

Complexidade computacional

▷ Algoritmos de tempo exponencial

- Uma função f é dita ser *exponencial* se para constantes $c_1, c_2 > 0$ e $d_1, d_2 > 1$ e um inteiro positivo k', tivermos:

$$c_1 d_1^k \leq f(k) \leq c_2 d_2^k, \text{ para todo inteiro } k \geq k'.$$

- Um exemplo de algoritmo de tempo exponencial é a enumeração completa de 2^k vetores binários k-dimensionais;
- A função 2^k não é limitada por nenhum polinômio em k e se torna demasiadamente grande para valores relativamente pequenos de k;
- Por exemplo, se tivermos $k = 60$ e cada operação elementar do algoritmo levar 1 microssegundo, então realizar 2^k operações levará mais de 300 séculos para finalizar, enquanto um algoritmo que requer k^5 operações finalizaria em menos de 15 minutos.

Complexidade computacional

▷ Tempos de execução de acordo com a função de complexidade

Função Complexidade	*Tamanho n* 10	20	30	40	50	60
n	0,00001 segundo	0,00002 segundo	0,00003 segundo	0,00004 segundo	0,00005 segundo	0,00006 segundo
n^2	0,0001 segundo	0,0004 segundo	0,0009 segundo	0,0016 segundo	0,0025 segundo	0,0036 segundo
n^3	0,001 segundo	0,008 segundo	0,027 segundo	0,064 segundo	0,125 segundo	0,216 segundo
n^5	0,1 segundo	3,2 segundos	24,3 segundos	1,7 minuto	5,2 minutos	13,0 minutos
2^n	0,001 segundo	1,0 segundo	17,9 minutos	12,7 dias	35,7 anos	366 séculos
3^n	0,059 segundo	58 minutos	6,5 anos	3855 séculos	2×10^8 séculos	$1,3 \times 10^{13}$ séculos
$n!$	3,628 segundos	771,4 séculos	$8,4 \times 10^{16}$ séculos	$2,5 \times 10^{32}$ séculos	$9,6 \times 10^{48}$ séculos	$2,6 \times 10^{66}$ séculos

Arenales et al. (2007)

Complexidade computacional

▷ Em termos gerais...

- Um algoritmo é rápido/eficiente quando seu tempo de execução é limitado superiormente por uma função polinomial que depende do tamanho da instância;
- Um problema é fácil quando existe um algoritmo rápido que garanta uma solução ótima.

Complexidade computacional

▷ Exemplo 1: Método simplex

- O método simplex é o grande responsável pelo sucesso da aplicação da Otimização Linear e Pesquisa Operacional;

Complexidade computacional

▷ Exemplo 1: Método simplex

- O método simplex é o grande responsável pelo sucesso da aplicação da Otimização Linear e Pesquisa Operacional;
- Praticamente todo software de otimização inclui o simplex como método padrão na resolução de problemas de programação linear;

Complexidade computacional

▷ Exemplo 1: Método simplex

- O método simplex é o grande responsável pelo sucesso da aplicação da Otimização Linear e Pesquisa Operacional;
- Praticamente todo software de otimização inclui o simplex como método padrão na resolução de problemas de programação linear;
- Na teoria, o método simplex possui tempo de execução de ordem exponencial :(
  - Klee V., Minty, G.J. (1972) How good is the simplex algorithm? In: Shisha, O. (ed) Inequalities: III. Acad. Press, New York.

Complexidade computacional

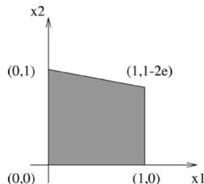
▷ https://doi.org/10.1007/978-0-387-74759-0_339

Klee–Minty Examples

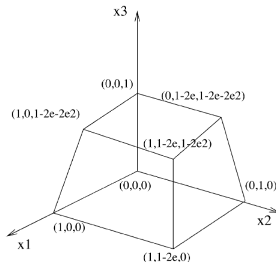
The Klee–Minty examples of order n are the linear problems of the form

$$\begin{cases} \max & \sum_{j=1}^{n} \varepsilon^{n-j} x_j \\ \text{s.t.} & x_1 \leq 1 \\ & 2\sum_{j=1}^{i-1} \varepsilon^{i-j} x_j + x_i \leq 1, \\ & \quad i = 2, \dots, n, \\ & x_j \geq 0, \quad j = 1, \dots, n, \end{cases} \tag{7}$$

where $0 < \varepsilon \leq 1/3$. In this section we will show that the feasible region of (7) is a slightly perturbed cube of dimension n, see Fig. 1 and Fig. 2. The optimal solution is $(0, 0, \dots, 1) \in \mathbf{R}^n$ and the optimal value is 1. A cube of dimension n has 2^n vertices. In the next section we will describe pivoting rules that force the simplex method to pass through all the vertices of the Klee–Minty examples. These pivoting rules require $2^n - 1$ iterations before optimality is reached and, hence, they are exponential.



Linear Programming: Klee–Minty Examples, Figure 1
Feasible region of Klee–Minty example of order $n = 2$



Linear Programming: Klee–Minty Examples, Figure 2
Feasible region of Klee–Minty example of order $n = 3$

Complexidade computacional

▷ Exemplo: Método simplex

- Por outro lado, existem algoritmos de tempo polinomial para resolver problemas de programação linear, como o Método Elipsóide e os Métodos de Pontos Interiores;

Complexidade computacional

▷ Exemplo: Método simplex

- Por outro lado, existem algoritmos de tempo polinomial para resolver problemas de programação linear, como o Método Elipsóide e os Métodos de Pontos Interiores;
- O Método Elipsóide, embora polinomial, é ineficiente na prática e nunca teve uma implementação competitiva; (*breakthrough* teórico ao mostrar que problemas de programação linear são *fáceis*!

Complexidade computacional

▷ Exemplo: Método simplex

- Por outro lado, existem algoritmos de tempo polinomial para resolver problemas de programação linear, como o Método Elipsóide e os Métodos de Pontos Interiores;
- O Método Elipsóide, embora polinomial, é ineficiente na prática e nunca teve uma implementação competitiva; (*breakthrough* teórico ao mostrar que problemas de programação linear são *fáceis*!
- Métodos de pontos interiores tem complexidade $O(\sqrt{n}L)$, $O(nL)$;

Complexidade computacional

▷ Exemplo: Método simplex

- Por outro lado, existem algoritmos de tempo polinomial para resolver problemas de programação linear, como o Método Elipsóide e os Métodos de Pontos Interiores;
- O Método Elipsóide, embora polinomial, é ineficiente na prática e nunca teve uma implementação competitiva; (*breakthrough* teórico ao mostrar que problemas de programação linear são *fáceis*!
- Métodos de pontos interiores tem complexidade $O(\sqrt{n}L)$, $O(nL)$;
- Ainda assim, o método simplex pode ter melhor desempenho que métodos de pontos interiores na prática.

Complexidade computacional

▷ Exemplo: Problema do Caixeiro Viajante

- Dado um conjunto de cidades, o Problema do Caixeiro Viajante (PCV) consiste em determinar uma rota que visite cada cidade exatamente uma vez e retorne à cidade inicial;

Complexidade computacional

▷ Exemplo: Problema do Caixeiro Viajante

- Dado um conjunto de cidades, o Problema do Caixeiro Viajante (PCV) consiste em determinar uma rota que visite cada cidade exatamente uma vez e retorne à cidade inicial;
- É extremamente simples de explicar, até uma criança consegue compreendê-lo! (já foi tema de um desafio em caixa de cereais);

Complexidade computacional

▷ Exemplo: Problema do Caixeiro Viajante

- Dado um conjunto de cidades, o Problema do Caixeiro Viajante (PCV) consiste em determinar uma rota que visite cada cidade exatamente uma vez e retorne à cidade inicial;
- É extremamente simples de explicar, até uma criança consegue compreendê-lo! (já foi tema de um desafio em caixa de cereais);
- Por outro lado é extremamente difícil de ser resolvido :(

Complexidade computacional

▷ Exemplo: Problema do Caixeiro Viajante

- Dado um conjunto de cidades, o Problema do Caixeiro Viajante (PCV) consiste em determinar uma rota que visite cada cidade exatamente uma vez e retorne à cidade inicial;
- É extremamente simples de explicar, até uma criança consegue compreendê-lo! (já foi tema de um desafio em caixa de cereais);
- Por outro lado é extremamente difícil de ser resolvido :(
- Está em uma classe de problemas para os quais não se conhece nenhum algoritmo de tempo polinomial para resolução.

Complexidade computacional

▷ Algumas observações

- Existem algoritmos que não são de tempo polinomial nem exponencial:
  - Alguns possuem função de complexidade com taxa de crescimento mais que polinomial mas menor que exponencial (p.ex. $k^{\log k}$);
  - Algumas funções têm taxa de crescimento maior que exponencial (p.ex. k^{k^k});

Complexidade computacional

▷ Algumas observações

- Existem algoritmos que não são de tempo polinomial nem exponencial:
  - Alguns possuem função de complexidade com taxa de crescimento mais que polinomial mas menor que exponencial (p.ex. $k^{\log k}$);
  - Algumas funções têm taxa de crescimento maior que exponencial (p.ex. k^{k^k});
- Embora tenhamos dado ênfase ao tempo computacional, a quantidade de memória computacional usada para resolver um problema também é importante:
  - Se $X \in \mathcal{P}$, deve existir um algoritmo para X cuja memória necessária seja uma função polinomial do tamanho da entrada (embora o contrário não seja verdadeiro).

Complexidade computacional

▷ Algumas observações

- Tempo exponencial também pode ocorrer quando o número de cálculos é em função do tamanho dos números da entrada;

Complexidade computacional

▷ Algumas observações

- Tempo exponencial também pode ocorrer quando o número de cálculos é em função do tamanho dos números da entrada;
- Seja θ o maior inteiro em uma dada instância;

Complexidade computacional

▷ Algumas observações

- Tempo exponencial também pode ocorrer quando o número de cálculos é em função do tamanho dos números da entrada;
- Seja θ o maior inteiro em uma dada instância;
- Assumindo representação binária dos dados, temos que θ é definido por uma cadeia de tamanho $O(\log \theta)$;

Complexidade computacional

▷ Algumas observações

- Tempo exponencial também pode ocorrer quando o número de cálculos é em função do tamanho dos números da entrada;
- Seja θ o maior inteiro em uma dada instância;
- Assumindo representação binária dos dados, temos que θ é definido por uma cadeia de tamanho $O(\log\theta)$;
- Assim, um algoritmo que requer θ passos é no mínimo de tempo exponencial.

- Obrigado pela atenção!
- Dúvidas?