Complexidade de Algoritmos

Mariana Kolberg

Visão Geral

- Objetivo
  - Estudar a análise e o projeto de algoritmos
- Estrutura
  - Parte I: análise de algoritmos, i.e. o estudo teórico do desempenho e uso de recursos
    - Pré-requisito para o projeto de algoritmos
  - Parte II: as principais técnicas para projetar algoritmos
    - Programação dinâmica
    - Algoritmos gulosos
    - Divisão e conquista
  - Parte III: classes de complexidade

Introdução

- Um algoritmo é um procedimento que consiste em um conjunto de regras não ambiguas as quais especificam, para cada entrada, uma sequência finita de operações, terminando com uma saída correspondente.
- Um algoritmo resolve um problema quando, para qualquer entrada, produz uma resposta correta, se forem concedidos tempo e memoria sucientes para a sua execução.

Motivação

- Teoria da computação
  - Quais problemas são efetivamente computáveis?
- Projeto de algoritmos
  - Quais problemas são eficientemente computáveis?

- Para responder, temos que saber o que eficiente signica.
  - Uma definição razoável é considerar algoritmos em tempo polinomial como eficiente (tese de Cobham-Edmonds).

Custos de algoritmos

- Qual custo é de interesse?
- Uma execução tem vários custos associados:
  - Tempo de execução, uso de espaço (cache, memoria, disco), consumo de energia, ...
  - Existem caractersticas e medidas que são importantes em contextos diferentes
    - Linhas de codigo fonte (LOC), legibilidade, manutenabilidade, corretude, custo de implementação, robustez, extensibilidade,...
  - A medida mais importante e nosso foco: tempo de execução.

Qual o melhor algoritmo?

- Um problema pode ser resolvido através de **diversos algoritmos** com complexidades diferentes;
  - Resolução de sistemas lineares de tamanho n
    - Método de Cramer precisa aprox. 6n! operações de ponto flutuante (OPF)
    - Método de Gauss precisa aprox. n^3-n OPF.

ordem	Método de Cramer	Método de Gauss
2	$22\mu s$	$50\mu s$
3	$102\mu s$	$159\mu s$
4	$456\mu s$	$353\mu s$
5	$2.35ms$	$666\mu s$
10	$1.19min$	$4.95ms$
20	15255 séculos	$38.63ms$

- O fato de um algoritmo resolver um dado problema não significa que seja aceitável na prática.

Motivação para algoritmos eficientes

Por que analisar a eficiência de algoritmos se os computadores estão cada dia mais rápidos?

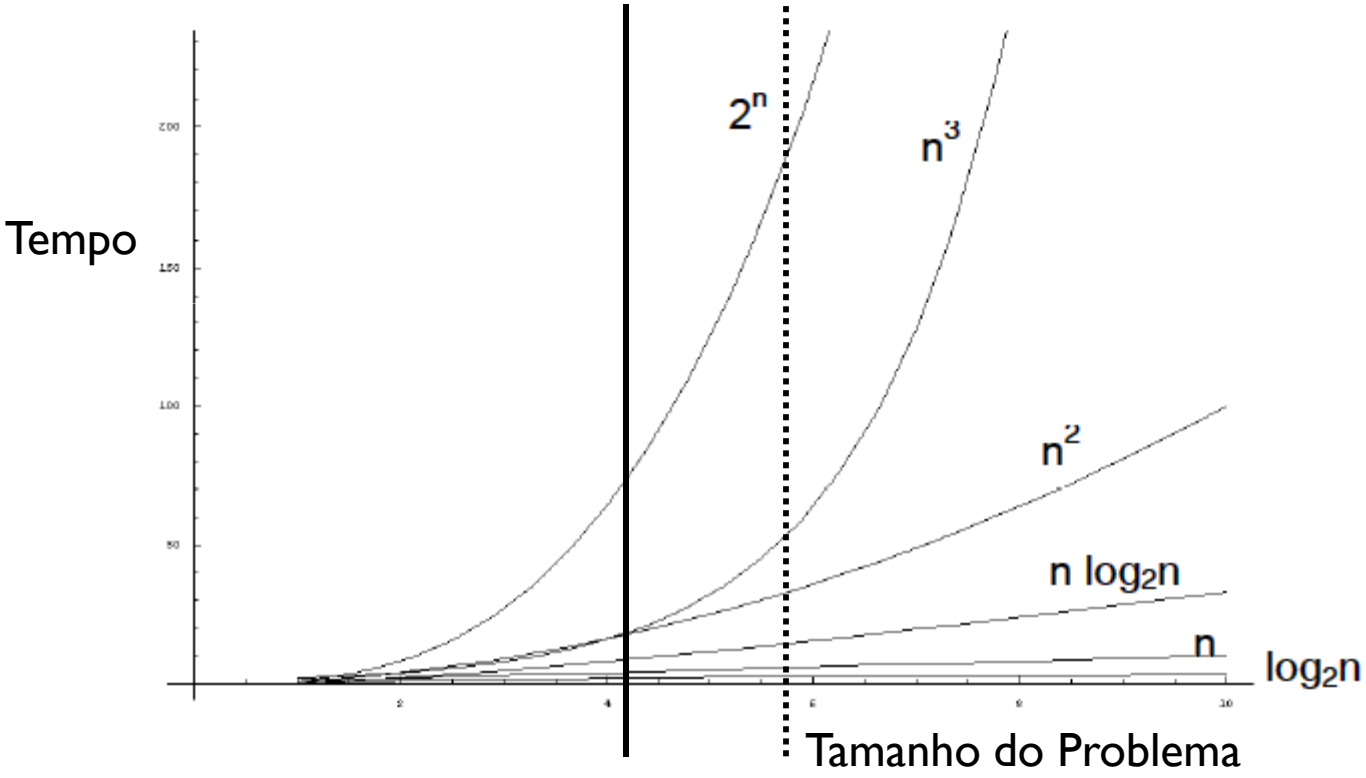
Motivação para algoritmos eficientes

- Um algoritmo ineficiente em um computador rápido não ajuda!
- Suponha que uma máquina resolva um problema de tamanho x em um dado tempo.
- Qual tamanho de problema uma maquina 10 vezes mais rapida resolve o mesmo tempo?

complexidade de tempo	máquina lenta	máquina rápida $(10x)$
$log_2 n$	x_0	x_0^{10}
n	x_1	$10x_1$
$n \log_2 n$	x_2	$10x_2$ (p/ x_2 grande)
n^2	x_3	$3.16x_3$
n^3	x_4	$2.15x_4$
2^n	x_5	$x_5 + 3.3$
3^n	x_6	$x_6 + 2.096$

Complexidade do Algoritmo x Tamanho máximo de problema resolvível

Crescimento de funções



Crescimento de funções

Tamanho da entrada

complexidade	10	100	10^3	10^4	10^5	10^6
$\log_2 n$	3	6	9	13	16	19
n	10	100	1000	10^4	10^5	10^6
$n \log_2 n$	30	664	9965	10^5	10^6	10^7
n^2	100	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{300}	10^{300}	10^{3000}	10^{300000}

1 ano = $365 \times 24 \times 60 \times 60 \approx 3 \times 10^7$ segundos

1 século $\approx 3 \times 10^9$ segundos

1 milénio $\approx 3 \times 10^{10}$ segundos

Panorama de tempo de execução

- *Tempo constante: O* (1) – mais rápido, impossível
- *Tempo sublinear:*
  - *O* (log log *n*) : super-rápido
  - *O* (log *n*) : logarítmico – muito bom
- *Tempo linear: O* (*n*) – é o melhor que se pode esperar quando é necessário examinar toda a entrada
- *Tempo n log n: O* (*n* log n) : limite de muitos problemas práticos, ex.: ordenar uma coleção de números. Comum em problemas de divisão e conquista.
- *Tempo polinomial:*
  - *O* (n^2) : quadrático
  - *O* (n^k) : polinomial – ok para *n* pequeno
- *Tempo exponencial*
  - *O* (k^n)
  - *O* (*n*!)
  - *O* (n^n)

Comparar eficiências

- Como comparar eficiências? Uma medida concreta do tempo depende
  - do tipo da maquina usada (arquitetura, cache, memoria, ...)
  - da qualidade e das opções do compilador ou ambiente de execução
  - do tamanho do problema (da entrada)
- Portanto, foram inventadas maquinas abstratas.
  - A análise da complexidade de um algoritmo consiste em determinar o número de operações básicas (atribuição, soma, comparação, ...) em relação ao tamanho da entrada.

Análise Assintótica

- O tempo de execução de um algoritmo para uma determinada entrada pode ser medido pelo número de operações primitivas que ele executa.
  - o número de operações fornece um nível de detalhamento grande.

- Portanto, analisamos somente a taxa ou ordem de crescimento, substituindo funções exatas com cotas mais simples.

Análise Assintótica

- Complexidade é também chamada esforço requerido ou quantidade de trabalho.

- Complexidade no pior caso
  - Considera-se a instância que faz o algoritmo funcionar mais lentamente;

- Complexidade média
  - Considera-se todas as possíveis instâncias e mede-se o tempo médio.

Medidas de Complexidade

- A complexidade pode ser calculada através do:
  - **Tempo de execução** do algoritmo determinado pelas instruções executadas : quanto "tempo" é necessário para computar o resultado para uma instância do problema de tamanho n;
  - **Espaço de memória** utilizado pelo algoritmo : quanto "espaço de memória/disco" é preciso para armazenar a(s) estrutura(s) utilizada(s) pelo algoritmo.

- O esforço realizado por um algoritmo é calculado a partir da quantidade de vezes que a operação fundamental é executada.
  - Para um algoritmo de ordenação, uma operação fundamental é a comparação entre elementos quando à ordem.

Comparação entre Complexidades

- A complexidade exata possui muitos detalhes
- A escolha de um algoritmo é feita através de sua taxa de crescimento
- Esta taxa é representada através de cotas que são funções mais simples.
- A ordem de crescimento do tempo de execução de um algoritmo fornece uma caracterização simples de eficiência do algoritmo.

Comparação entre Complexidades

- A complexidade por ser vista como uma propriedade do problema.
- Logo, é possível obter uma medida independente do tratamento dado ao problema ou do caminho percorrido na busca da solução, portanto independente do algoritmo.

Comparação entre Complexidades

- Imagine um algoritmo com número de operações

$$an^2+bn+c$$

- Para análise assintótica não interessam
  - os termos de baixa ordem
  - os coeficientes constantes

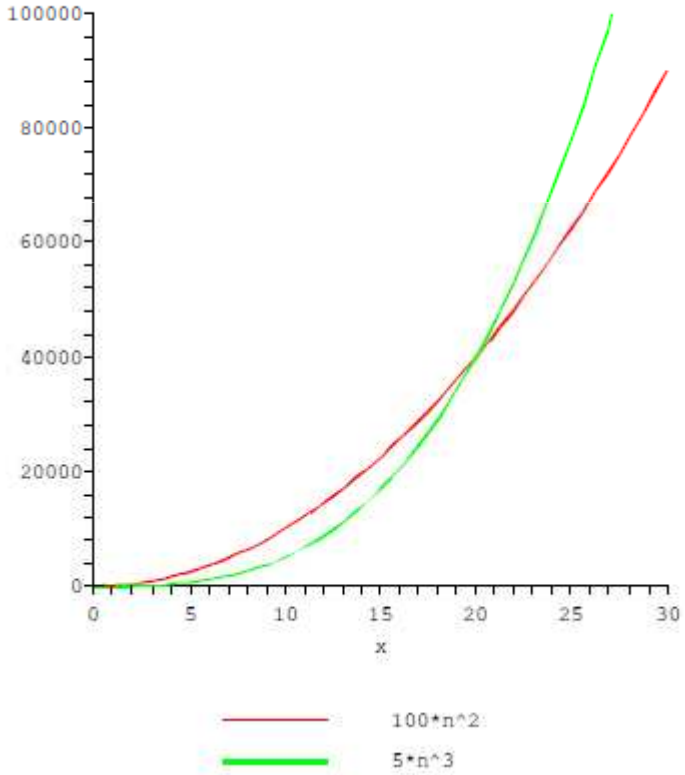
Logo o tempo de execução deste algoritmo tem cota igual a n^2, ou seja, $O(n^2)$.

Comparação entre Complexidades

- Considere dois algoritmos A e B com tempo de execução $O(n^2)$ e $O(n^3)$, respectivamente.
  - Qual deles é o mais eficiente ?
- Considere dois programas A e B com tempos de execução $100n^2$ milisegundos, e $5n^3$ milisegundos, respectivamente.
  - Qual é o mais eficiente?

Comparação entre Complexidades

- Assintoticamente consideramos um algoritmo com complexidade $O(n^2)$ melhor que um algoritmo com $O(n^3)$.
- De fato, para n sucientemente grande $O(n^2)$ sempre é melhor.
- Mas na pratica, não podemos esquecer o tamanho do problemo real.
  - tamanho n<20 , B será mais eficiente que A.
  - Se n é grande, A é mais eficiente



Comparação entre Complexidades

- Considere dois computadores :
  - C_1 que executa 10^7 instruções por segundo (10 milhões);
  - C_2 que executa 10^9 instruções por segundo (1 bilhão).
- Considere dois algoritmos de ordenação:
  - A - cujo código exige $2n^2$ instruções
  - B - cujo código exige 50nlog n instruções.
- Quanto tempo C_1 e C_2 gastam para ordenar um milhão de números usando os algoritmos A e B ?

Comparação entre Complexidades

Algoritmo	Comp. C_1	Comp. C_2
Alg A	$\frac{2.(10^6)^2 \text{instruções}}{10^7 \text{instruções/s}} = \mathbf{2.10^5 s}$	$\frac{2.(10^6)^2 \text{instruções}}{10^9 \text{instruções/s}} = \mathbf{2.10^3 s}$
Alg B	$\frac{50.10^6 \log 10^6 \text{instruções}}{10^7 \text{instruções/s}} = \mathbf{30s}$	$\frac{50.10^6 \log 10^6 \text{instruções}}{10^9 \text{instruções/s}} = \mathbf{0.3s}$

É importante analisar a complexidade de um algoritmo
para utilizar adequadamente os recursos disponíveis!!!!

Comparação entre Complexidades

- Considere dois algoritmos A e B com complexidades $8n^2$ e n^3, respectivamente.
  - Qual é o mais eficiente ?
  - Qual é o maior valor de n, para o qual o algoritmo B é mais eficiente que o algoritmo A?

Os algoritmos têm igual desempenho quando n=8.

Até n=7, B é mais eficiente que A.

Comparação entre Complexidades

Um algoritmo tem complexidade $2n^2$. Num certo computador A, em um tempo t, o algoritmo resolve um problema de tamanho 25. Imagine agora que você tem disponível um computador B 100 vezes mais rápido. Qual é o tamanho máximo do problema que o mesmo algoritmo resolve no mesmo tempo t?

- No computador A, a quantidade de instruções executadas é $2.(25)^2$
- No computador B, a quantidade de instruções executadas é $100.2.(25)^2$

- O tamanho do problema resolvido em B é $2n^2 = 100.2.(25)^2$ ➔ $n = 250$.

Problemas superpolinomiais?

- Consideramos a classe P de problemas com solucão em tempo polinomial tratável.
- NP é outra classe importante que contem muitos problemas práticos (e a classe P).
  - Não se sabe se todos possuem algoritmo eficiente.
- Problemas NP-completos são os mais complexos da classe NP:
  - Se um deles tem uma solução eficiente, toda classe tem.
- Vários problemas NP-completos são parecidos com problemas que têm algoritmos eficientes.
  - Ciclo euleriano x Ciclo hamiltoniano
  - Caminhos mais curtos x Caminhos mais longo
  - Satisfatibilidade 2-CNF x Satisfatibilidade 3-CNF

Revisão

- Logaritmos - propriedades

$\log_a(1) = 0$	
$a^{\log_a(n)} = n$	por definição
$\log_a(n \cdot m) = \log_a(n) + \log_a(m)$	propriedade do produto
$\log_a(\frac{n}{m}) = \log_a(n) - \log_a(m)$	propriedade da divisão
$\log_a(n^m) = m \cdot \log_a(n)$	propriedade da potência
$\log_a(n) = \log_b(n) \cdot \log_a(b)$	troca de base
$\log_a(n) = \frac{\log_c(n)}{\log_c(a)}$	mudança de base
$\log_b(a) = \frac{1}{\log_a(b)}$	
$a^{\log_c(b)} = b^{\log_c(a)}$	expoentes

Revisão – somatórios

Para k uma constante arbitrário temos

$$\sum_{i=1}^{n} k a_i = k \sum_{i=1}^{n} a_i$$ Distributividade

$$\sum_{i=1}^{n} k = nk$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} a_i b_j = \left(\sum_{i=1}^{n} a_i \right) \left(\sum_{j=1}^{m} b_j \right)$$ Distributividade generalizada

$$\sum_{i=1}^{n} (a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$$ Associativiade

$$\sum_{i=1}^{p} a_i + \sum_{i=p+1}^{n} a_i = \sum_{i=1}^{n} a_i$$

$$\sum_{i=0}^{n} a_{p-i} = \sum_{i=p-n}^{p} a_i$$

Revisão – séries

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$ série aritmética

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1}$$ série geométrica

se $|x| < 1$ então

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}$$ série geométrica infinitamente decrescente

$$\sum_{i=1}^{n} 2^i = 2^{n+1} - 2$$

$$\sum_{i=0}^{n} i^2 = \frac{n \cdot (n+1) \cdot (2 \cdot n + 1)}{6}$$

Revisão – indução matemática

- Importante para provar resultados envolvendo inteiros
  - Seja P(n) uma propriedade relativa aos inteiros.
  - Se P(n) é verdadeira para n=1 e
  - se P(k) verdadeira implica que P(k+1) é verdadeira
  - então P(n) é verdadeira para todo inteiro n≥1
- Para aplicarmos indução matemática deve-se:
  - Passo inicial: verificar se P(n) é verdadeira para a base n_0
  - Hipótese: assumir P(n) válida
  - Prova: provar que P(n) é valida para qualquer valor de $n \geq n_0$
  - Se os passos acima forem vericados, conclui-se que P(n) é valida para qualquer valor de $n \geq n_0$

Exercícios

- mostre que $n! \leq n^n$
- mostre que $\frac{1}{\log_a(c)} = \log_c(a)$
- Demonstre a propriedade dos expoentes
- Encontre uma fórmula alternativa para

$$\sum_{i=1}^{n} (2 \cdot i - 1)$$

  e prove seu resultado via indução matemática.
- Use indução matemática para provar que

$$\sum_{i=0}^{n-1} a \cdot q^i = \frac{a \cdot (q^n - 1)}{q - 1}$$

- Dica: theoretical computer science cheat sheet: http://www.tug.org/texshowcase/cheat.pdf