Estruturas de Dados I
com Códigos C++

Prof. Igor Machado Coelho
igor.machado@ime.uerj.br

Departamento de Informática e Ciência da Computação
Universidade do Estado do Rio de Janeiro

IME-04-10820 2016/1

Conteúdo do Curso

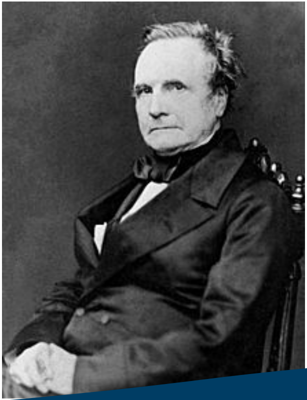
Análise de Complexidade de Algoritmos

Análise de Complexidade de Algoritmos

- A análise de complexidade de algoritmos trata da eficiência dos algoritmos
- Mas como medir eficiência de algoritmos?
- Material baseado na disciplina de ED-1 do prof. Fabiano Oliveira
- https://pt.wikipedia.org/wiki/Complexidade_computacional

Análise de Complexidade

Charles Babbage (1864) diz: "As soon as analytic engine exists, it will necessarily guide the future course of science. Whenever any result is sought by its aid, the question will raise ? By what means of calculation can these results be arrived at by this machine in the shortest time?"



Análise de Complexidade

- Considere a seguinte métrica de eficiência: em qualquer execução, o algoritmo deve responder em menos de t unidades de tempo.
- Exemplo: sistemas do Google são notáveis por manterem tempo de resposta tolerável em qualquer interação.

Análise de Complexidade

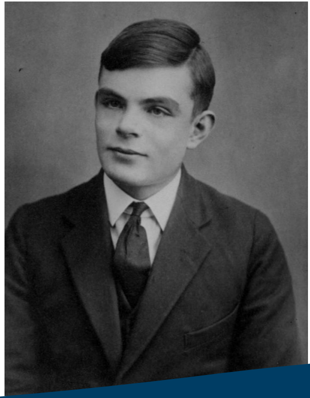
- Métrica inadequada, pois tempo de execução varia com o hardware!
- Por exemplo, o que seria da análise de complexidade se o hardware fosse infinitamente rápido e sem restrições de quantidade?

Análise de Complexidade

- Métrica inadequada, pois tempo de execução varia com o hardware!
- Por exemplo, o que seria da análise de complexidade se o hardware fosse infinitamente rápido e sem restrições de quantidade?

Análise de Complexidade

Alan Turing (1947) diz: “It is convenient to have a measure of the amount of work involved in a computing process, even if it has to be a very crude one. We may count up the number of things that various times at various elementary operations are applied in the whole process.”



Análise de Complexidade

- Consideremos, portanto, outra métrica: assumindo que cada instrução leva tempo constante, a eficiência é dada pelo número de passos de uma execução
- Podemos estudar eficiência para complexidades de **pior caso, médias, de melhor caso**, ou mesmo para instâncias de execução particulares.

Análise de Complexidade

- Assume-se, portanto, o modelo Random-Access Machine de computação com um único processador.
- Não se leva em conta hierarquia de memória (cache, memória virtual, etc.)
- **Instruções que não dependam do dado sendo processado contam como um passo; caso contrário, contam como uma função do tamanho do dado.**

Análise de Complexidade

- Vantagens do modelo
- Independência de arquitetura
- Abstrai razoavelmente bem o comportamento geral de uma máquina real

Exemplo: Algoritmo de Ordenação

```
procedimento Ordenar(B(), N: Inteiro)
    //Assume: N ≤ |B|
    //Garante: B(i) ≤ B(i+1), para todo 1 ≤ i < N
    var i, j, t: Inteiro

    para i = N até 1 passo -1 faça
        para j = 1 até i -1 faça
            se B(j) > B(j+1) então
                t ← B(j)
                B(j) ← B(j+1)
                B(j+1) ← t
```

Análise de Complexidade

procedimento Ordenar(B(), N: Inteiro)
//Assume: N ≤ |B|
//Garante: B(i) ≤ B(i+1), para todo 1 ≤ i < N

var i, j, t: Inteiro	
para i = N **até** 1 **passo** -1 **faça**	= N+1 execuções
para j = 1 **até** i -1 **faça**	= N(N+1)/2 execuções
se B(j) > B(j+1) **então**	= N(N-1)/2 execuções
t ← B(j)	≤ N(N-1)/2 execuções
B(j) ← B(j+1)	≤ N(N-1)/2 execuções
B(j+1) ← t	≤ N(N-1)/2 execuções
	≤ $(5/2)N^2-(1/2)N+1$ passos

Análise de Complexidade

- Portanto, o algorimo Ordenar executa no máximo $(5/2)N^2 - (1/2)N + 1$ passos.
- Procedendo da mesma maneira com outro algoritmo, podemos comparar qual tem a melhor eficiência.

Análise de Complexidade

- Considere uma outra implementação de ordenação e suponha que a complexidade desta implementação particular seja 10 N log N + 1000 N + 1000000
- Qual das duas funções você escolheria utilizar?

Análise de Complexidade

N	Tempo 1.a Ordenação (1000 instruções/ms)	Tempo 2.a Ordenação (10 instruções/ms)
1	~ 0,003 ms	~ 2 min
10	~ 0,2 ms	~ 2 min
100	~ 25 ms	~ 2 min
1000	~ 2,5 s	~ 4 min
10000	~ 4 min	~ 18 min
100000	~ 7 h	~ 3 h
1000000	~ 29 dias	~ 2 dias
10000000	~ 8 anos	~ 12 dias

Análise de Complexidade

- Note que o grau do polinômio é o que realmente conta a medida que a variável do polinômio cresce!
- Esta complexidade é chamada de assintótica
- Ela é um dos critérios mais utilizados para comparação de eficiência de algoritmos

Análise de Complexidade

- Motivado por isto, existe uma notação especial que evidencia a essência da complexidade, descartando as constantes multiplicativas e os termos do polinômio de graus menores – a notação O (e família).
- https://pt.wikipedia.org/wiki/Grande-O

Intuição da notação O

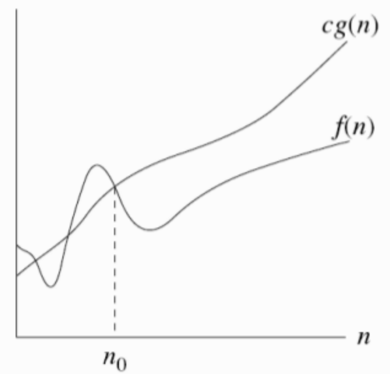
- Suponha uma função T(N) desconhecida. Veja exemplos de afirmações sobre o termo de maior crescimento assintótico

Termo de maior crescimento	Notação	Funções T(N) que satisfazem notação	Funções T(N) que NÃO satisfazem notação
Igual a N^2	$\theta(N^2)$	$\frac{1}{2} N^2 + 10N$; $100N^2$	$10N$; $N^3 + 2N^2$; $N \log N$
No máximo N^2	$O(N^2)$	$100N^2$; 100; $N \log N$	$N^3 + 2N^2$; $N^2 \lg N$
No mínimo N^2	$\Omega(N^2)$	$100N^2$; $N^3 + 2$; $N^2 \lg N$	$10N$; 100; $N \log N$
Menor que N^2	$o(N^2)$	$10N$; 100; $N \log N$	$100N^2$; $N^3 + 2N^2$; $N^2 \lg N$
Maior que N^2	$\omega(N^2)$	$N^3 + 2N^2$; $N^2 \lg N$	$100N^2$; $10N$; 100; $N \log N$

Notação O

$O(g(n)) = \{ h(n) \mid \exists$ constantes positivas c, n_0 tais que $0 \leq h(n) \leq cg(n),\ \forall\ n \geq n_0\}$

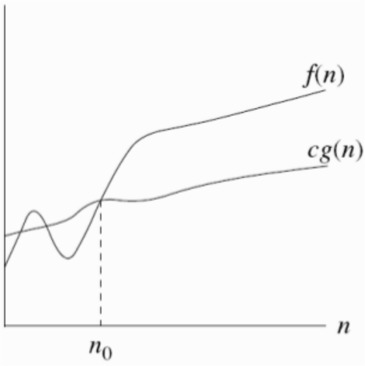
Pode-se denotar
$f(n) \in O(g(n))$ por
$f(n) = O(g(n))$



Notação Ω

$\Omega(g(n)) = \{ h(n) \mid \exists$ constantes positivas $c>0$, n_0 tais que $0 \leq cg(n) \leq h(n),\ \forall\ n \geq n_0 \}$

Pode-se denotar
$f(n) \in \Omega(g(n))$ por
$f(n) = \Omega(g(n))$



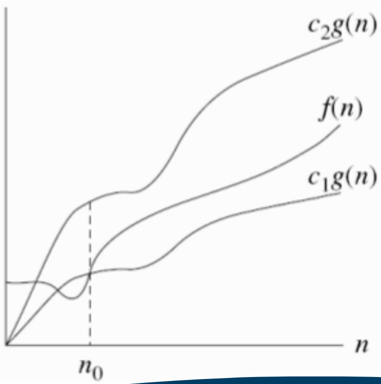
Notação Θ

$\theta(g(n)) = \{ h(n) \mid \exists$ constantes positivas $c_1 > 0$, c_2, n_0 tais que $0 \leq c_1 g(n) \leq h(n) \leq c_2 g(n)$, $\forall\ n \geq n_0\}$

Pode-se denotar

$f(n) \in \theta(g(n))$ por

$f(n) = \theta(g(n))$



- Notação o
  $o(g(n)) = \{ h(n) \mid \forall$ constante positiva $c > 0$, $\exists$ constante positiva n_0 tais que $0 \leq h(n) < cg(n)$, $\forall\ n \geq n_0$ $\}$
  Pode-se denotar $f(n) \in o(g(n))$ por $f(n) = o(g(n))$

- Notação ω
  $\omega(g(n)) = \{ h(n) \mid \forall$ constante positiva $c > 0$, $\exists$ constante positiva n_0 tais que $0 \leq cg(n) < h(n)$, $\forall\ n \geq n_0$ $\}$
  Pode-se denotar $f(n) \in \omega(g(n))$ por $f(n) = \omega(g(n))$

Análise de Complexidade

- Há um processo prático para se chegar à notação assintótica de uma função?

Para notação Θ

- Elimine as constantes multiplicativas
  - $10n^2 = \Theta(n^2)$
- Para cada variável, elimine todos os termos exceto aquele de maior crescimento assintótico
  - $3n^2 + 100n = \Theta(n^2)$
  - $10n + 100lgn + 1?2nlgn = \Theta(nlgn)$
  - $3n^2 - 90n + 2m^2 = \Theta(n^2 + m^2)$
- Elimine um termo positivo se existe outro termo positivo de maior crescimento assintótico
  - $3n^2 + 4mn + 2m^2 = \Theta(n^2 + m^2)$

Para demais notações, suponha $f(n) = \Theta(g(n))$

- $f(n)$ será $O(g(n))$ e também $O(h(n))$ para $h(n)$ assintoticamente maior que $g(n)$
  - $n^2 + n = O(n^2)$, $n^2 + n = O(n^3)$, $n^2 + n = O(n^4)$
  - $1?2nlgn = O(nlgn)$, $1?2nlgn = O(n^2)$
- $f(n)$ será $\Omega(g(n))$ e também $\Omega(h(n))$ para $h(n)$ assintoticamente menor que $g(n)$
  - $n^2 + n = \Omega(n^2)$, $n^2 + n = \Omega(n)$, $n^2 + n = \Omega(1)$
  - $1?2nlgn = \Omega(nlgn)$, $1?2nlgn = \Omega(n)$, $1?2nlgn = \Omega(lgn)$
- Analogamente, para notações o e ω, f(n) será $o(h(n))$ e f(n) será $\omega(h(n))$.

Análise de Complexidade

Função	Complexidade
$2N^2 + 1000N + 100000$	$\theta(N^2)$, $O(N^3)$, $o(N^3)$, $\Omega(N)$, $\omega(N)$
$10N + N \lg N + 20M - 10$	$\theta(N \lg N + M)$
$(5/3)\, 2^{N/2} + (3/4)N^{100}$	$\theta(2^{N/2})$
$2 N \lg N + 3 MV + 4V^2$	$\theta(N \lg N + MV + V^2)$
$(N + M)^2$	$\theta(N^2 + M^2)$

Análise de Complexidade

- Note que, na análise da ordenação, computamos o menor número de passos e o maior número de passos em que o algoritmo executa para uma entrada de tamanho N (neste caso, um vetor de N posições, que é a variável que influencia no número de passos deste algoritmo). Estas complexidades são chamadas respectivamente de **pior caso** e **melhor caso**.
- Podemos ainda fazer a análise de **caso médio**, onde consideramos a média do número de passos para uma entrada de tamanho N.

Análise de Complexidade

- Formalmente, seja U o conjunto de entradas com entradas de tamanho N e T(E) o número de passos em que o algoritmo executa com entrada $E \in U$.
- **Pior caso:** $\max \; \mathrm{T(E)} : E \in U$
- **Melhor caso:** $\min \; \mathrm{T(E)} : E \in U$
- **Caso médio:** $\sum \mathrm{p(E)T(E)} : E \in U$ onde p(E) é a probabilidade de ocorrência da entrada E

Análise de Complexidade

- Suponha um algoritmo cuja função que conta o número de passos no pior caso seja $N^3 + N$ e no melhor caso seja 1?2 N - 10. São afirmações precisas:
- Qualquer entrada executa em $O(N^3)$ e em $\Omega(N)$ passos; $\Theta(N^3)$ ou $\Theta(N)$ é errado!
- No pior caso, o algoritmo executa em $\Theta(N^3)$ passos; $O(N^3)$ não é preciso!
- No melhor caso, o algoritmo executa em $\Theta(N)$ passos; $\Omega(N)$ não é preciso!
- No caso médio, o algoritmo executa em $O(N^3)$ e em $\Omega(N)$ passos; $\Theta(N^3)$ ou $\Theta(N)$ é errado!

Análise de Complexidade

Dicas:

- Tenha em mente qual caso esta analisando;
- Descreva um exemplo para seguir;
- Analise a complexidade para cada sub-parte do seu algoritmo (em blocos!)

Topico extra

Teorema Mestre

- O Teorema Mestre fornece uma solucao em termos assintoticos para relacoes de recorrencia, muito comuns em problemas de divisao e conquista.
- https://en.wikipedia.org/wiki/Master_theorem

Teorema Mestre

$$T(N) = aT(\frac{N}{b}) + f(n), \;\; a \geq 1 \; , b > 1$$

- n e o tamanho do problema
- a e o numero de subproblemas na recursao
- n/b e o tamanho do subproblema
- f(n) e o custo da operacao efetuada para cada recursao

Análise de Complexidade

Teorema Mestre (Caso 1)

$$T(N) = aT(\frac{N}{b}) + f(n), \ \ a \geq 1 \, , b > 1$$

- Se $f(n) \in O(n^c)$, onde $c < log_b \ \ a$
- Entao $T(N) \in \Theta(n^{log_b \ \ a})$
- Exemplo: $T(N) = 8T(N/2) + 1000N^2$
- $T(N) \in \Theta(n^3)$ (VERIFIQUE!)

Teorema Mestre (Caso 2)

$$T(N) = aT(\frac{N}{b}) + f(n), \quad a \geq 1\,, b > 1$$

- Se $f(n) \in \Theta(n^c log^k \ n)$, onde $k \geq 0$, $c = log_b \ a$
- Entao $T(N) \in \Theta(n^c log^{k+1} \ n)$
- Exemplo: $T(N) = 2T(N/2) + 10N$
- $T(N) \in \Theta(nlogn)$ (VERIFIQUE!)

Teorema Mestre (Caso 3)

$$T(N) = aT(\frac{N}{b}) + f(n), \quad a \geq 1\,, b > 1$$

- Se $f(n) \in \Omega(n^c)$, onde $c > log_b \;\; a$
- Entao $T(N) \in \Theta(f(n))$
- Exemplo: $T(N) = 2T(N/2) + N^2$
- $T(N) \in \Theta(N^2)$ (VERIFIQUE!)

Análise de Complexidade

- Qual o formato da busca binaria?
- E da travessia em uma arvore binaria?
- MergeSort?

Bibliografia

Szwarcfiter, J.L; Markenzon, L. Estruturas de Dados e seus Algoritmos. Rio de Janeiro, LTC, 1994.

Bibliografia Adicional:

- Cerqueira, R.; Celes, W.; Rangel, J.L. Introdução a estruturas de dados: com técnicas de programação em C. Editora, 2004.
- Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein Algoritmos: Teoria e Prática. Ed. Campus, 2002.
- Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. Introduction to Algorithms, 3rd ed.. The MIT Press, 2009.
- Preiss, B.R. Estruturas de Dados e Algoritmos Ed. Campus, 2000;
- Knuth, D.E. The Art of Computer Programming - Vols I e III. 2nd Edition. Addison Wesley, 1973.
- Graham, R.L., Knuth, D.E., Patashnik, O. Matemática Concreta. Segunda Edição, Rio de Janeiro, LTC, 1995.